Cuaderno de práticas de Aprendizaje Automático.

Correspondiente a las titulaciones de:

Grado en Ingeniería Informática y

Doble Grado de Informática y Matemáticas

Curso 2015-2016.

Profesor de teoría:

Perez de la Blanca N., email: nicolas@decsai.ugr.es

Profesores de prácticas:

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

Matarán P., email: mataran@decsai.ugr.es

Página web de la asignatura: **http : //decsai.ugr.es**

Dpto. Ciencias de la Computación e Inteligencia Artificial

E.T.S. de Ingenierías Informática y de Telecomunicación. Universidad de Granada

1

El cuaderno de prácticas

Los objetivos marcados para el cuaderno de prácticas son:

1. Familiarizar al alumno con la sintaxis del lenguaje R necesaria para desarrollar las prácticas de Aprendizaje Automático.
2. Ver los tipos de objetos y funciones básicas para su manipulación.
3. Generación de gráficos.
4. Lectura, gestión básica y almacenamiento de datos.
5. Definición de funciones para la automatización de las tareas anteriores.

En él se va utilizar R como herramienta para la realización de los ejemplos propuestos, y los ejercicios indicados aunque se puede utilizar el entorno de desarrollo Rstudio (ver apéndice para la instalación de ambos).
Al final de las primeras sesiones se estará en disposición de realizar los ejercicios establecidos en el Trabajo 1 de Programación.

Algunas consideraciones sobre este cuaderno de prácticas:

- El cuaderno de prácticas no pretende ser un manual, sino una forma de visitar los comandos necesarios para adquirir las destrezas requeridas. La documentación de referencia para ampliar conocimiento de cualquier comando son los manuales de referencia de R, que vienen en todas las instalaciones de R. Entre otros se encuentra: An introduction to R. (De lectura recomendada).
- El cuaderno está organizado en varias unidades didácticas. Una unidad didáctica se corresponde aproximadamente con una sesión real de prácticas (2 horas). El propósito de las distintas unidades es visitar la mayoría de los comandos necesarios para cubrir los objetivos que se han expuesto.
- Con objeto de conseguir un mejor aprovechamiento de los comandos que se exploran aquí, conviene seguir, y ejecutar los ejemplos propuestos secuencialmente y explorar diferentes variaciones de los comandos.

- La realización de los ejercicios del cuaderno es voluntaria aunque muy recomendable ya que complementan y dan soporte a los conocimientos que se exponen en las clases teóricas. No se va a exigir la entrega de los ejercicios indicados como

  Ejercicio 1.1 *comandos o conjunto de comandos para realizar una tarea*

  que figuran en el cuaderno, tan solo se entregan los trabajos especificados para tal efecto.

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
```

2.2. Salir de R

Para finalizar cualquier sesión de trabajo, `q()`.
Antes de salir, R nos pregunta si deseamos conservar los objetos creados durante la sesión para la siguiente, en ese caso se guarda la imagen.

2.2.1. Pedir ayuda

Hay disponible mucha ayuda en línea sobre los comandos de R: El primero y más conocido es el comando `help()` para ver la sintaxis del comando en cuestión

```
> 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
```

Guardar una copia de lo realizado

Con `savehistory()` se guarda el histórico de comandos de la sesión activa. La salida es un fichero de texto.

Ejemplo 2.1 `> savehistory() # .Rhistory fichero por defecto`

```
> savehistory("nombrefichero.txt) # parámetro opcional
```

El fichero se guarda en el directorio de trabajo, para saber cuál es `getwd()`, éste se puede cambiar mediante el comando `setwd()`.

2.3. 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 2.1 *Guardar los ejercicios de la sesion de prácticas, en el fichero sesion01.R*

2.4. Las secuencias

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

Ejemplo 2.2

```
> -12:3
> 3:-12
```

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

Ejemplo 2.3

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

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

Ejemplo 2.4 `> seq(-1, 2, by = 0.1)`

Ejercicio 2.2 *crear una secuencia de 1 a 1000 en 4 intervalos*

2.5. Vectores

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

Ejemplo 2.5

```
> x <-c (1.1, 2.2, 3.3, 4.4)   # se crea el vector x y se asigna valores

> x  # se ve su contenido
```

o de forma equivalente:

```
> print(x)

> y <-x+10  # se crea el vector y apartir de x
```

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

Ejemplo 2.6

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

Ejercicio 2.3 *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 2.7

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

Se puede replicar con un factor diferente

Ejemplo 2.8

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

2.5.1. Cómputos sobre vectores

Ejemplo 2.9 `> 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()`

2.5.2. Revisitando las operaciones aritméticas sobre vectores

Ejercicio 2.4 *Estima las longitudes resultantes de las operaciones siguientes:*

Ejemplo 2.10
```
> x + 1 + 2 * y
> x - y
> x + s
> xx + x
```

Ejemplo 2.11
```
> zz <-  xx + x    # A pesar del warning se crea el nuevo vector
```

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 2.5 *Deje ordenado x en orden decreciente.*

Ejercicio 2.6 *Calcule la media del vector x, usando* `sum()` *y* `length()`.

Qué hace la siguiente expresión ?

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

Funciones estadísticas :
`mean, var, sd`

Ejercicio 2.7 *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 2.8 *Simule el lanzamiento de dos dados. Los lanzamientos son independientes.*
Nota: utilice la función `sample`*, necesitará algún parámetro, cuál?*

2.5.3. Acceso a las componentes

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

Ejemplo 2.12
```
> x[1] # accede al primer componente del vector
> x[2:lenght(x)-1]   # todos los componentes excepto el primero y ultimo
```

2.5.4. Imponiendo condiciones entre [ ]

Ejemplo 2.13 `> x[x>4]  # muestra los elementos que hacen verdad condicion`

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

Ejercicio 2.9 *Dado un vector de puntuaciones se han de quitar los valores extremos*

2.5.5. Poniendo nombres a las posiciones

Ejemplo 2.14

```
names(x) <- c("uno","dos","tres","cuatro")
impar <- x[c("uno","tres")]
```

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

Ejemplo 2.15 `> w <- numeric()  # crea el objeto y reserva espacio`

Ejemplo 2.16

```
> length(w)<- 10; w  # se redimensiona y muestra contenido
> w[3:5] <- 5           # o se rellena a medias, ver contenido
```

Ejercicio 2.10 *Dado el vector actual de w, ignore los valores perdidos.*

Ejemplo 2.17 `> w[3:5] <- 5  # se rellena a medias, ver contenido`

2.5.7. Componiendo vectores

Las funciones `cbind(), rbind(), seq()`

A veces interesa mantener la estructura de los datos, pero bajo un mismo identificador. Se pueden combinar los componentes bien por columnas `c`, bien por filas `r`

Ejemplo 2.18

```
sabado.vntas = c(200, 300, 320)    # unidades vendidas el sabado
domingo.vntas = c(120, 230, 200)   # unidades vendidas el domingo
finde.vntas = c(sabado.vntas, domingo.vntas)  # agregación de las ventas
nombre.vntas = c("AB", "CD", "EF")      # nombre articulos
```

Ejemplo 2.19

```
fila.vntas = cbind(sabado.vntas, domingo.vntas)      # se pegan las colmnas
columna.vntas =  rbind(sabado.vntas, domingo.vntas)  # se pegan las filas
colnames(columna.vntas) = nombre.vntas
```

De qué tipo es? `columna.vntas` £Es eso lo que queríamos? Aparcamos las matrices por un momento y vemos otro tipo de objetos.

2.5.8. Data frame

Son objetos para contener datos, como los vectores o arrays pero con columnas de diferentes tipos.

Ejemplo 2.20

```
ventas <- data.frame(nombre.vntas, sabado.vntas, domingo.vntas )
```

La forma de acceder acada uno de los vectores componentes es mediante el nombre del objeto `$` prefijando el nombre del campo.

Ejemplo 2.21

```
 > ventas$sabado.vntas # se muestran las ventas del sabado
 > ventas$sabado.vntas + c(10,5,75)  # se actualiza las ventas del sabado
```

2.5.9. 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 2.22 `> ls() # lista todos los objetos`

```
> rm(x) # elimina el objeto indicado
```

Ejercicio 2.11 *Elimine todos los objetos creados durante la sesión.*

Unidad didáctica 3

Matrices y Gráficos

3.1. Arrays y matrices

Una matriz es un array bidimensional de números, se pueden crear bien `array()` o `matriz()`.

Ejemplo 3.1

```
> a <- array(0, c(2,4))  # se reserva espacio y se asigna tdo a 0
> a <- array(1:8, c(2,4))
> m <- matrix(1:8, nrow = 2, ncol = 4, byrow = TRUE)
> dim(m) # muestra las dimensiones, puede cambiarse
```

3.1.1. Acceso por filas y por columnas

Ejemplo 3.2

```
> m[1,]
> m[,4]
```

Ejercicio 3.1 *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

Ejemplo 3.3 *Dado una matriz cuadrada inicializada al valor de la fila, se quiere poner a 0 los elementos (1,3) (2,2) y (3,1)*

```
> a <- array(1:5, dim=c(5,5))  # Genera array de 5X5 por cols
> i <- array(c(1:3,3:1), dim=c(3,2))  # i bidimensional va a guardar las coordenadas
> a[i]   # se accede a aquellos componentes que seleccionamos
> a[i] <- 0     # se asigna valor
```

Ejercicio 3.2 *Asigne el valor -2 a la diagonal principal de la matriz* `a`.

utilidades sobre matrices

```
ncol(), nrow, t(), rownames(), colnames(), solve()
```

Ejemplo 3.4 *Se le da semántica a filas y columnas*

```
> rownames(a) = LETTERS[1 : 5] # le pone nombre a las filas
> colnames(a) = letters[1 : 5]  # a las columnas
> a
```

Donde `LETTERS` y `letters` son vectores definidos con los caracteres alfabéticos en mayúscula y minúscula, también existen: `month.name` y `month.abb`.

Ejemplo 3.5

```
> y =  t(a) # traspuesta
> z = solve(a) # inversa
```

3.1.2. Operaciones con matrices

```
C = c(rep(0, 3), seq(1, 6, 1), rep(seq(1, 6, 1), 2))
A = matrix(sample(C, 12), nrow = 3, byrow = T); A
B = matrix(sample(C, 12), nrow = 3, byrow = T); B
```

Ejemplo 3.6 *Vamos a realizar operaciones con las*

```
D = A + B; D # Suma de matrices.
E = A - B; E # Resta de A menos B.
F = A %*%t(B); F # Producto de matrices
G = A %o% B ; G #  Producto exterior de matrices.
# de forma equivalente G =  outer(A,B)
```

3.2. Representar gráficos

Los comandos de dibujo se dividen en tres grupos básicos:

- Funciones de alto nivel
- Funciones de bajo nivel que añaden información a gráficos ya existentes

Además, R permite exportar y salvar estos gráficos en múltiples formatos `.pdf, .jpeg`

Cuando creamos un gráfico, asociado a ese gráfico se encuentra el dispositivo gráfico, que puede ser una ventana donde se represente dicho gráfico.

Funciones como: `dev.new(), x11() , window(), dev.off(), ...` para trabajar con estos dispositivos cuando trabajamos con funciones de bajo nivel.

Las funciones de alto nivel nos abren y cierran estos dispositivos **automáticamente**. La más importante: `plot()`.

Vamos a trabajar con nuevos datos

```
> x <- rnorm (100) # datos generados a partir de una normal
> y <- rnorm (100) + 50

> help(plot)
> ?plot # ? comando abreviado

> help(plot)
> ?plot # comando abreviado ?
> plot(x,y) # se abre una ventana
```

Puede observarse en la figura 3.1 una región principal donde se sitúa el plot, con o sin ejes, medido en unidades del propio gráfico. Esta región está delimitado por 4 márgenes.

Ejemplo 3.7

```
> plot(x,y,xlab=" texto eje X",ylab="texto eje Y", main=" Plot de X vs Y")
# se vuelve a abrir la ventana
> text(-1,48, "texto en -2,48") # añade texto
> mtext(paste("lado",1:4), side= 1:4, line= -1, font =2)   # anade texto en los margenes
```

Los comados secuendarios de `text` y `mtext` para escribir texto en la región principal y en los márgenes.

Ejemplo 3.8

```
> plot(x,y,type="n",xlab=" texto eje X",ylab="texto eje Y", main=" Sin Ejes", axes=F)    # s
> points(x,y,col = "green") # coloca los puntos
> axis(1) # coloca eje X
> axis(2, at=seq(48,54,length.out=5)) # coloca eje Y
> box() # cierra la region principal
```

Aunque hay otros que nos pueden ayudar a entender mejor un comando como son: `abline()`,

Ejemplo 3.9

```
> plot(x,y, col= "red", xlab=" texto eje X",ylab="texto eje Y", main=" Plot de X vs Y")    #
> abline(h = 50, v = 0) # rectas horizontal y vertical
> abline(a = 50, b = -3)  # Recta  ax+b
```

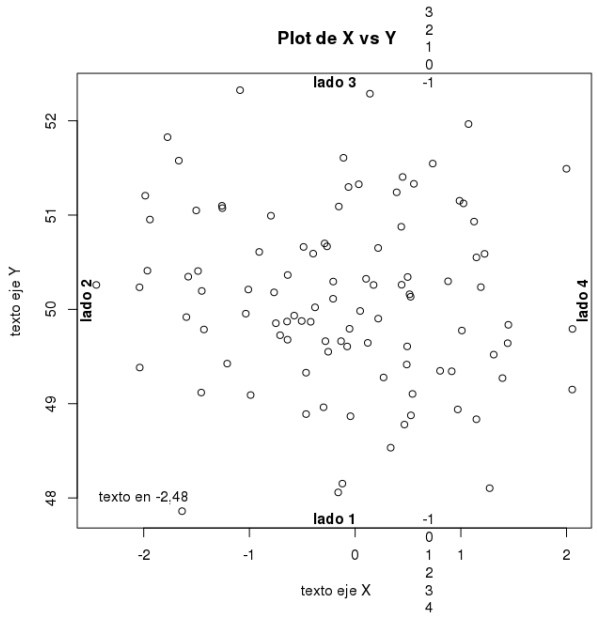

Figura 3.1: Disposición de regiones en un gráfico

3.2.1. Controlando aún más la apariencia

El comando `par()` permite afinar en línea, anchura, tipo, font, color, tamaño. Los márgenes señalados en la figura 3.1 tienen valores por defecto de `5,4,4,2` líneas, éstos valores se pueden cambiar.

Ejemplo 3.10

```
> par(mar=c(4,4,2,2)) # se gana espacio si no hay titulo
> plot(x,y,col=2, xlab=" texto eje X",ylab="texto eje Y", main=" Plot de X vs Y")
```

Ejemplo 3.11

```
> par(mfrow = c(2, 2))
> plot(x, col = "red", type = "l", main = "Figura1")
> plot(rnorm, col = "green", type = "l", main = "Figura2", lwd = 5, lty = 3)
> plot(x, col = "yellow", type = "o", main = "Figura3",
> sub = "Color yellow", lwd = 3)
> plot(x, col = "grey21", type = "S", main = "Figura4")
```

Ejemplo 3.12 *Comandos para la generación de los gráficos de la figura 3.2*

```
> par(mfrow = c(2, 3))
> qqnorm(x, main = "qqnorm")
> hist(x, main = "hist", freq=F)
```

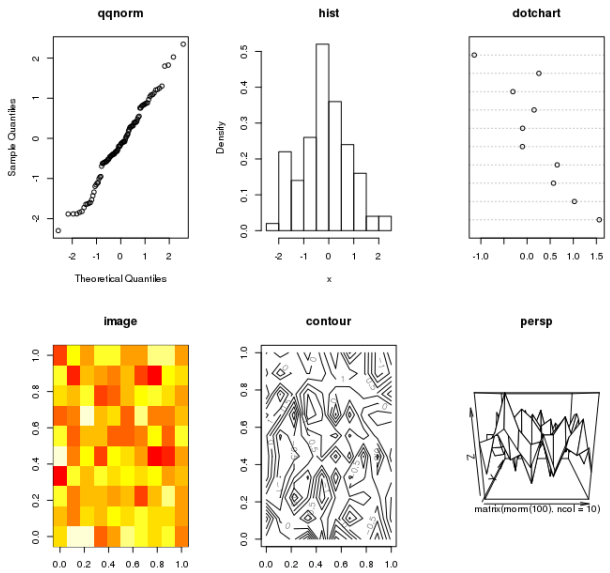

Figura 3.2: Distintos gráficos que se pueden generar.

```
> dotchart(rnorm(10), main = "dotchart")
> image(matrix(rnorm(100), ncol = 10), main = "image")
> contour(matrix(rnorm(100), ncol = 10), main = "contour")
> persp(matrix(rnorm(100), ncol = 10), main = "persp")

> savePlot('plot02.png', type = 'png',  device = dev.cur()) # para salvar en .png
```

Pero hay más tipos de gráficos comunmente utilizados:

- Boxplots
- Barplots
- Pies

Ejercicio 3.3 *Representar en un diagrama de quesos,* `pie` *la misma información que la que se muestra en con el siguiente comando.*

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

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

3.2.2. Para guardar los gráficos

El comando `pdf()` abre un dispositivo gráfico que produce un fichero `.pdf` Conviene cerrar dispositivo gráfico.

Ejemplo 3.13 *Es probable que tengamos varios dispositivos gráficos abiertos... la ventana gráfica y varios* `pdf()`

```
> pdf("plot03.pdf") # cada vez se abre un dispositivo nuevo
> hist(x, freq=F) # histograma
>  curve(dnorm(x), add=T) # se superpone la gráfica
```

Ejemplo 3.14 *se han ejecutado varias veces las instrucciones anteriores...*

```
>  dev.list() # se consulta la lista
>  dev.off(dev.cur()) # cierra el actual o se espefica número
```

Para otros formatos, se puede usar `savePlot` , como el visto en el ejemplo 3.2.1, Con el argumento `type` seleccionamos el formato en el que vamos a guardar la figura, por defecto, se guarda el gráfico del dispositivo actual, dev.cur().

`type` toma valores `"png", "jpeg", "tiff", "bmp"`.

Ejercicio evaluable 4

Trabajo 1. Programación

Fecha límite de entrega: 23 Marzo
Valoración: 8 puntos

4.1. Normas de desarrollo y entrega

4.1.1. Un informe

Para este trabajo asíŋ como, para los sucesivos es obligatorio presentar un **informe escrito** en formato **pdf** con sus *valoraciones* y *decisiones* adoptadas en el desarrollo de cada uno de los apartados además de incluir *los gráficos generados*. Por último, debe incluirse **una valoración** de la calidad de los resultados encontrados.
Nota: **Sin este informe se considera que el trabajo es NO PRESENTADO.**

4.1.2. Normas para el desarrollo de los trabajos

EL INCUMPLIMIENTO DE ESTAS NORMAS SIGNIFICA PERDIDA DE 2 PUNTOS POR CADA INCUMPLIMIENTO.

- El código se debe estructurar en un único script R con distintas funciones o apartados, uno por cada ejercicio/apartado de la práctica.
- Todos los resultados numéricos o gráficas serán mostrados por pantalla, parando la ejecución después de cada apartado. No escribir nada en el disco.
- El path que se use en la lectura de cualquier fichero auxiliar de datos debe ser siempre "datos/nombre_fichero". Es decir, crear un directorio llamado "datosâĂİ dentro del directorio donde se desarrolla y se ejecuta la práctica.
- Un código es apto para ser corregido si se puede ejecutar de principio a fin sin errores.
- NO ES VALIDO usar opciones en las entradas. Para ello fijar al comienzo los parámetros por defecto que considere que son los óptimos.

- El código debe estar obligatoriamente comentado explicando lo que realizan los distintos apartados y/o bloques.
- Poner puntos de parada para mostrar imágenes o datos por consola.
- Todos los ficheros se entregan juntos dentro de un único fichero zip.
- ENTREGAR SOLO EL CODIGO FUENTE, NUNCA LOS DATOS.
- **Forma de entrega**: Subir el zip al Tablón docente de CCIA, hay una entrega habilitada a tal efecto.

4.2. Ejercicio de Generación y Visualización de datos (2 puntos)

1. Construir una función $lista = simula\_unif(N, dim, rango)$ que calcule una lista de longitud N de vectores de dimensión dim conteniendo números aleatorios uniformes en el intervalo $rango$.

2. Construir una función $lista = simula\_gaus(N, dim, sigma)$ que calcule una lista de longitud N de vectores de dimensión dim conteniendo números aleatorios gaussianos de media 0 y varianzas dadas por el vector $sigma$.

3. Suponer $N = 50$, $dim = 2$, $rango = [-50, +50]$ en cada dimensión. Dibujar una gráfica de la salida de la función correspondiente.

4. Suponer $N = 50$, $dim = 2$ y $sigma = [5, 7]$ dibujar una gráfica de la salida de la función correspondiente.

5. Construir la función *v=simula_ recta(intervalo)* que calcula los parámetros, $v = (a, b)$ de una recta aleatoria, $y = ax + b$, que corte al cuadrado $[-50, 50] \times [-50, 50]$ (Ayuda: Para calcular la recta simular las coordenadas de dos puntos dentro del cuadrado y calcular la recta que pasa por ellos)

6. Generar una muestra 2D de puntos usando *simula_ unif()* y etiquetar la muestra usando el signo de la función $f(x, y) = y - ax - b$ de cada punto a una recta simulada con *simula_ recta()*. Mostrar una gráfica con el resultado de la muestra etiquetada junto con la recta usada para ello.

7. Usar la muestra generada en el apartado anterior y etiquetarla con +1,-1 usando el signo de cada una de las siguientes funciones

   - $f(x, y) = (x - 10)^2 + (y - 20)^2 - 400$
   - $f(x, y) = 0{,}5(x + 10)^2 + (y - 20)^2 - 400$
   - $f(x, y) = 0{,}5(x - 10)^2 - (y + 20)^2 - 400$
   - $f(x, y) = y - 20x^2 - 5x + 3$

   Visualizar el resultado del etiquetado de cada función junto con su gráfica y comparar el resultado con el caso lineal Â£Que consecuencias extrae sobre la forma de las regiones positiva y negativa?

8. Considerar de nuevo la muestra etiquetada en el apartado.6. Modifique las etiquetas de un 10 % aleatorio de muestras positivas y otro 10 % aleatorio de negativas.

   - Visualice los puntos con las nuevas etiquetas y la recta del apartado.6.
   - En una gráfica aparte visualice nuevo los mismos puntos pero junto con las funciones del apartado.7.

Observe las gráficas y diga que consecuencias extrae del proceso de modificación de etiquetas en el proceso de aprendizaje.

4.3. Ejercicio de Ajuste del Algoritmo Perceptron (3 puntos)

1. Implementar la función $sol = ajusta\_PLA(datos, label, max\_iter, vini)$ que calcula el hiperplano solución a un problema de clasificación binaria usando el algoritmo PLA. La entrada *datos* es una matriz donde cada item con su etiqueta está representado por una fila de la matriz, *label* el vector de etiquetas (cada etiqueta es un valor +1 o-1), max_iter es el número máximo de iteraciones permitidas y *vini* el valor inicial del vector. La salida *sol* devuelve los coeficientes del hiperplano.

2. Ejecutar el algoritmo PLA con los valores simulados en apartado.6 del ejercicio.4.2, inicializando el algoritmo con el vector cero y con vectores de números aleatorios en $[0, 1]$ (10 veces). Anotar el número medio de iteraciones necesarias en ambos para converger. Valorar el resultado

3. Ejecutar el algoritmo PLA con los datos generados en el apartado.8 del ejercicio.4.2, usando valores de 10, 100 y 1000 para max_iter. Etiquetar los datos de la muestra usando la función solución encontrada y contar el número de errores respecto de las etiquetas originales. Valorar el resultado.

4. Repetir el análisis del punto anterior usando la primera función del apartado.7 del ejercicio.4.2

5. Modifique la función $ajusta\_PLA$ para que le permita visualizar los datos y soluciones que va encontrando a lo largo de las iteraciones. Ejecute con la nueva versión el apartado.3 del ejercicio.4.2

6. A la vista de la conducta de las soluciones observada en el apartado anterior, proponga e implemente una modificación de la función original $sol = ajusta\_PLA\_MOD(...)$ que permita obtener soluciones razonables sobre datos no linealmente separables. Mostrar y valorar el resultado encontrado usando los datos del apartado.7 del ejercicio.4.2

4.4. Ejercicio sobre Regresión Lineal (3 puntos)

1. Abra el fichero ZipDigits.info disponible en la web del curso y lea la descripción de la representación numérica de la base de datos de números manuscritos que hay en el fichero ZipDigits.train.

2. Lea el fichero ZipDigits.train dentro de su código y visualize las imágenes. Seleccione solo las instancias de los números 1 y 5. Guardelas como matrices de tamaíśo 16x16.

3. Para cada matriz de números calcularemos su valor medio y su grado de simetríŋa vertical. Para calcular la simetríŋa calculamos la suma del valor absoluto de las diferencias en cada píŋxel entre la imagen original y la imagen que obtenemos invertiendo el orden de las columnas. Finalmente le cambiamos el signo.

4. Representar en los ejes { X=Intensidad Promedio, Y=Simetríŋa} las instancias seleccionadas de 1's y 5's.

5. Implementar la función $sol = Regress\_Lin(datos, label)$ que permita ajustar un modelo de regresion lineal (usar SVD). Los datos de entrada se interpretan igual que en clasificación.

6. Ajustar un modelo de regresion lineal a los datos de (Intensidad promedio, Simetria) y pintar la solución junto con los datos. Valorar el resultado.

7. En este ejercicio exploramos cómo funciona regresión lineal en problemas de clasificación. Para ello generamos datos usando el mismo procedimiento que en ejercicios anteriores. Suponemos $\mathcal{X} = [-10, 10] \times [-10, 10]$ y elegimos muestras aleatorias uniformes dentro de $\mathcal{X}$. La función f en cada caso será una recta aleatoria que corta a $\mathcal{X}$ y que asigna etiqueta a cada punto con el valor de su signo. En cada apartado generamos una muestra y le asignamos etiqueta con la función f generada. En cada ejecución generamos una nueva función f

   a) Fijar el tamaíśo de muestra N = 100. Usar regresión lineal para encontrar g y evaluar E_{in}, (el porcentaje de puntos incorrectamente clasificados). Repetir el experimento 1000 veces y promediar los resultados Â£Qué valor obtiene para E_{in}?

   b) Fijar el tamaíśo de muestra N = 100. Usar regresión lineal para encontrar g y evaluar E_{out}. Para ello generar 1000 puntos nuevos y usarlos para estimar el error fuera de la muestra, E_{out} (porcentaje de puntos mal clasificados). De nuevo, ejecutar el experimento 1000 veces y tomar el promedio. Â£Qué valor obtiene de E_{out}? Valore los resultados

   c) Ahora fijamos N = 10, ajustamos regresión lineal y usamos el vector de pesos encontrado como un vector inicial de pesos para PLA. Ejecutar PLA hasta que converja a un vector de pesos final que separe completamente la muestra de entrenamiento. Anote el número de iteraciones y repita el experimento 1.000 veces Â£Cual es valor promedio de iteraciones que tarda PLA en converger? (En cada iteración de PLA elija un punto aleatorio del conjunto de mal clasificados). Valore los resultados

8. En este ejercicio exploramos el uso de transformaciones no lineales. Consideremos la función objetivo $f(x_1, x_2) = \text{sign}(x_1^2 + x_2^2 - 0{,}6)$. Generar una muestra de entrenamiento de $N = 1000$ puntos a partir de $\mathcal{X} = [-10, 10] \times [-10, 10]$ muestreando cada punto $x \in \mathcal{X}$ uniformemente. Generar las salidas usando el signo de la función en los puntos muestreados. Generar ruido sobre las etiquetas cambiando el signo de las salidas a un 10 % de puntos del conjunto aleatorio generado.

- Ajustar regresion lineal, para estimar los pesos w. Ejecutar el experimento 1.000 y calcular el valor promedio del error de entrenamiento E_{in}. Valorar el resultado.
- Ahora,consideremos $N = 1000$ datos de entrenamiento y el siguiente vector de variables: $(1, x_1, x_2, x_1x_2, x_1^2, x_2^2)$. Ajustar de nuevo regresión lineal y calcular el nuevo vector de pesos $\hat{w}$. Mostrar el resultado.
- Repetir el experimento anterior 1.000 veces calculando en cada ocasión el error fuera de la muestra. Para ello generar en cada ejecución 1.000 puntos nuevos y valorar sobre ellos la función ajustada. Promediar los valores obtenidos Â£Qué valor obtiene?. Valorar el resultado.

Apéndice A

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.

A.1. INSTALACIÓN DE R

A.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-3.2.3 for Windows" para descargar el fichero "R-3.2.3-win.exe" en nuestro ordenador.

A.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 A.1, escribimos la ruta `X:\R-3.2.3`, 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

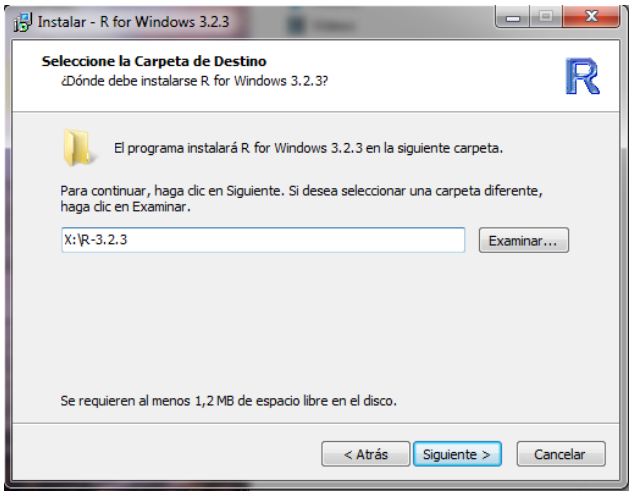

Figura A.1: Captura de pantalla de selección de "Carpeta de Destino".

- 32 o 64 bits dependiendo de nuestro sistema operativo

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 A.2

A.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.

A.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-3.2.3\bin\x64\Rgui.exe` para la versión de 64 bits, o

`X:\R-3.2.3\bin\i386\Rgui.exe` para la versión de 32 bits.

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

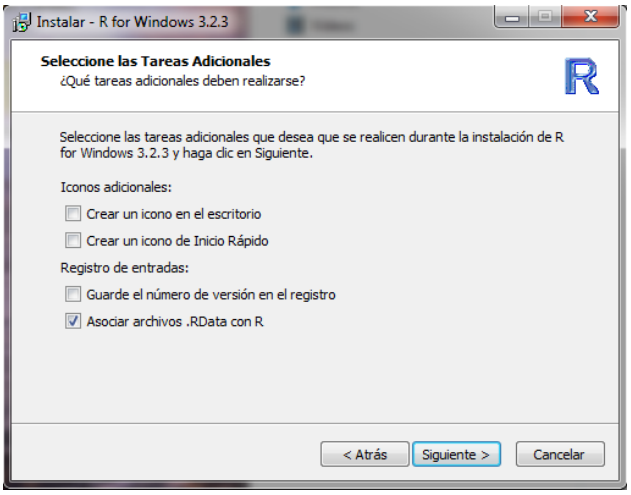

Figura A.2: Captura de pantalla de selección de "TareasAdicionales".

A.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.

A.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.

A.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.

A.2.1. Descargando datos

Desde el navegador accedemos a la siguiente url :`https://www.rstudio.com/products/rstudio/download`. Como vemos en la figura A.3, aquí podemos elegir entre descargar un instalador para Windows, o el formato .zip. En nuestro caso elegiremos éste último.

A.2.2. Instalación de RStudio

Una vez descargado el fichero "RStudio-0.99.878.zip", creamos el directorio `''X:\RStudio''`, y descomprimimos el contenido del fichero descargado en el directorio.

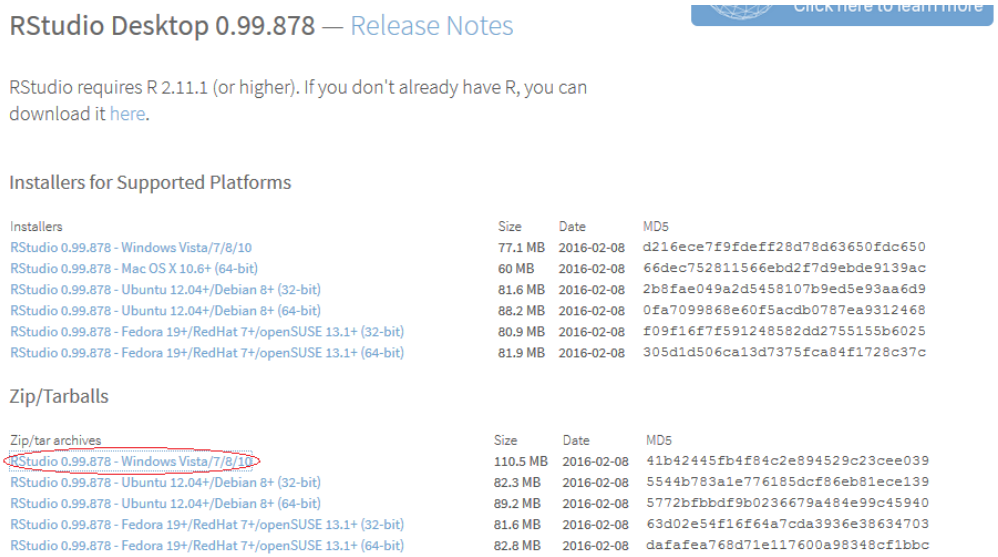

Figura A.3: Captura de pantalla de selección de "RStudio".

A.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:

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

A.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.

A.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.