Cuaderno de prácticas de Aprendizaje Automático.

Correspondiente a las titulaciones de:
Grado en Ingeniería Informática y
Doble Grado de Informática y Matemáticas
Curso 2016-2017.

Profesor de teoría:
Perez de la Blanca N., email: nicolas@decsai.ugr.es
Profesora de prácticas:
Acid S., email: acid@decsai.ugr.es

Dpto. Ciencias de la Computación e Inteligencia Artificial
E.T.S. de Ingenierías Informática y de Telecomunicación. Universidad de Granada

Contents

Unidad didáctica 1

Antes de comenzar

Para el desarrollo de las prácticas vamos a utilizar herramientas “Open Source". En concreto usaremos “R", un entorno software gratuito para el cálculo estadístico y gráficos. En esta sección describimos como instalar R y RStudio en un ordenador particular, pero también en una unidad USB que nos permitirá trabajar en cualquier máquina con sistema operativo windows, sin necesidad de hacer una instalación en cada una de ellas.

1.1 INSTALACIÓN DE R

1.1.1 Descarga de archivos

Desde el navegador accedemos a la siguiente url :`https://www.r-project.org`. Y desde allí pinchamos en “Download CRAN → Spain” y elegimos uno de los cuatro servidores disponibles.

Seguidamente pinchamos es “Download R for Windows → base → Download R-3.3.1 for Windows” para descargar el fichero “R-3.3.1-win.exe” en nuestro ordenador. Aunque las imágenes se corresponden a una versión anterior.

1.1.2 Instalación

Ejecutamos el fichero descargado y seguimos las instrucciones hasta la selección de “Carpeta de destino". Aquí podremos elegir si queremos instalar R en nuestro ordenador personal o en una unidad USB.

Como en la figura 1.1, escribimos la ruta `X:\R-3.3.1`, donde `X:` será la unidad correspondiente al disco duro si queremos instalar R en nuestro ordenador personal o será la letra correspondiente a la unidad USB previamente conectada.

El siguiente paso es la “Selección de componentes” a instalar, donde activaremos las opciones

- Core files
- Message translations
- 32 o 64 bits dependiendo de nuestro sistema operativo

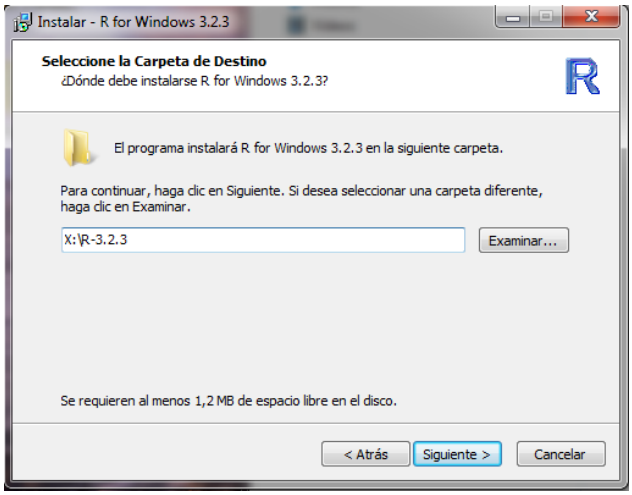

Figure 1.1: Captura de pantalla de selección de "Carpeta de Destino".

En "Opciones de Configuración" mantendremos las opciones por defecto seleccionando la opción "No".

En la "Carpeta del Menú de Inicio", activaremos la pestaña "No crear una carpeta en el menú de Inicio".

Finalmente en "Tareas adicionales" desactivamos todas las opciones excepto "Asociar archivos .RDATA con R", como se indica en la figura 1.2

1.1.3 Creando el directorio de trabajo

A continuación crearemos un directorio de trabajo con el nombre "work". Si hemos optado por la instalación en ordenador personal, este directorio podrá estar ubicado donde se desee. Sin embargo, para la instalación en unidad USB, deberá estar ubicado en el directorio raiz de la misma.

1.1.4 Creando Acceso Directo

Ahora necesitamos crear un acceso directo para lanzar R cuando lo deseemos. Si hemos optado por la instalación en ordenador personal podemos crear el acceso directo en el Escritorio, si no, debemos crearlo en el directorio raiz de la unidad USB. Para ello pinchamos en el botón derecho del ratón y seleccionamos "Nuevo → Acceso directo", y especificamos la ruta:

`X:\R-3.3.1\bin\x64\Rgui.exe` para la versión de 64 bits, o

`X:\R-3.3.1\bin\i386\Rgui.exe` para la versión de 32 bits.

Para el nombre del acceso directo podemos utilizar "R".

1.1.5 Configuración del directorio de trabajo

Ahora pinchamos en el botón derecho del ratón sobre el acceso directo creado, y accedemos al menú de "Propiedades". En la ruta "Iniciar en:" indicamos la ruta del directorio de trabajo.

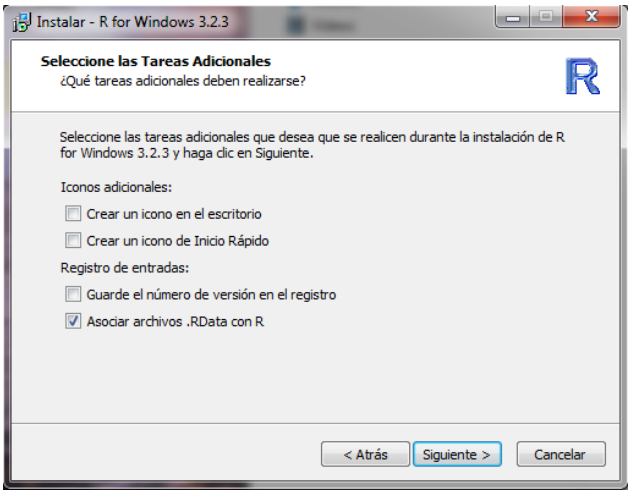

Figure 1.2: Captura de pantalla de selección de "TareasAdicionales".

1.1.6 Comprobando instalación

Finalmente ejecutamos el acceso directo y en la consola escribimos el comando "getwd()", que nos debe devolver el directorio de trabajo que le hemos indicado.

1.2 INSTALACIÓN DE RSTUDIO

Rstudio es un IDE ("Integrated Development Envelopment") que hace más fácil y productiva la programación en R, el cual incluye: Consola, un Editor que reconoce la sintaxis de R, herramientas de dibujo y manejo de historial y visualizador del espacio de variables.

1.2.1 Descargando datos

Desde el navegador accedemos a la siguiente url :`https://www.rstudio.com/products/rstudio/download`. Como vemos en la figura 1.3, aquí podemos elegir entre descargar un instalador para Windows, o el formato .zip. En nuestro caso elegiremos éste último.

1.2.2 Instalación de RStudio

Una vez descargado el fichero "RStudio-0.99.903.zip", creamos el directorio `''X:\RStudio''`, y descomprimimos el contenido del fichero descargado en el directorio.

1.2.3 Creando Acceso Directo

Ahora necesitamos crear un acceso directo para lanzar RStudio cuando lo deseemos. Si hemos optado por la instalación en ordenador personal podemos crear el acceso directo en el Escritorio, si no, debemos crearlo en el directorio raiz de la unidad USB. Para ello pinchamos en el botón derecho del ratón y seleccionamos "Nuevo → Acceso directo", y especificamos la ruta:

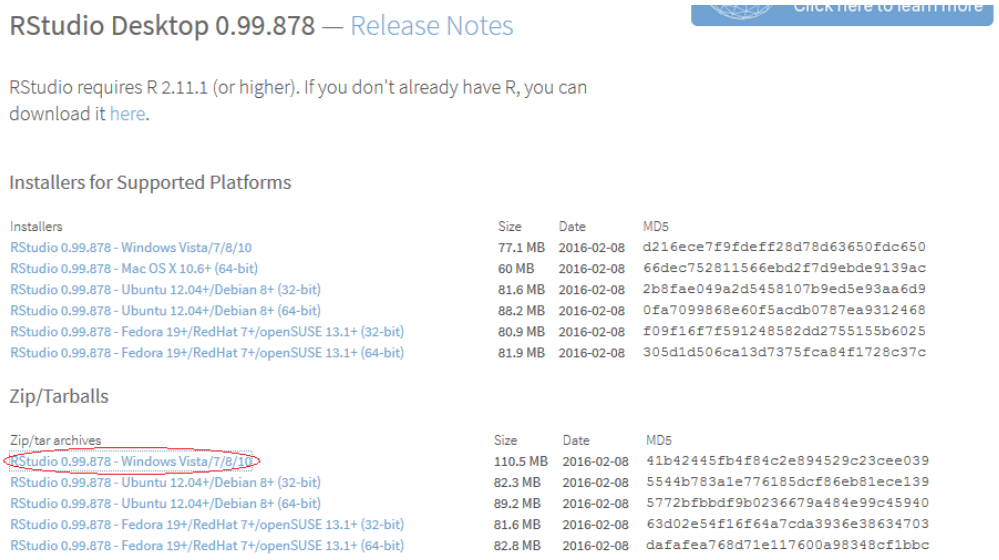

Figure 1.3: Captura de pantalla de selección de "RStudio".

`X:\RStudio\bin\rstudio.exe` Para el nombre del acceso directo podemos utilizar "RStudio".

1.2.4 Configuración del directorio de trabajo

Ahora pinchamos en el botón derecho del ratón sobre el acceso directo creado, y accedemos al menú de "Propiedades". En la ruta "Iniciar en:" indicamos la ruta del directorio de trabajo.

1.2.5 Comprobando instalación

Finalmente ejecutamos el acceso directo y en la consola escribimos el comando "getwd()", que nos debe devolver el directorio de trabajo que le hemos indicado.

Unidad didáctica 2

Lo esencial del lenguaje R

2.1 Tipos básicos

Valores reales y enteros, son **numeric**; 4, y 4.0
Valores booleanos (TRUE o FALSE) son **logical**, abreviado como (T o F)
Valores de texto (o string), caracter son **characters**; "GR0017AF".
Para escribir **expresiones** numéricas :$+, -, *, /,$' %% (módulo)
La asignación `<-` o bien `=`
Operadores **relacionales** `==, !=, <, <=, >, >=`

```
> a <- 3.5^3
```

2.1.1 Pedir ayuda

Queremos lanzar un dado estándar, esto es generar aleatoriamente números del 1 al 6. La función a usar es **sample()** todo lo que usted quiso saber y nunca se atrevió a preguntar sobre sample

```
> help(sample)
> ?sample            # comando abreviado
```

Aunque hay otros que nos pueden ayudar a entender mejor un comando como son:

```
> apropos("sample")   # da varios comandos relacionados con
> args(sample)        # muestra los parametros formales obligatorios y x defecto
> example(sample)     # se visitan ejemplos de uso
```

Ejercicio 2.1 *Muestre el lanzamiento de 1 dado (de 6 caras).*

Ejercicio 2.2 *Muestre 10 lanzamientos de 1 dado.*

Ejercicio 2.3 *Muestre el lanzamiento de 2 dados.*

Ejercicio 2.4 *Cargue el paquete* **ISLR***, consulte la lista de datasets que contiene. Consulte la descripción de los datos Auto, Carseats y Hitters. Idem sobre el dataset Boston, ubicado dentro de* **MASS***.*

2.2 Ejecutar comandos de R

Desde línea de comandos

```
> -15:15            # crea una secuencia de enteros, la salida se muestra en 2 líneas
```

Desde un fichero

Otra alternativa, recomendada, para trabajar es editar y lanzar ficheros de comandos que se pueden guardar y volver a relanzar desde R. Estos ficheros de comandos R se pueden crear con cualquier editor, que deben tener extensión `.R`.

Para ejecutar los comandos de R almacenados en un fichero, `source("nombrefichero.R")` se puede mostrar echo de los comandos utilizando un parámetro adicional, `echo=T`.

Ejercicio 2.5 *Guardar los ejercicios de la sesion de prácticas, en el fichero sesion01.R*

2.3 Las secuencias

Se crean considerando que son enteros y el incremento o decremento es 1

Ejemplo 2.1
```
> -12:3
> 3:-12
```

`:` es una abreviacion del comando `seq()`, para secuencias más particulares.

Ejemplo 2.2
```
> seq(-12,3)
> ?seq #pedir ayuda

seq(from = 1, to = 1, by = ((to - from)/(length.out - 1)),
         length.out = NULL, along.with = NULL, ...)
```

Ejemplo 2.3 `> seq(-1, 2, by = 0.1)`

Ejercicio 2.6 *crear una secuencia de 1 a 1000 en 4 intervalos*

2.4 Vectores

Las tres funciones más habituales para crear vectores son: `seq(), c()` y `rep()`. (secuenciar, concatenar y replicar respectivamente.)

de un vector nos pueden interesar concer, longitud, tipos de componentes, tipo genérico de componentes `length(), mode(), class()` .

Ejemplo 2.4
```
> xx<- c(x,0,x)
> length(xx)
```

Ejercicio 2.7 *Cree un vector de 10 componentes enteras decrecientes denominado s. Compruebe los atributos que tiene el sistema sobre los tipos de los vectores x y s.*

`rep()` como replicar un vector, sin for...

Ejemplo 2.5
```
> rep(x,2)
> rep(x,each=2)
```

Se puede replicar con un factor diferente

Ejemplo 2.6
```
 > u <- rep(1 : 2, c(8, 4)) # 1, 8 veces y 2,  4 veces
```

Ejercicio 2.8 *Cómo reproducir mediante algunas de las funciones vistas los siguientes vectores:*

1. *los valores: 1, 2, 3, 4, 1, 2, 3, 4, 1, 2, 3, 4, 1, 2, 3, 4*
2. *los valores: 4, 4, 4, 4, 3, 3, 3, 3, 2, 2, 2, 2, 1, 1, 1, 1*
3. *los valores: 1, 2, 2, 3, 3, 3, 4, 4, 4, 4, 5, 5, 5, 5, 5*

2.4.1 Cómputos sobre vectores

Ejemplo 2.7 `> x+1 # suma 1 a cada uno de los componentes del vector x`
`> 2*y`

Las operaciones anteriores en realidad se hace entre vectores. Un valor escalar, vector de longitud 1 se repite tantas veces como necesario hasta alcanzar la longitud del vector mayor.

Operadores elementales:
`+, -, *, / , ^` (potencia)
y, **funciones matemáticas:**
`abs(), log(), exp(), sin(), cos(), tan(), sqrt()`

2.4.2 Visitando las operaciones aritméticas sobre vectores

Ejercicio 2.9 *Estima las longitudes resultantes de las operaciones siguientes:*

Ejemplo 2.8
```
> x + 1 + 2 * y
> x - y
> x + s
> xx + x
```

Ejemplo 2.9
```
> zz <-  xx + x    # A pesar del warning se crea el nuevo vector
```

Ejercicio 2.10 *Con los siguientes números: 7.3, 6.8, 0.005, 9, 12, 2.4, 18.9, .9*

1. *Calcula la media.*
2. *Calcula la raíz cuadrada de los números.*
3. *Obtén los números que son mayores que su raíz cuadrada.*
4. *Cuántos valores son mayores que 1?*
5. *Obtén la raíz cuadrada de los núumeros redondeados con dos cifras decimales.*
6. *Cuánto difieren los números redondeados de los originales*

Pista: consulte la función **round()**

funciones útiles sobre vectores:
`min()`, `max()`, `range()`, `sum()`, `prod()`, `sort()` para hallar respectivamente el mínimo, máximo, el par mínimo máximo, sumatoria y producto de componentes y finalmente, realiza ordenación.

Ejercicio 2.11 *Deje ordenado x en orden decreciente.*

Ejercicio 2.12 *Calcule la media del vector x, usando* `sum()` *y* `length()`.

Qué hace la siguiente expresión ?

```
> sum((x - mean(x))^2)/(length(x)-1)
```

Funciones de dispersion :
`mean, var, sd ...`

Ejercicio 2.13 *Se registran ventas de 5 artículos el sábado, se registran en otro vector las ventas del domingo. Guarde en un vector articulo.nombres los nombres de los artículos correspondientes a las ventas. Obtener en un vector* `finde.ventas` *las ventas del fin de semana.*

Ejercicio 2.14 *Simule el 10 lanzamientos de 2 dados. Los lanzamientos son independientes.*
Nota: utilice la función `sample`*, necesitará algún parámetro, cuál?*

2.4.3 Acceso a las componentes

Los índices de los vectores van de `1..length(<vector>)`

Ejemplo 2.10

```
> x[1] # accede al primer componente del vector
> x[2:lenght(x)-1]   # todos los componentes excepto el primero y ultimo
```

Rellena los vectores siguientes,

Ejercicio 2.15 `x`

```
[1]  3  5  7  9 11 13 15
y
[1]  2  4  6  8 10 12 14
```

al vector x se le suma la cantidad 6 a todos los componentes, realizar la suma, resta y multiplicación de los 2 vectores

2.4.4 Imponiendo condiciones entre [ ]

Ejemplo 2.11 `> x[x>4]  # muestra los elementos que hacen verdad condicion`

```
> x > 4     # vector logico que sirve para indexar
```

Ejercicio 2.16 *Dado un vector de puntuaciones se han de quitar los valores extremos*

Ejercicio 2.17 *Convierta el vector* $y = c(-10 : 10)$ *en su valor absoluto:* `y=abs(y)` *Restricción: utilizando los* `[ ]` *!!!.*

2.5 Valores nulos o NA (not available) presente en cualquier tipo

Ejemplo 2.12 `> w <- numeric()  # crea el objeto y reserva espacio`

Ejemplo 2.13

```
> length(w)<- 10; w  # se redimensiona y muestra contenido
> w[1:3] <- 2:6           # o se rellena a medias??? ver contenido
```

Ejercicio 2.18 *Para cada uno de los datasets Auto, Carseats y Hitters, compruebe cuántas muestras tienen y cuántos atributos predictores. Comprueba si tienen datos perdidos.*

Ejercicio 2.19 *Se dispone de un fichero de datos llamado* `iris_miss.csv` *se ha de leer y cargar los datos ya limpios. Se puede usar con* `read.csv()` *o bien utilizando*
`Tools > Import dataset > From text file` . **Pista** *: Los datos perdidos en el fichero están notados con '?' se ha de indicar a la hora de cargar los datos que el string '?' se corresponde con un NA. El resultado es un dataframe tal como:*

```
%iris.limpio= na.exclude(iris_miss)
 > dim(iris.limpio)
[1] 137   5
```

2.6 Matrices

Ejercicio 2.20 `Rellena la matriz siguiente`

```
     [,1] [,2] [,3] [,4]
[1,]    1    5    9   13
[2,]    2    6   10   14
[3,]    3    7   11   15
[4,]    4    8   12   16
```

Mostrar primera fila, mostrar diagonal principal. Calcular sumatoria total

Ejercicio 2.21 *Cambie el valor de la esquina superior izda a -1 y los valores de la tercera fila a 1.*

Para acceder a patrones más complejos se puede usar una matriz para indexar otra

Ejercicio 2.22 *Asigne el valor -2 a la diagonal principal de la matriz* `a`.
Pista: *utilizar la* **rep()**.

Ejercicio 2.23 *Dado una matriz cuadrada inicializada al valor de la fila, se quiere poner a 0 los elementos (1,3) (2,2) y (3,1).*
Pista: *Se guarda en una matriz bidimensional las coordenadas a cambiar.*

2.7 Factores

Ejemplo 2.14 *Crear un factor de 30 elementos con tres categorías (1, 2 y 3). Etiquetar las categorías como Granada, Sevilla y Malaga. Utilizar la función* `table()` *para comprobar que realmente hay 10 de cada categoría. Pistas:* `sample, levels, as.factor`

```
ciudades <-as.factor(sample(rep(1:3, times=10)))
levels(ciudades)<-c("Granada", "Sevilla", "Malaga")
table(ciudades)
```

Ejercicio 2.24 *Construir un factor f con los niveles 1, 2, 3, 4 y 5, que tengan las siguientes frecuencias: 10, 20, 30, 40 y 50, respectivamente, use la función* **sample()** *para que se rellene de forma aleatoria. Compruebe con* **table()** *que se ha creado bien. Utilizar la función levels para asignar etiquetas a cada nivel forzar a que sea ordenado.*

Ejercicio 2.25 *En algunas ocasiones queremos discretizar una variable continua en categorías, para ello utilizaremos la función* **cut()**.
Simular 100 valores de una $N(0,1)$ *y dividir los valores en 5 categorías.*
Pista : `y<-cut(rnorm(100),breaks=5)`

2.8 Gráficos

Ejercicio 2.26 *Guardar en un fichero* **boxplotSample.pdf** *el gráfico correspondiente a las siguiente salida de la instrucción:*

```
rnorm(50),rnorm(100),rlnorm(50)
```

Pista : *abrir dispositivo pdf, realizar gráfico y finalmente cerrar dispositivo gráfico.*

Ejercicio 2.27 *Representar en un diagrama de quesos,* `pie` *la misma información que la que se muestra en con el siguiente comando.*

```
x = rnorm(100)
hist(x, main="Histograma", breaks=10)
```

Pista : *se puede guardar los cálculos del histograma sin mostrarlo en una variable con el parámetro* `plot=F`*.*

2.9 Funciones

Ejemplo 2.15 *Vamos a realizar Construir una función que ordene un vector de forma creciente o decreciente según el parámetro que se le pase como argumento. Además comprobamos que se pueda aplicar*

```
ordena <- function(vector, ordenascen=T){
if (!is.numeric(vector))
    stop("El argumento debe ser un vector numerico")
sort(vector, decreasing = !ordenascen)
}
```

Ejercicio 2.28 *Construir una función que convierta una unidad métrica inglesa a sus equivalencias en el sistema métrico decimal devolverlo en una lista.* **Pista:** *los argumentos se introducen por nombre ejemplo de llamada:* `convierte(milla=0.5)`*.*

Para el cálculo, utilice la siguiente tabla de conversión del sistema métrico:

1 pulgada = 25,4 milímetros
1 pie = 30,48 centímetros
1 yarda = 0,9144 metros
1 milla = 1609,344 metros

2.10 La familia de los apply

Vamos a calcular la media de cada una de las columnas numéricas del dataframe `iris_miss` (sin considerar valores perdidos). Para ello se supone que tenemos el dataframe cargado,2.19.

Ejemplo 2.16
```
> iris_miss <- read.csv("~/Docencia/AAprendizaje/iris_miss.csv",
header=FALSE, na.strings="?")
> sapply(iris_miss[1:4], mean, na.rm=T)
      V1       V2       V3       V4
 5.851351 3.066897 3.780272 1.187671
```

Se usa solo un subconjunto de columnas pues, la clase no es numérica. `sapply` *Devuelve un vector con las medias de cada columna.*

Ejercicio 2.29 *Queremos conocer mínimo y máximo de cada columna de dataframe anterior en las mismas condiciones.* **Pista :** *usar* `sapply` *y* `range` *devuelve una matriz pues range*

Solución 2.1 `sapply(iris_miss[1:4], range, na.rm=T)`

2.11 Para terminar vamos a limpiar

Hasta aquí se han creado numerosos objetos de prueba, que se pueden conservar entre sesiones si fuera necesario... pero no en esta ocasión.

Ejemplo 2.17
```
> ls() # lista todos los objetos
> rm(x) # elimina el objeto indicado
```

Ejercicio 2.30 *Elimine* **todos** *los objetos creados durante la sesión.*