

Trabajo 1

Alejandro García Montoro

20 de marzo de 2016

Generalización y visualización de datos

Ejercicio 1

Para la realización de este ejercicio se ha implementado la siguiente función, que genera una lista de N vectores, cada de los cuales contiene una muestra de tamaño `dim` de una distribución uniforme entre los valores especificados por el vector de dos elementos `rango`.

```
# Devuelve una lista de N vectores de dimensión dim con una muestra  
# uniforme en el intervalo [rango[1], rango[2]]  
simula_unif <- function(N, dim, rango){  
  lapply(rep(dim, N), runif, min = rango[1], max = rango[2])  
}
```

La implementación es sencilla: como `runif` —el generador aleatorio de muestras de una distribución uniforme— recibe el número de muestras que se desea generar, basta aplicar esta función al número `dim` tantas veces como se indique en N . Esto se consigue con la función `lapply`, que devuelve una lista del mismo tamaño que la lista que se le pasa como primer argumento, ejecutando entonces la función pasada como segundo argumento sobre cada uno de los valores de la lista.

Ejercicio 2

De forma análogo al ejercicio anterior, la siguiente función genera una lista de N vectores, cada de los cuales contiene una muestra de tamaño `dim` de una distribución gaussiana de media 0 y desviación típica correspondiente al elemento i -ésimo especificado en el vector `sigma`.

```
# Devuelve una lista de N vectores de dimensión dim con una muestra  
# gaussiana de media cero y desviación sigma  
simula_gauss <- function(N, dim, sigma){  
  lapply(rep(dim, N), rnorm, mean = 0, sd = sigma)  
}
```

La implementación es análoga a la anterior, cambiando la función `runif` por `rnorm`. En este caso, además, es necesario notar que el elemento i -ésimo del vector `sigma`, que tiene dimensión igual a `dim` —si no tiene la misma dimensión, se ciclará sobre él como es habitual en R—, indica la desviación que tendrá la muestra i -ésima de cada uno de los vectores devueltos.

Ejercicio 3

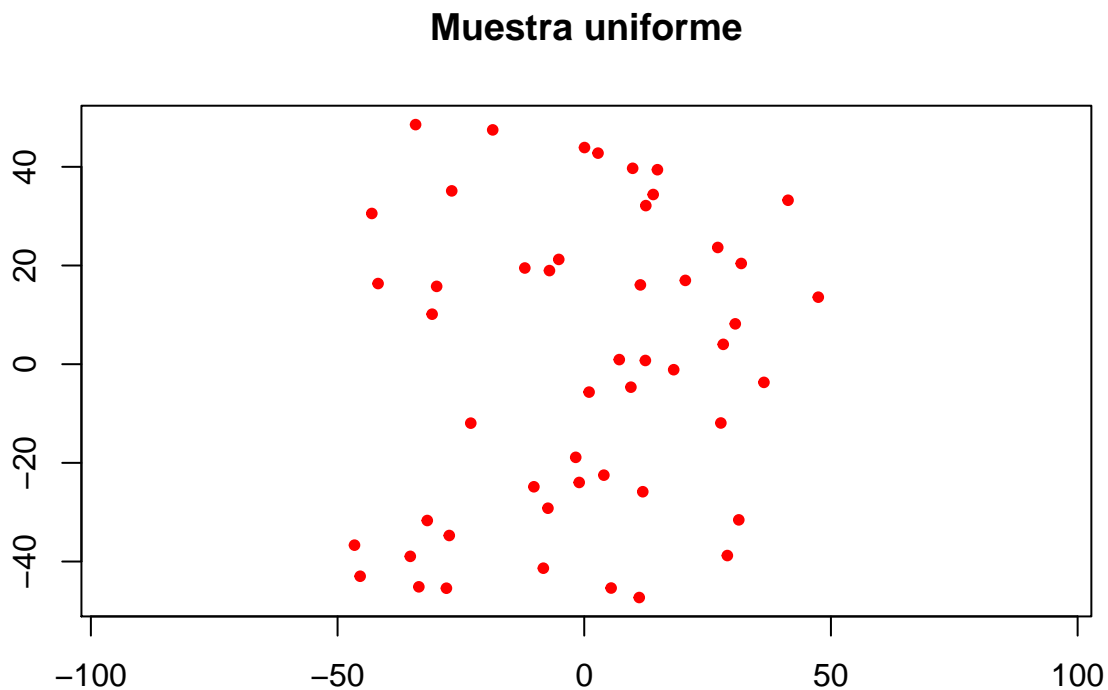
Para la resolución de este ejercicio generamos primero los datos solicitados, usando la función anterior y tomando como x (resp. y) los primeros (resp. segundos) valores de cada vector generado:

```
# Muestra uniforme de 50 puntos en el cuadrado [-50,50] x [-50,50]
datos_unif <- simula_unif(50, 2, c(-50,50))

# Guardamos las coordenadas en variables diferentes
unif.x <- unlist(lapply(datos_unif, '[[', 1))
unif.y <- unlist(lapply(datos_unif, '[[', 2))
```

Para el dibujo de la gráfica basta usar la función plot:

```
plot(unif.x, unif.y, col = 'red', pch = 20, asp=1,
     main="Muestra uniforme", xlab="", ylab="")
```



Ejercicio 4

Igual que antes, generamos primero los datos y guardamos en variables diferentes las ordenadas y las abscisas:

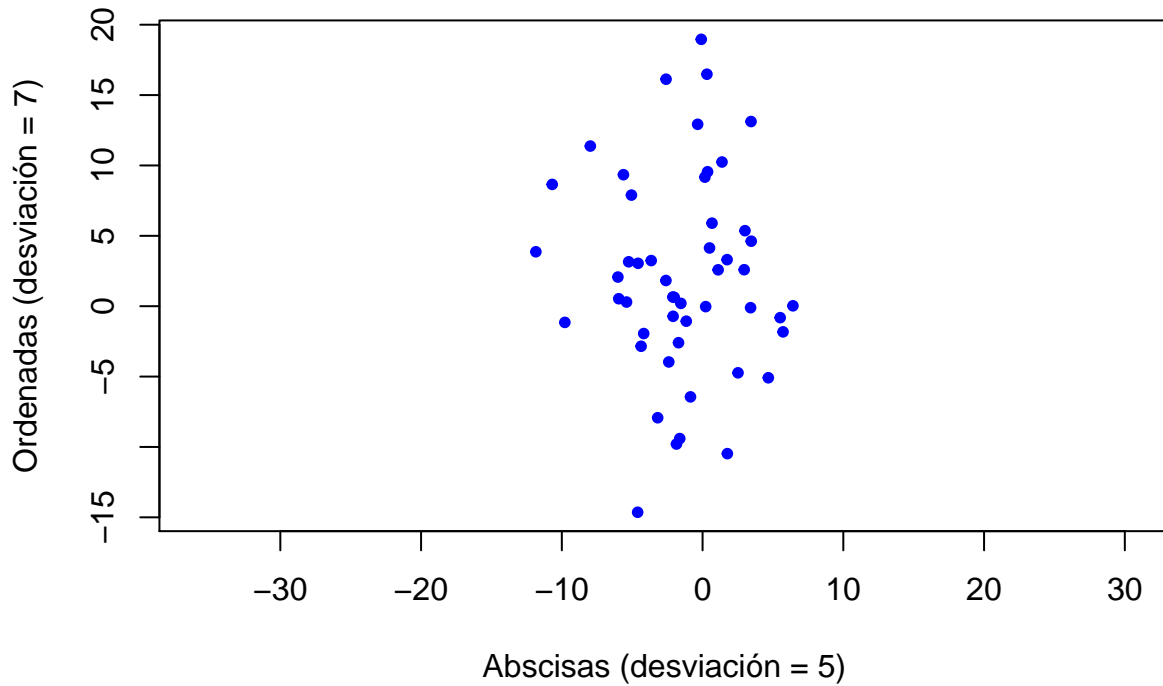
```
# Muestra gaussiana de 50 puntos con media 0, desviación 5 en la
# abscisa y desviación 7 en la ordenada

# Guardamos las coordenadas en variables diferentes
datos_norm <- simula_gauss(50, 2, c(5,7))
norm.x <- unlist(lapply(datos_norm, '[[', 1))
norm.y <- unlist(lapply(datos_norm, '[[', 2))
```

Para la generación de la gráfica necesitamos conocer el intervalo en el que se mueven los datos. La función range nos devuelve justo eso, y con su resultado podemos definir los intervalos de cada eje:

```
plot(norm.x, norm.y, col = 'blue', pch = 20, asp = 1,
     main="Muestra gaussiana", xlab="Abscisas (desviación = 5)", ylab="Ordenadas (desviación = 7)")
```

Muestra gaussiana



Ejercicio 5

Para la generación de una recta aleatoria se ha implementado la siguiente función:

```
# Genera los parámetros (a,b) de una recta aleatoria y = a*x + b que corta  
# al cuadrado [intervalo[1], intervalo[2]] x [intervalo[1], intervalo[2]]  
simula_recta <- function(intervalo){  
  # Simulamos dos puntos dentro del cuadrado intervalo x intervalo  
  punto1 <- runif(2, min=intervalo[1], max=intervalo[2])  
  punto2 <- runif(2, min=intervalo[1], max=intervalo[2])  
  
  # Generamos los parámetros que definen la recta  
  a <- (punto2[2] - punto1[2]) / (punto2[1] - punto1[1])  
  b <- -a * punto1[1] + punto1[2]  
  
  # Devolvemos un vector concatenando ambos parámetros  
  c(a,b)  
}
```

Ejercicio 6

Para este ejercicio y el resto de la práctica vamos a crear una estructura abstracta que genera funciones etiquetadoras. Para ello, definimos `generador_etiquetados`, una función que al ser llamada con otra función `f` como parámetro devuelve una función etiquetadora; es decir, una función que devuelve 1 o -1 según el signo que toma la función `f` al recibir los parámetros `x` e `y`:

```

# Devuelve una función etiquetadora basada en el signo que
# toma el parámetro f
generador_etiquetados <- function(f){
  function(x,y){
    sign(f(x,y))
  }
}

```

Resolver el ejercicio con la estructura anterior es ahora más sencillo: simplemente tenemos que definir la función $f(x,y) = y - ax - b$, donde a y b son los parámetros que definen la recta $y = ax + b$.

Por tanto, basta simular una recta haciendo uso de la función implementada anteriormente y definir la función f_1 , que será la que pasaremos al generador de funciones etiquetadoras:

```

# Generamos recta aleatoria
recta <- simula_recta(c(-50,50))

# Definimos la función cuyo signo etiquetará los datos
f1 <- function(x,y){
  y - recta[1]*x - recta[2]
}

# Generamos función etiquetadora
etiquetado1 <- generador_etiquetados(f1)

```

Por último, generamos la gráfica solicitada simulando datos de una distribución uniforme: extraemos los datos de las ordenadas, los de las abscisas y generamos los etiquetados con la función `etiquetado1`:

```

# Generamos muestra uniforme aleatoria
datos_unif <- simula_unif(50, 2, c(-50,50))

# Guardamos las coordenadas en variables separadas
unif.x <- unlist(lapply(datos_unif, '[', 1))
unif.y <- unlist(lapply(datos_unif, '[', 2))

# Encapsulamos los datos y las etiquetas en un data frame
datos <- data.frame(X = unif.x, Y = unif.y, Etiqueta = etiquetado1(unif.x, unif.y))

```

Por último, generamos la gráfica con `plot`, asignando un color diferente según el etiquetado, y añadiendo la gráfica de la recta simulada:

```

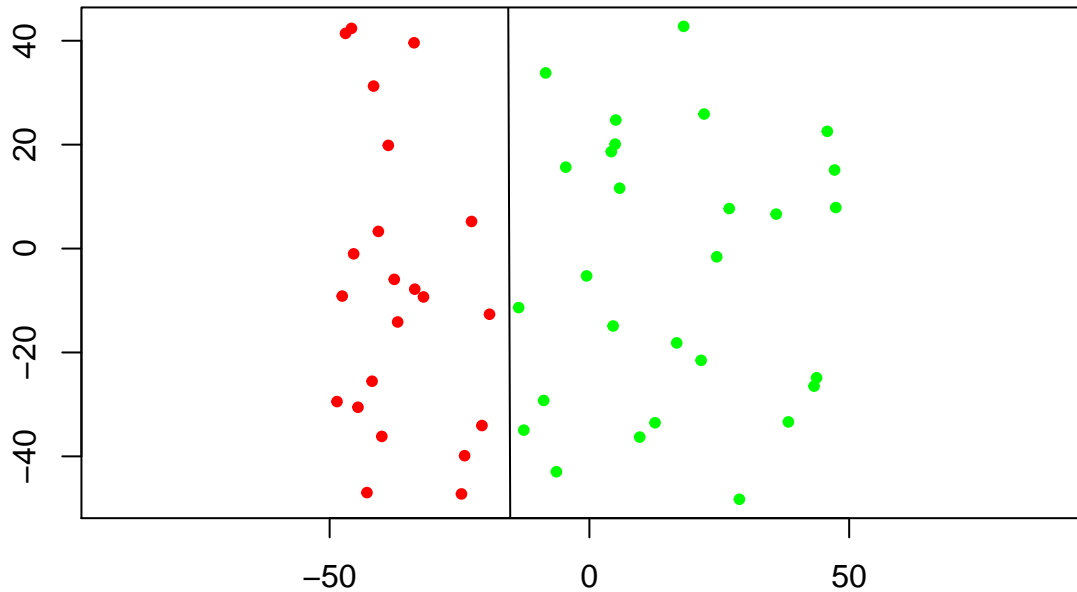
# Definimos un vector de colores basado en las etiqueta
colores <- ifelse(datos$Etiqueta == 1, "green", "red")

# Generamos la gráfica
plot(datos$X, datos$Y, asp = 1, col = colores, pch = 20,
      main="Muestra uniforme etiquetada", xlab="", ylab="")

# Dibujamos la recta clasificadora
abline(rev(recta))

```

Muestra uniforme etiquetada



Ejercicio 7

Para realizar este ejercicio definimos las funciones f_2, f_3, f_4 y f_7 como sigue:

```
# Generación de funciones para etiquetar
f2 <- function(x,y){
  (x-10)**2 + (y-20)**2 - 400
}

f3 <- function(x,y){
  0.5*(x+10)**2 + (y-20)**2 - 400
}

f4 <- function(x,y){
  0.5*(x-10)**2 - (y+20)**2 - 400
}

f5 <- function(x,y){
  y - 20*x**2 - 5*x + 3
}
```

Vamos a definir ahora una función que encapsula el trabajo de etiquetar la muestra, dibujarla y añadir la gráfica de las funciones f_i . Dejamos además que se le pasen argumentos arbitrarios con los que llamar a la función plot, para poder manejar desde fuera las opciones del dibujado:

```
# Etiqueta la muestra (dat.x, dat.y) con la función f y genera el
# correspondiente gráfico
genera_grafico <- function(dat.x, dat.y, f, ...){
  # Etiqueta la muestra
  etiquetado <- generador_etiquetados(f)
```

```

etiquetas <- etiquetado(dat.x, dat.y)

# Genera vector de colores basado en las etiquetas
colores <- ifelse(etiquetas == 1, "green", "red")

# Dibujo de la muestra (pasamos los argumentos ... a plot)
plot(dat.x, dat.y, asp = 1, col = colores, pch = 20, ...)

# Dibujo de la gráfica de la función
f.x <- seq(-50, 50, length=1000)
f.y <- seq(-50, 50, length=1000)
f.z <- outer(f.x, f.y, f)
contour(f.x, f.y, f.z, levels=0, col = "blue", add=T, drawlabels=F)

# Devolvemos las etiquetas, que las necesitaremos más adelante:
return(etiquetas)
}

```

Generamos ahora un gráfico multiple con 4 plots. Para esto, definimos la estructura con `par()` y llamamos a la función anterior con los parámetros de cada plot:

```

# Definimos una rejilla 2x2 para los plots
old_mfrow <- par()$mfrow
old_mar <- par()$mar
old_oma <- par()$oma

par(mfrow=c(2, 2), mar=c(0.1, 1.1, 2.1, 0.1), oma=2.5*c(1,1,1,1))

# Generamos los cuatro plots
etiquetas_2 <- genera_grafico(unif.x, unif.y, f2, cex=0.5, cex.main=0.9, xlab="", ylab="", xaxt="n",
                             main=expression(f[2](x,y) == (x-10)^2 + (y-20)^2 - 400))

etiquetas_3 <- genera_grafico(unif.x, unif.y, f3, cex=0.5, cex.main=0.9, xlab="", ylab="", xaxt="n", ya
                             main=expression(f[3](x,y) == 0.5(x+10)^2 + (y-20)^2 - 400))

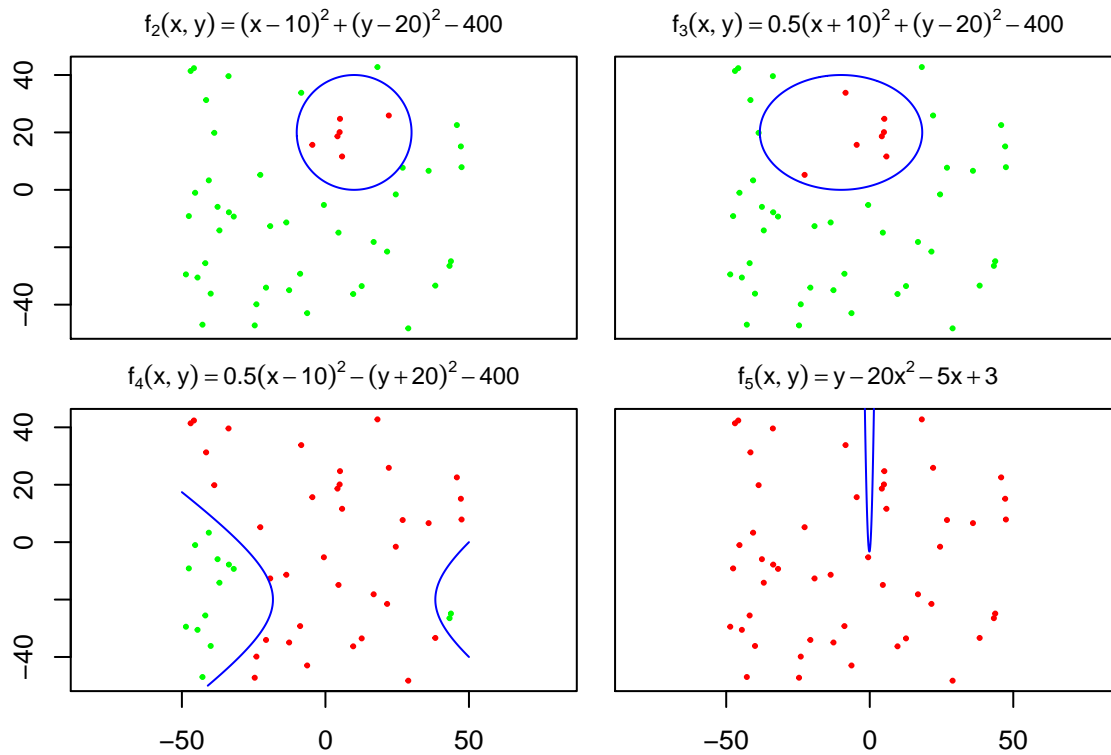
etiquetas_4 <- genera_grafico(unif.x, unif.y, f4, cex=0.5, cex.main=0.9, xlab="", ylab="",
                             main=expression(f[4](x,y) == 0.5(x-10)^2 - (y+20)^2 - 400))

etiquetas_5 <- genera_grafico(unif.x, unif.y, f5, cex=0.5, cex.main=0.9, xlab="", ylab="", yaxt="n",
                             main=expression(f[5](x,y) == y - 20*x^2 - 5*x + 3))

mtext("Etiquetado de una muestra con diversas funciones", outer=TRUE, line=0.5)

```

Etiquetado de una muestra con diversas funciones



```
# Dejamos los parámetros como estaban anteriormente
par(mfrow=old_mfrow, mar=old_mar, oma=old_oma)
```

La forma de las regiones positiva y negativa depende, obviamente, de la función con la que se genera el etiquetado. f_2 divide la muestra con una circunferencia, f_3 con una elipse, f_4 con una hipérbola y f_5 con una parábola. Podemos notar, además, como la muestra considerada no da ninguna información con f_4 , ya que la región verde se queda vacía; todas las muestras se etiquetan como rojas. En las demás pasa algo parecido, ya que una de las regiones es mucho más pequeña que la otra, con lo que la cantidad de muestras en la región menor dará poca información para el proceso de aprendizaje.

Ejercicio 8

Para realizar este ejercicio retomamos la variable `datos` definida en el ejercicio 6 y generamos ruido, tomando un 10% de las muestras positivas, otro 10% de las negativas y cambiándole el signo a ambas.

```
# Tomamos los índices de las etiquetas positivas y los de las etiquetas negativas.
indices_pos <- which(datos$Etiqueta %in% 1)
indices_neg <- which(datos$Etiqueta %in% -1)

# Tomamos una muestra del 10% de esos índices
cambiar_pos <- sample(indices_pos, round(0.1*length(indices_pos)))
cambiar_neg <- sample(indices_neg, round(0.1*length(indices_neg)))

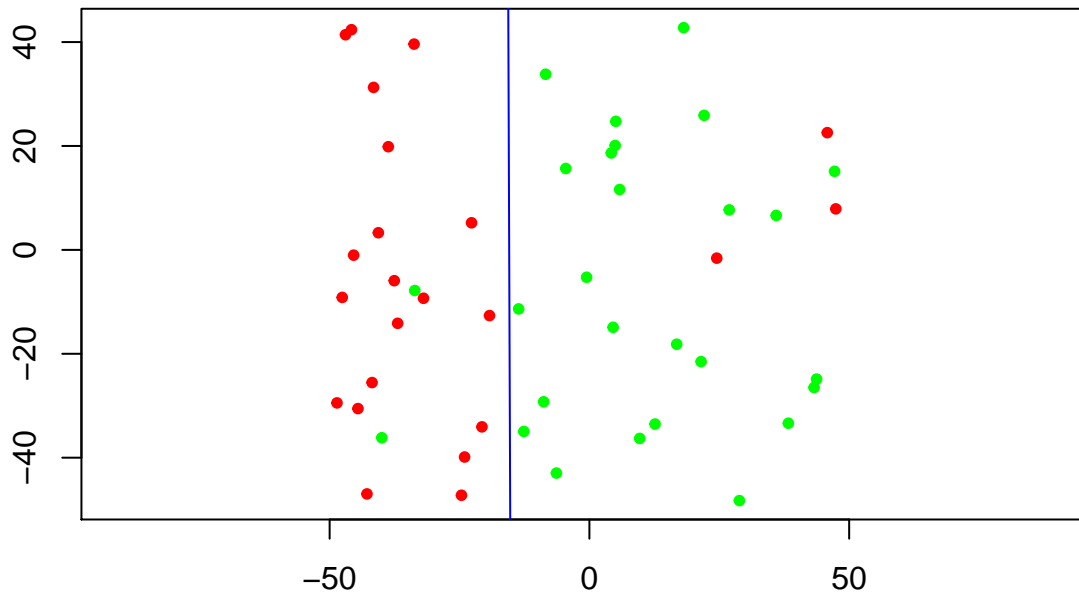
# Los índices positivos los ponemos a -1 y los negativos a 1
datos$Etiqueta[cambiar_pos] <- -1
datos$Etiqueta[cambiar_neg] <- 1
```

Generamos ahora el gráfico con las etiquetas cambiadas. Retomamos además la variable `recta` definida en el ejercicio 6 y la dibujamos encima.

```
# Generamos vector de colores con las nuevas etiquetas
colores <- ifelse(datos$Etiqueta == 1, "green", "red")

# Dibujamos la muestra con los colores recién calculados y la recta del apartado 6
plot(datos$X, datos$Y, asp = 1, col = colores, pch = 20,
     main="Muestra uniforme con ruido", xlab="", ylab="")
abline(rev(recta), col="blue")
```

Muestra uniforme con ruido



Generalizamos ahora la función `genera_grafico()` definida en el anterior apartado. Añadimos dos parámetros:
 * `etiquetas`: Vector de etiquetas previamente calculado. Si no se pasa este parámetro, se generan con la f. *
 * `generarRuido`: Variable booleana que indica si se genera un 10% de ruido en las muestras tanto positivas como negativas.

El parámetro `generarRuido` lo necesitamos ahora para cambiar el 10% de las etiquetas en cada ejecución. El parámetro `etiquetas`, aunque ahora no lo usaremos, será necesario más adelante:

```
# Etiqueta la muestra (dat.x, dat.y) con la función f y genera el
# correspondiente gráfico
genera_grafico <- function(dat.x, dat.y, f, etiquetas, generarRuido = F, ...){
  # Si el parámetro etiquetas no se ha pasado, se generan con la f
  if(missing(etiquetas)){
    etiquetado <- generador_etiquetados(f)
    etiquetas <- etiquetado(dat.x, dat.y)
  }

  # Se genera ruido en el 10% de etiquetas positivas y negativas
  if(generarRuido){
    # Tomamos los índices de las etiquetas positivas y los de las etiquetas negativas.
    indices_pos <- which(etiquetas %in% 1)
```



```

indices_neg <- which(etiquetas %in% -1)

# Tomamos una muestra del 10% de esos índices
cambiar_pos <- sample(indices_pos, round(0.1*length(indices_pos)))
cambiar_neg <- sample(indices_neg, round(0.1*length(indices_neg)))

# Los índices positivos los ponemos a -1 y los negativos a 1
etiquetas[cambiar_pos] <- -1
etiquetas[cambiar_neg] <- 1
}

# Genera vector de colores basado en las etiquetas
colores <- ifelse(etiquetas == 1, "green", "red")

# Dibujo de la muestra (pasamos los argumentos ... a plot)
plot(dat.x, dat.y, asp = 1, col = colores, pch = 20, ...)

# Dibujo de la gráfica de la función
f.x <- seq(-50, 50, length=1000)
f.y <- seq(-50, 50, length=1000)
f.z <- outer(f.x, f.y, f)
contour(f.x, f.y, f.z, levels=0, col = "blue", add=T, drawlabels=F)

# Devolvemos las etiquetas, que las necesitaremos más adelante:
return(etiquetas)
}

```

Generamos ahora los cuatro gráficos pedidos con las funciones f_i anteriores. Para ello, le pasamos la muestra a la función `genera_grafico`, calculamos las etiquetas con cada función f_i y generamos un 10% de ruido en cada subconjunto de etiquetas:

```

par(mfrow=c(2, 2), mar=c(0.1, 1.1, 2.1, 0.1), oma=2.5*c(1,1,1,1))

# Generamos los cuatro plots
etiquetas_ruido_2 <- genera_grafico(datos$X, datos$Y, f2, generarRuido = T, cex=0.5, cex.main=0.9,
                                   xlab="", ylab="", xaxt="n",
                                   main=expression(f[2](x,y) == (x-10)^2 + (y-20)^2 - 400))

etiquetas_ruido_3 <- genera_grafico(datos$X, datos$Y, f3, generarRuido = T, cex=0.5, cex.main=0.9,
                                   xlab="", ylab="", xaxt="n", yaxt="n",
                                   main=expression(f[3](x,y) == 0.5(x+10)^2 + (y-20)^2 - 400))

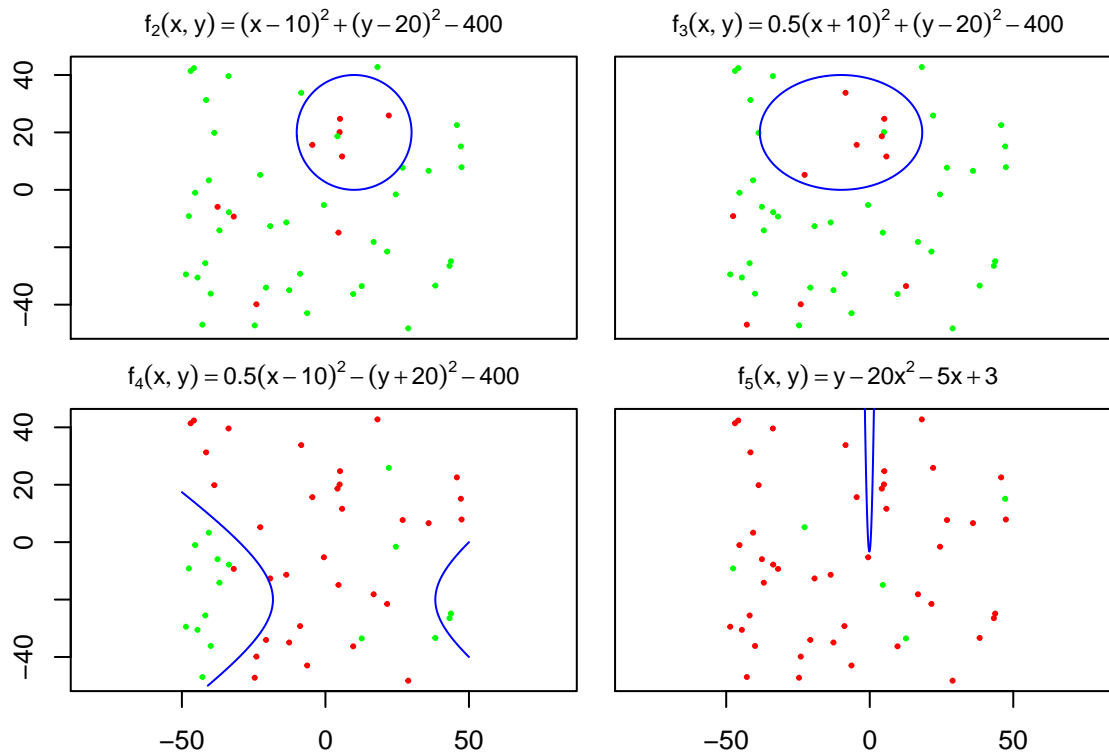
etiquetas_ruido_4 <- genera_grafico(datos$X, datos$Y, f4, generarRuido = T, cex=0.5, cex.main=0.9,
                                   xlab="", ylab="",
                                   main=expression(f[4](x,y) == 0.5(x-10)^2 - (y+20)^2 - 400))

etiquetas_ruido_5 <- genera_grafico(datos$X, datos$Y, f5, generarRuido = T, cex=0.5, cex.main=0.9,
                                   xlab="", ylab="", yaxt="n",
                                   main=expression(f[5](x,y) == y - 20*x^2 - 5*x + 3))

mtext("Etiquetado ruidoso de una muestra con diversas funciones", outer=TRUE, line=0.5)

```

Etiquetado ruidoso de una muestra con diversas funciones



```
# Dejamos los parámetros como estaban anteriormente
par(mfrow=old_mfrow, mar=old_mar, oma=old_oma)
```

El comentario final del ejercicio 7 se refuerza ahora con estos resultados: al tener muestras con ruido, el etiquetado de estas funciones, con este número tan bajo de muestras, no aporta prácticamente información ninguna. Por ejemplo, en los dos primeros casos, el número de etiquetas dentro de la elipse es prácticamente igual al número de muestras mal etiquetadas fuera de ella. Así, el proceso de aprendizaje será complicado. Por otro lado, en el último caso, la única información —errónea— que tendremos será la del ruido, ya que el etiquetado de la función que define una región tan pequeña es inútil para el aprendizaje.

Ajuste del algoritmo Perceptron

Ejercicio 1

La función implementada para el algoritmo Perceptron es la siguiente:

```
# Implementa el algoritmo Perceptron. Devuelve una lista con dos valores:
# Sol: Recta solución
# Iter: Número de iteraciones que han sido necesarias
ajusta_PLA <- function(datos, label, max_iter, vini){

  # Definimos w como el vector inicial. Si tiene una posición más que
  # el número de datos, lo dejamos como está; si no, añadimos un 0.
  w <- ifelse(length(vini) == ncol(datos) + 1, vini, c(vini, 0))
```

```

# Variables usadas en el bucle
changing <- T
iteraciones <- 0

# Añadimos una columna de unos a los datos
datos <- cbind(datos, 1)

# Bucle principal, del que salimos si no ha habido cambios tras una
# pasada completa a los datos (solución encontrada) o si se ha llegado
# al máximo de iteraciones permitidas (solución no encontrada)
while(changing && iteraciones < max_iter){
  iteraciones <- iteraciones+1

  changing <- F

  # Bucle sobre toda la muestra
  for(index in seq(label)){
    dato <- datos[index,]
    etiq <- label[index]

    # Comportamiento principal: si la muestra está mal etiquetada,
    # recalculamos el hiperplano para etiquetarla bien.
    if(sign(w %*% dato) != etiq) {
      w <- w + etiq*dato
      changing <- T
    }
  }
}

# Devolvemos los pesos, el vector (a,b), coeficientes que determinan la
# recta  $y = ax + b$  y el número de iteraciones.
return(list(Coefs = w, Recta = -c(w[1], w[3]) / w[2], Iter = iteraciones))
}

```

Esta función implementa el algoritmo tal y como lo vimos en clase. La función espera que se reciban los datos originales, sin la columna de unos necesaria para hacer el producto del bucle, así que antes del bucle se añade esta columna. Además, la función se puede llamar con vector inicial sin el b del final, en cuyo caso se añade $b = 0$.

En el bucle principal se encuentra el algoritmo: se itera sobre todos los datos ajustando el vector de pesos w en caso de que la etiqueta contemplada esté mal etiquetada. Cuando no haya cambios tras hacer una pasada completa a la muestra —en cuyo caso se ha conseguido la solución óptima— o se llegue al máximo número de iteraciones —en cuyo caso no se ha conseguido la solución óptima, bien porque los datos no sean linealmente separables o porque sean necesarias más iteraciones—, el algoritmo termina.

Se devuelve el vector de pesos y el número de iteraciones que han sido necesarias. Además, se devuelve la recta generada con los pesos; como el hiperplano considerado en todos los ejemplos posteriores es particularmente una recta, esto evita tener que hacer esta operación siempre que llamemos a la función y necesitemos la recta.

Ejercicio 2

La primera parte de este ejercicio consiste en llamar a la función anteriormente implementada con el vector inicial $(0, 0, 0)$ sobre los datos generados en el ejercicio 6 de la anterior sección.

Preparamos entonces los datos generados allí —`unif.x` y `unif.y`— en la forma admitida por la función:

```
# Damos forma de matriz a los datos y generamos el vector de etiquetas
datos.matriz <- matrix(c(unif.x, unif.y), ncol = 2)
datos.etiqueta <- etiquetado1(unif.x, unif.y)
```

Ejecutamos la función implementada anteriormente sobre esos datos, con un número máximo de iteraciones muy alto para que encuentre la solución óptima —existe porque los datos son linealmente separables—:

```
# Ejecución del algoritmo
out_PLA <- ajusta_PLA(datos.matriz, datos.etiqueta, 10000000, c(0,0))
```

```
## Warning in if (sign(w %*% dato) != etiq) {: la condición tiene longitud > 1
## y sólo el primer elemento será usado
```

```
# Guardamos la solución encontrada y el número de iteraciones:
solucion <- out_PLA$Recta
iteraciones <- out_PLA$Iter
```

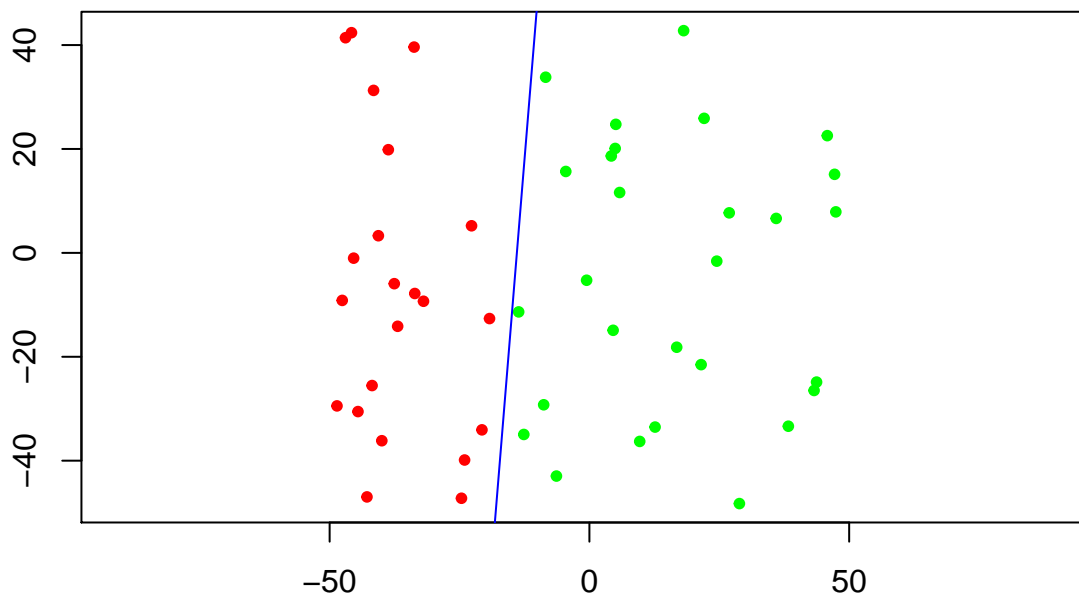
En este caso, el número de iteraciones es de 93.

Podemos también ver la solución devuelta por el algoritmo junto con las muestras usadas; para ello, generamos un gráfico como hemos hecho en ocasiones anteriores:

```
# Generamos el vector de colores basado en las etiquetas
colores <- ifelse(datos.etiqueta == 1, "green", "red")

# Generamos el gráfico de la muestra
plot(datos.matriz, asp = 1, col = colores, pch = 20,
      main="PLA sobre muestra uniforme", xlab="", ylab="")
# Añadimos la recta devuelta
abline(rev(solucion), col="blue")
```

PLA sobre muestra uniforme



Ahora hacemos una prueba con vectores iniciales aleatorios. Vamos a usar la función `lapply`, así que primero necesitamos encapsular los pasos que queremos dar en una función, que recibe como único parámetro el vector inicial con el que llamaremos al PLA:

```
# Encapsula ajusta_PLA para llamar a lapply
wrapper <- function(vector){
  datos <- ajusta_PLA(datos.matriz, datos.etiqueta, 10000000, vector)
  return(datos$Iter)
}
```

Ahora llamamos a esa función con 10 vectores aleatorios de 3 componentes dentro de un `lapply`. Esto nos devuelve una lista de iteraciones, así que lo convertimos en un vector y calculamos su media:

```
# Ejecutamos PLA 10 veces con vectores aleatorios en [0,1]
inicio_aleatorio <- lapply(simula_unif(10, 3, c(0,1)), wrapper)
```

```
## Warning in if (sign(w %%% dato) != etiq) {: la condición tiene longitud > 1
## y sólo el primer elemento será usado
```

```
## Warning in if (sign(w %%% dato) != etiq) {: la condición tiene longitud > 1
## y sólo el primer elemento será usado
```

```
## Warning in if (sign(w %%% dato) != etiq) {: la condición tiene longitud > 1
## y sólo el primer elemento será usado
```

```
## Warning in if (sign(w %%% dato) != etiq) {: la condición tiene longitud > 1
## y sólo el primer elemento será usado
```

```
## Warning in if (sign(w %%% dato) != etiq) {: la condición tiene longitud > 1
## y sólo el primer elemento será usado
```

```
## Warning in if (sign(w %%% dato) != etiq) {: la condición tiene longitud > 1
## y sólo el primer elemento será usado
```

```
## Warning in if (sign(w %%% dato) != etiq) {: la condición tiene longitud > 1
## y sólo el primer elemento será usado
```

```
## Warning in if (sign(w %%% dato) != etiq) {: la condición tiene longitud > 1
## y sólo el primer elemento será usado
```

```
## Warning in if (sign(w %%% dato) != etiq) {: la condición tiene longitud > 1
## y sólo el primer elemento será usado
```

```
## Warning in if (sign(w %%% dato) != etiq) {: la condición tiene longitud > 1
## y sólo el primer elemento será usado
```

```
## Warning in if (sign(w %%% dato) != etiq) {: la condición tiene longitud > 1
## y sólo el primer elemento será usado
```

```
## Warning in if (sign(w %%% dato) != etiq) {: la condición tiene longitud > 1
## y sólo el primer elemento será usado
```

```
## Warning in if (sign(w %%% dato) != etiq) {: la condición tiene longitud > 1
```

[illegible]

[illegible]

```
## y sólo el primer elemento será usado

## Warning in if (sign(w %*% dato) != etiq) {: la condición tiene longitud > 1
## y sólo el primer elemento será usado

## Warning in if (sign(w %*% dato) != etiq) {: la condición tiene longitud > 1
## y sólo el primer elemento será usado

## Warning in if (sign(w %*% dato) != etiq) {: la condición tiene longitud > 1
## y sólo el primer elemento será usado

## Warning in if (sign(w %*% dato) != etiq) {: la condición tiene longitud > 1
## y sólo el primer elemento será usado

## Warning in if (sign(w %*% dato) != etiq) {: la condición tiene longitud > 1
## y sólo el primer elemento será usado

# Calculamos media de iteraciones
media <- mean(unlist(inicio_aleatorio))
```

Como vemos, la media es de 76.6, algo menor que en el caso anterior. Así que en este caso y ante estos datos, parece más sensato llamar a esta función con vectores aleatorios.

Ejercicio 3

En este ejercicio analizamos cómo se comporta el algoritmo PLA con un número fijo de iteraciones ante datos no linealmente separables —reusamos los datos del ejercicio 8, cuyas etiquetas tenían un 10% de ruido en cada clase—.

Para realizar este experimento encapsulamos la tarea en la función `analisis_PLA`, cuya implementación es la siguiente:

```
# Analiza el algoritmo PLA ante un número fijo de iteraciones:
analisis_PLA <- function(max_iter, dat.x, dat.y, dat.etiquetas){

  # Damos forma a los datos para pasárselos a ajusta_PLA
  datos.matriz <- matrix(c(dat.x, dat.y), ncol = 2)

  # Llamamos a la función y guardamos la recta solución
  out_PLA <- ajusta_PLA(datos.matriz, dat.etiquetas, max_iter, c(0,0))
  recta <- out_PLA$Recta

  # Clasificamos con la función etiquetadora determinada por la recta
  nuevo_etiquetado <- generador_etiquetados(function(x, y){ y - recta[1]*x - recta[2] })
  nuevas_etiquetas <- nuevo_etiquetado(dat.x, dat.y)

  # Devolvemos el número de muestras mal etiquetadas:
  return(sum(nuevas_etiquetas != dat.etiquetas))
}
```

La función `analisis_PLA` llama al algoritmo PLA, etiqueta la muestra con la recta solución y devuelve el número de muestras mal etiquetadas.

Para llamarlo ejecutamos la función con `lapply` sobre un vector con el número de iteraciones: 10, 100 y 1000.

```
# Llamamos a la función con 10, 100 y 1000 iteraciones
iteraciones <- c(10, 100, 1000)
analisis <- lapply(iteraciones, analisis_PLA, datos$X, datos$Y, datos$Etiqueta)
```

```
## Warning in if (sign(w %>% dato) != etiq) {: la condición tiene longitud > 1
## y sólo el primer elemento será usado
```

```
## Warning in if (sign(w %>% dato) != etiq) {: la condición tiene longitud > 1
## y sólo el primer elemento será usado
```

```
## Warning in if (sign(w %>% dato) != etiq) {: la condición tiene longitud > 1
## y sólo el primer elemento será usado
```

La ejecución del anterior trozo de código nos da el siguiente resultado:

- Número de etiquetas diferentes con 10 iteraciones: 31
- Número de etiquetas diferentes con 100 iteraciones: 33
- Número de etiquetas diferentes con 1000 iteraciones: 26

Es claro ante esta salida que un número de iteraciones mayor no implica un número menor de errores. Si los datos no son linealmente separables, el algoritmo PLA está constantemente iterando, pero no se acerca a la mejor solución de forma iterativa. Nada nos asegura que el número de errores sea mejor tras un millón de iteraciones que tras una.

Ejercicio 4

Realizamos el mismo experimento que antes, esta vez sobre el etiquetado de la segunda función del ejercicio 7 de la sección anterior. Estos datos tampoco son linealmente separables, no ya porque se haya generado ruido, sino porque el etiquetado original no separaba linealmente las muestras.

De forma análoga al ejercicio anterior, ejecutamos la función `analisis_PLA` con el etiquetado guardado en `etiquetas_2`

```
# Llamamos a la función con 10, 100 y 1000 iteraciones
iteraciones <- c(10, 100, 1000)
analisis <- lapply(iteraciones, analisis_PLA, unif.x, unif.y, etiquetas_2)
```

```
## Warning in if (sign(w %>% dato) != etiq) {: la condición tiene longitud > 1
## y sólo el primer elemento será usado
```

```
## Warning in if (sign(w %>% dato) != etiq) {: la condición tiene longitud > 1
## y sólo el primer elemento será usado
```

```
## Warning in if (sign(w %>% dato) != etiq) {: la condición tiene longitud > 1
## y sólo el primer elemento será usado
```

La ejecución del anterior trozo de código nos da el siguiente resultado:

- Número de etiquetas diferentes con 10 iteraciones: 39
- Número de etiquetas diferentes con 100 iteraciones: 17
- Número de etiquetas diferentes con 1000 iteraciones: 40

Ante estos datos se refuerza el anterior comentario: nada pueda asegurar, ante datos no linealmente separables, que un número mayor de iteraciones produzca un mejor resultado.

Ejercicio 5

Modificamos ahora la función `ajusta_PLA` de manera que en cada iteración —esto es, tras cada pasada completa de los datos— se redibuje la gráfica con las muestras y la recta encontrada hasta ese momento. Para hacer esto basta llamar a `plot` y `abline` para generar el gráfico y llamar a `Sys.sleep()` para pausar la ejecución y poder ver la animación. La función modificada se queda como sigue:

```
# Implementa el algoritmo Perceptron. Devuelve una lista con dos valores:
# Sol: Recta solución
# Iter: Número de iteraciones que han sido necesarias
ajusta_PLA_anim <- function(datos, label, max_iter, vini){

  # Definimos w como el vector inicial. Si tiene una posición más que
# el número de datos, lo dejamos como está; si no, añadimos un 0.
  w <- ifelse(length(vini) == ncol(datos) + 1, vini, c(vini, 0))

  # Variables usadas en el bucle
  changing <- T
  iteraciones <- 0

  # Añadimos una columna de unos a los datos
  datos <- cbind(datos, 1)

  # Generamos el vector de colores basado en las etiquetas
  colores <- ifelse(label == 1, "green", "red")

  # Bucle principal, del que salimos si no ha habido cambios tras una
# pasada completa a los datos (solución encontrada) o si se ha llegado
# al máximo de iteraciones permitidas (solución no encontrada)
  while(changing && iteraciones < max_iter){
    iteraciones <- iteraciones+1

    changing <- F

    # Bucle sobre toda la muestra
    for(index in seq(label)){
      dato <- datos[index,]
      etiq <- label[index]

      # Comportamiento principal: si la muestra está mal etiquetada,
# recalculamos el hiperplano para etiquetarla bien.
      if(sign(w %*% dato) != etiq) {
        w <- w + etiq*dato
        changing <- T
      }
    }

    # Generamos la gráfica con la muestra coloreada
    plot(tail(datos,-1), asp = 1, col = colores, pch = 20,
         main=paste("Ajustando PLA con",max_iter,"iteraciones"), xlab="", ylab="")
    # Añadimos la recta actual
    abline(-c(w[3], w[1]) / w[2], col="blue")
    # Pausamos 0.1 segundos
    Sys.sleep(0.2)
  }
}
```

```

}

# Redibujamos la recta con más grosor
abline(-c(w[3], w[1]) / w[2], col="blue", lwd=2)
# Pausamos 1 segundo
Sys.sleep(1)

# Devolvemos los pesos, el vector (a,b), coeficientes que determinan la
# recta  $y = ax + b$  y el número de iteraciones.
return(list(Coefs = w, Recta = -c(w[1], w[3]) / w[2], Iter = iteraciones))
}

```

Basta ahora ejecutar el siguiente código para ver las tres animaciones que se generan:

```

# Damos forma a los datos
datos.matriz <- matrix(c(datos$X, datos$Y), ncol = 2)

# Llamamos a la función con 10, 100 y 1000 iteraciones
for(max_iter in c(10,100,1000)){
  ajusta_PLA_anim(datos.matriz, datos$Etiqueta, max_iter, c(0,0))
}

```

Ejercicio 6

La modificación del algoritmo PLA es sencilla: tratamos ahora de quedarnos con la mejor solución. Basta ir almacenando la recta que menor etiquetas erróneas tenga, de manera que tras el número máximo de iteraciones, si no se ha encontrado la óptima, se devuelva la mejor de entre las analizadas. Esta modificación es, de hecho, el algoritmo *Pocket*. La función modificada es la siguiente:

```

# Implementa el algoritmo Perceptron. Devuelve una lista con dos valores:
# Sol: Recta solución
# Iter: Número de iteraciones que han sido necesarias
ajusta_PLA_MOD <- function(datos, label, max_iter, vini){

  # Definimos w como el vector inicial. Si tiene una posición más que
  # el número de datos, lo dejamos como está; si no, añadimos un 0.
  w <- ifelse(length(vini) == ncol(datos) + 1, vini, c(vini, 0))

  # Variables usadas en el bucle
  changing <- T
  iteraciones <- 0

  # Añadimos una columna de unos a los datos
  datos <- cbind(datos, 1)

  # Inicializamos el número de errores al total de datos recibidos
  mejor_error <- length(datos[,1])

  # Bucle principal, del que salimos si no ha habido cambios tras una
  # pasada completa a los datos (solución encontrada) o si se ha llegado
  # al máximo de iteraciones permitidas (solución no encontrada)
  while(changing && iteraciones < max_iter){

```

```

iteraciones <- iteraciones+1

changing <- F

# Bucle sobre toda la muestra
for(index in seq(label)){
  dato <- datos[index,]
  etiq <- label[index]

  # Comportamiento principal: si la muestra está mal etiquetada,
# recalculamos el hiperplano para etiquetarla bien.
  if(sign(w %*% dato) != etiq) {
    w <- w + etiq*dato
    changing <- T
  }
}

# Definimos la recta generada por los pesos
recta <- -c(w[1], w[3]) / w[2]

# Etiquetamos la muestra con la nueva recta
nuevo_etiquetado <- generador_etiquetados(function(x, y){ y - recta[1]*x - recta[2] })
nuevas_etiquetas <- nuevo_etiquetado(datos[,1], datos[,2])

# Calculamos el número de muestras mal etiquetadas con la recta actual
error_actual <- sum(nuevas_etiquetas != label)

# Si el error actual es mejor que el mejor encontrado hasta ahora, guardamos
# los pesos actuales y actualizamos el mejor error
if(error_actual < mejor_error){
  mejor_error <- error_actual
  mejor_solucion <- w
}

}

# Actualizamos w con la mejor solución
w <- mejor_solucion

# Devolvemos los pesos, el vector (a,b), coeficientes que determinan la
# recta  $y = ax + b$  y el número de iteraciones.
return(list(Coefs = w, Recta = -c(w[1], w[3]) / w[2], Iter = iteraciones))
}

```

La única modificación se encuentra tras el bucle que recorre todas las muestras. Después de cada pasada se etiqueta la muestra con la recta encontrada en ese momento y se contabiliza el número de muestras mal etiquetadas. Si ese error es mejor que el mejor encontrado hasta el momento, se guarda la solución actual y se actualiza el error. La función devuelve entonces la mejor solución encontrada.

Para probar la modificación vamos a comparar el número de mejoras que se producen con la muestra uniforme y los etiquetados generados en el ejercicio 7 de la sección anterior. Para ello, modificamos nuestra función de análisis para que el primer parámetro, `dat.etiquetas`, sea el vector de etiquetas y para que reciba la función con la que ajustar PLA: el parámetro `ajuste`. La función modificada queda así:

```

# Analiza el algoritmo PLA ante un número fijo de iteraciones:
analisis_PLA_MOD <- function(dat.etiquetas, ajuste, dat.x, dat.y){

  # Damos forma a los datos para pasárselos a ajusta_PLA
  datos.matriz <- matrix(c(dat.x, dat.y), ncol = 2)

  # Llamamos a la función y guardamos la recta solución
  out_PLA <- ajuste(datos.matriz, dat.etiquetas, 1000, c(0,0))
  recta <- out_PLA$Recta

  # Clasificamos con la función etiquetadora determinada por la recta
  nuevo_etiquetado <- generador_etiquetados(function(x, y){ y - recta[1]*x - recta[2] })
  nuevas_etiquetas <- nuevo_etiquetado(dat.x, dat.y)

  # Devolvemos el número de muestras mal etiquetadas:
  return(sum(nuevas_etiquetas != dat.etiquetas))
}

```

Llamamos ahora a esta función dos veces, una con la función `ajusta_PLA` y otra con `ajusta_PLA_MOD`, en ambos casos iterando sobre la lista de etiquetas generadas en el apartado 7:

```

# Lista de etiquetas del ejercicio 7 de la sección anterior
lista_etiquetas <- list(etiquetas_2, etiquetas_3, etiquetas_4, etiquetas_5)

# Generamos un vector con el número de muestras mal etiquetadas con ajusta_PLA
errores <- unlist(lapply(lista_etiquetas, analisis_PLA_MOD, ajusta_PLA, unif.x, unif.y))

## Warning in if (sign(w %*% dato) != etiq) {: la condición tiene longitud > 1
## y sólo el primer elemento será usado

## Warning in if (sign(w %*% dato) != etiq) {: la condición tiene longitud > 1
## y sólo el primer elemento será usado

## Warning in if (sign(w %*% dato) != etiq) {: la condición tiene longitud > 1
## y sólo el primer elemento será usado

## Warning in if (sign(w %*% dato) != etiq) {: la condición tiene longitud > 1
## y sólo el primer elemento será usado

# Generamos un vector con el número de muestras mal etiquetadas con ajusta_PLA_MOD
errores_MOD <- unlist(lapply(lista_etiquetas, analisis_PLA_MOD, ajusta_PLA_MOD, unif.x, unif.y))

## Warning in if (sign(w %*% dato) != etiq) {: la condición tiene longitud > 1
## y sólo el primer elemento será usado

## Warning in if (sign(w %*% dato) != etiq) {: la condición tiene longitud > 1
## y sólo el primer elemento será usado

## Warning in if (sign(w %*% dato) != etiq) {: la condición tiene longitud > 1
## y sólo el primer elemento será usado

## Warning in if (sign(w %*% dato) != etiq) {: la condición tiene longitud > 1
## y sólo el primer elemento será usado

```

El resultado devuelto por el anterior código se resume así:

- El algoritmo pocket tiene 34 muestras erróneas menos que el algoritmo PLA original con la función f_2 .
- El algoritmo pocket tiene 36 muestras erróneas menos que el algoritmo PLA original con la función f_3 .
- El algoritmo pocket tiene 35 muestras erróneas menos que el algoritmo PLA original con la función f_4 .
- El algoritmo pocket tiene 0 muestras erróneas menos que el algoritmo PLA original con la función f_5 .

Como vemos, la mejora con el algoritmo pocket es evidente.

Regresión lineal

Ejercicio 2

```
# Leemos el fichero
digitos <- read.table("datos/zip.train")

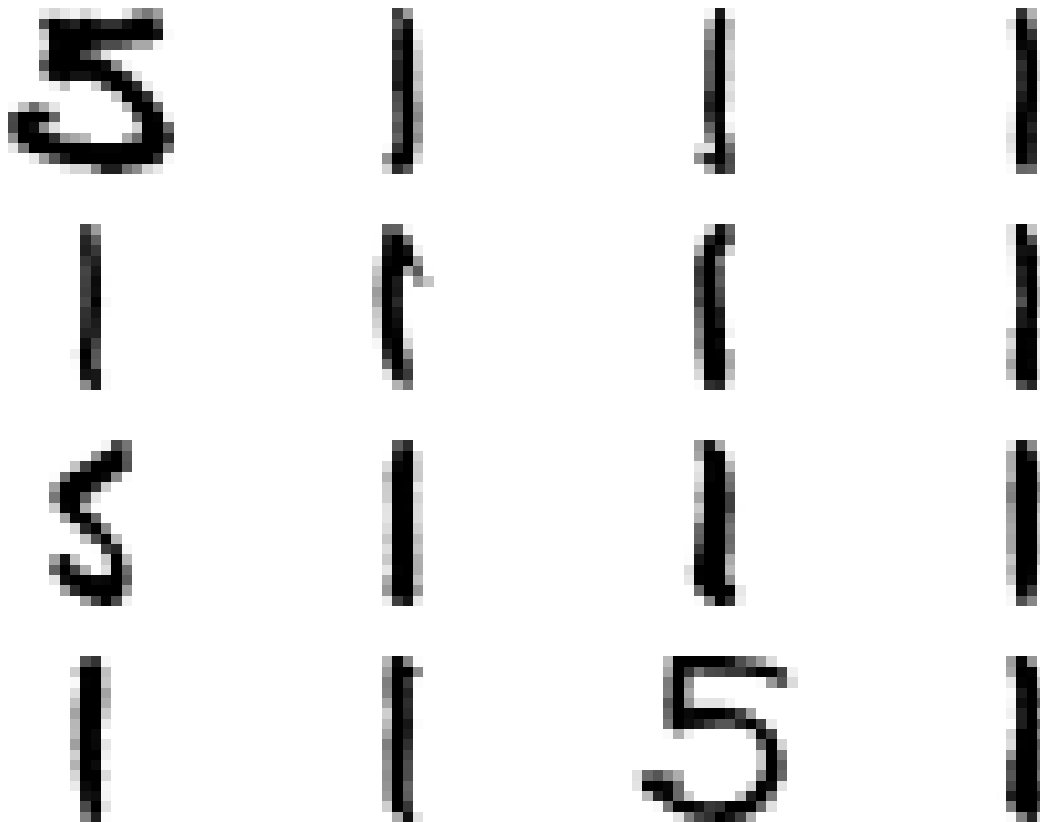
## Warning in scan(file, what, nmax, sep, dec, quote, skip, nlines,
## na.strings, : número de items leídos no es múltiplo del número de columnas

# Nos quedamos únicamente con los números 1 y 5
digitos <- digitos[ is.element(digitos[,1], c(1,5)), ]
digitos.unos <- digitos[ digitos[,1] == 1, ]
digitos.cincos <- digitos[ digitos[,1] == 5, ]

prev_mar <- par()$mar

par(mfrow = c(4, 4), "mar" = c(1,1,1,1))

apply(digitos[1:16,], 1, function(digito) {
  numero <- matrix(tail(digito, -1), nrow = 16)
  numero <- numero[, ncol(numero):1]
  image(numero, col = gray(seq(1,0,length=256)), axes = F, pty="s", asp=1)
})
```

```
## NULL
```

```
par(mfrow=c(1, 1), "mar" = prev_mar)
```

Ejercicio 3

```
analisis_digito <- function(digito){
  numero <- matrix(tail(digito, -1), nrow = 16)
  numero.inv <- numero[,ncol(numero):1]

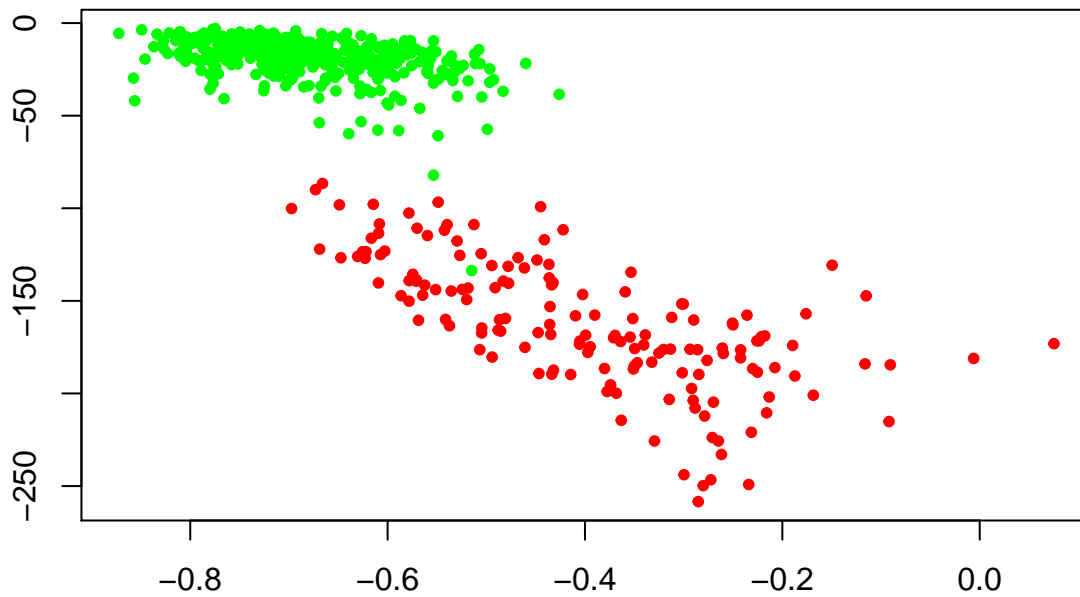
  intensidad <- mean(digito)
  simetria <- -sum( abs(numero.inv - numero) )

  c(head(digito, 1), intensidad, simetria)
}
```

Ejercicio 4

```
digitos.analisis <- t(apply(digitos, 1, analisis_digito))

colores <- ifelse(digitos.analisis[,1] == 1, "green", "red")
plot(digitos.analisis[,2:3], col = colores, pch = 20)
```



Ejercicio 5

```
library(MASS)

Regress_Lin <- function(datos, label){
  X <- cbind(datos, 1)
  Y <- label

  SVD <- svd(X)
  D <- diag(SVD$d)
  U <- SVD$u
  V <- SVD$v

  XtX_inv <- V %*% ginv(D ** 2) %*% t(V)
  X_pseudoinv <- XtX_inv %*% t(X)

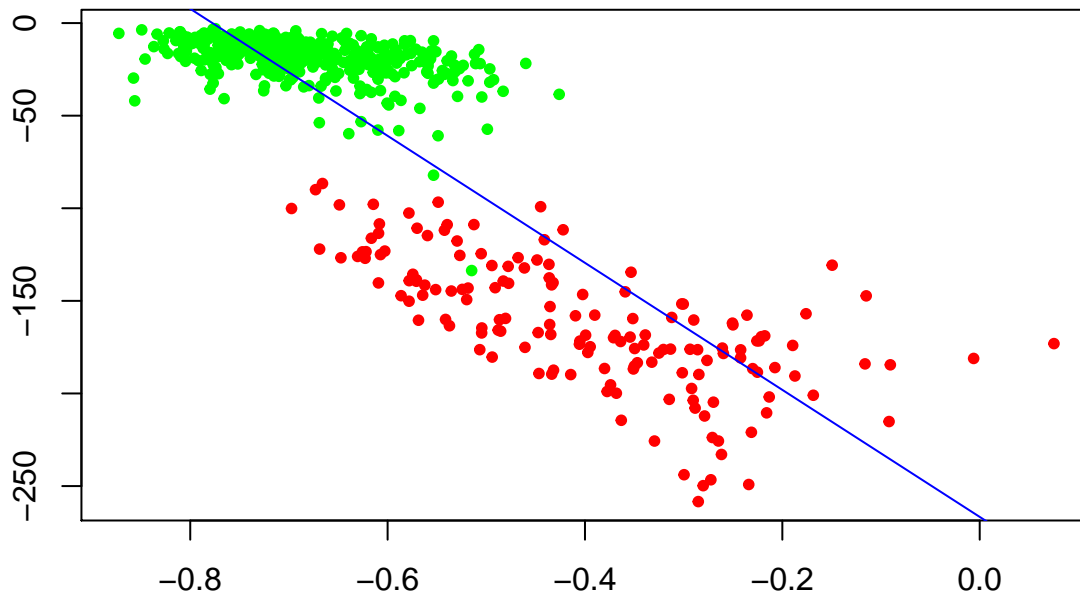
  sol <- X_pseudoinv %*% Y

  return(sol)
}
```

Ejercicio 6

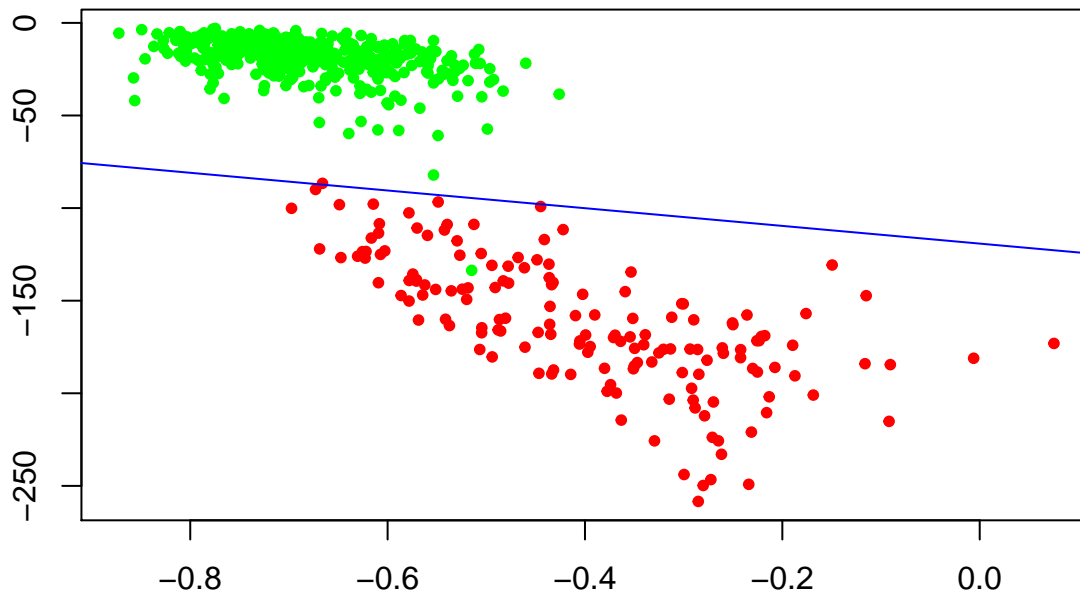
```
digitos.regresion <- Regress_Lin(digitos.analisis[, 2], digitos.analisis[, 3])

colores <- ifelse(digitos.analisis[,1] == 1, "green", "red")
plot(digitos.analisis[,2:3], col = colores, pch=20)
abline(rev(digitos.regresion), col="blue")
```



```
digitos.clasificacion <- Regress_Lin(digitos.analisis[, 2:3], ifelse(digitos.analisis[, 1] == 1, 1, -1))

plot(digitos.analisis[,2:3], col = colores, pch=20)
digitos.recta_clas <- -c(digitos.clasificacion[3], digitos.clasificacion[1]) / digitos.clasificacion[2]
abline(digitos.recta_clas, col="blue")
```



Ejercicio 7

```
generar_muestra <- function(tam = 100){
  datos.muestra <- simula_unif(N = tam, dim = 2, rango = c(-10, 10))

  datos.x <- unlist(lapply(datos.muestra, '[', 1))
  datos.y <- unlist(lapply(datos.muestra, '[', 2))
}
```

```

    return(matrix(c(datos.x, datos.y), ncol = 2))
}

generar_etiquetado <- function(muestra){
  recta <- simula_recta(intervalo = c(-10, 10))

  f <- function(x,y){
    y - recta[1]*x - recta[2]
  }

  etiquetador <- generador_etiquetados(f)
  etiquetas <- etiquetador(muestra[, 1], muestra[, 2])

  return(list(Foo = f, Recta = recta, Etiquetas = etiquetas, Etiquetador = etiquetador))
}

```

Apartado a

```

reg.muestra <- generar_muestra(100)

medir_Ein <- function(muestra){
  f.etiquetado <- generar_etiquetado(muestra)

  regresion <- Regress_Lin(muestra, f.etiquetado$Etiquetas)
  g.recta <- -c(regresion[1], regresion[3]) / regresion[2]

  g <- function(x,y){
    y - g.recta[1]*x - g.recta[2]
  }

  g.etiquetador <- generador_etiquetados(g)
  g.etiquetas <- g.etiquetador(muestra[, 1], muestra[, 2])

  E_in <- sum(g.etiquetas != f.etiquetado$Etiquetas)

  return(E_in / nrow(muestra))
}

num_rep <- 1000

E_in <- sum(replicate(num_rep, medir_Ein(reg.muestra))) / num_rep
mensaje <- paste("Error medio dentro de la muestra tras", num_rep, "repeticiones:", E_in * 100, "%")
print(mensaje)

```

```
## [1] "Error medio dentro de la muestra tras 1000 repeticiones: 5.586 %"
```

Apartado b

```

reg.muestra <- generar_muestra(100)

medir_Eout <- function(muestra){
  # Generamos etiquetado con f
  f.etiquetado <- generar_etiquetado(muestra)

  # Estimamos g
  regresion <- Regress_Lin(muestra, f.etiquetado$Etiquetas)
  g.recta <- -c(regresion[1], regresion[3]) / regresion[2]

  g <- function(x,y){
    y - g.recta[1]*x - g.recta[2]
  }

  # Generamos una nueva muestra
  test.muestra <- generar_muestra(1000)

  # Generamos etiquetado con f de la nueva muestra
  test.etiquetas <- f.etiquetado$Etiquetador(test.muestra[, 1], test.muestra[, 2])

  # Generamos etiquetado con g de la nueva muestra
  g.etiquetador <- generador_etiquetados(g)
  g.etiquetas <- g.etiquetador(test.muestra[, 1], test.muestra[, 2])

  E_out <- sum(g.etiquetas != test.etiquetas)

  return(E_out / nrow(test.muestra))
}

num_rep <- 1000

E_out <- sum(replicate(num_rep, medir_Eout(reg.muestra))) / num_rep
mensaje <- paste("Error medio fuera de la muestra tras", num_rep, "repeticiones:", E_out * 100, "%")
print(mensaje)

```

```
## [1] "Error medio fuera de la muestra tras 1000 repeticiones: 6.8664 %"
```

Apartado c

```

#TODO: mirar qué pasa con la recta si w[2] es cero
muestra <- generar_muestra(10)

medir_iter_PLA <- function(muestra){
  # Generamos etiquetado con f
  f.etiquetado <- generar_etiquetado(muestra)

  # Estimamos g
  regresion <- Regress_Lin(muestra, f.etiquetado$Etiquetas)
  PLA <- ajusta_PLA(datos = muestra, label = f.etiquetado$Etiquetas, max_iter = 1000, vini = regresion

```


[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

```
mensaje <- paste("Número medio de iteraciones tras", num_rep, "repeticiones:", num_iter)
print(mensaje)
```

```
## [1] "Número medio de iteraciones tras 1000 repeticiones: 40.005"
```

Ejercicio 8

```
# Definición de la función f y la función etiquetadora
f <- function(x1, x2){
  x1 ** 2 + x2 ** 2 - 25
}

f.etiquetado <- generador_etiquetados(f)
```

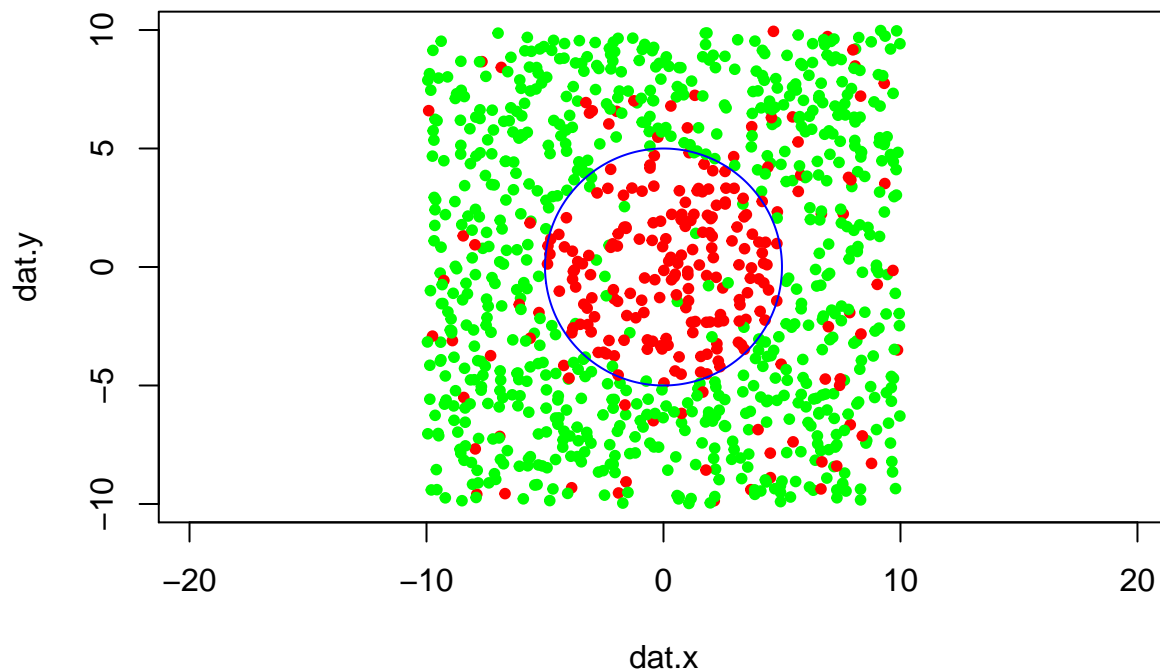
Generamos la muestra, las etiquetas y el ruido:

```
# Generación de la muestra y sus etiquetas con f
muestra <- generar_muestra(1000)

f.etiquetas <- f.etiquetado(muestra[, 1], muestra[, 2])

ruido.indices <- sample(1000, 100)
f.etiquetas[ruido.indices] <- -f.etiquetas[ruido.indices]

genera_grafico(muestra[, 1], muestra[, 2], f, f.etiquetas)
```



```
## [1] 1 1 1 1 1 1 1 -1 1 -1 1 1 1 1 1 1 1 1 1 1
## [24] 1 1 -1 1 -1 1 -1 1 -1 1 1 1 -1 -1 -1 1 -1 1 1 1
```

```

## [47] 1 -1 -1 1 1 1 -1 1 1 -1 1 1 1 1 1 1 1 1 1 1 1 -1
## [70] 1 1 1 1 1 1 -1 1 1 1 1 1 -1 1 1 1 1 1 1 -1 1 1
## [93] 1 -1 1 -1 1 1 1 1 1 1 1 1 1 1 1 -1 -1 -1 1 1 1 -1 1
## [116] 1 1 1 1 1 1 -1 1 -1 1 1 1 1 1 -1 1 -1 1 1 1 1 -1 1
## [139] 1 1 1 1 -1 -1 1 1 -1 -1 1 -1 1 1 1 -1 1 1 1 1 -1 1 1
## [162] 1 1 1 1 -1 -1 1 1 1 1 1 1 1 1 1 1 1 -1 -1 -1 1 1 1
## [185] -1 -1 -1 -1 1 -1 1 1 1 1 -1 1 1 -1 1 -1 1 -1 -1 1 1 1 -1
## [208] 1 1 1 1 1 1 -1 -1 1 1 -1 1 1 -1 -1 1 -1 1 1 1 1 1 1
## [231] -1 1 1 -1 1 -1 -1 -1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1
## [254] -1 -1 -1 1 1 1 -1 1 1 -1 -1 -1 1 1 1 1 1 1 -1 1 -1 1 1
## [277] -1 1 1 -1 1 -1 1 1 1 1 1 1 1 1 1 1 1 1 1 -1 1 1 1
## [300] 1 1 1 1 1 1 1 1 1 -1 1 -1 -1 -1 -1 1 -1 1 1 1 -1 1 1
## [323] -1 -1 -1 -1 1 -1 -1 1 1 1 -1 1 1 -1 1 1 -1 1 -1 1 -1 1 1
## [346] -1 1 1 1 1 -1 -1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1
## [369] 1 1 1 -1 -1 1 1 1 -1 1 1 1 1 1 1 -1 1 -1 1 1 -1 1 1
## [392] 1 1 1 1 1 -1 1 1 1 1 1 1 1 -1 1 1 -1 -1 -1 -1 1 1 1
## [415] 1 1 1 1 1 1 1 1 1 -1 1 1 -1 1 -1 1 -1 1 -1 1 -1 1 1
## [438] 1 1 1 1 -1 1 -1 1 1 -1 1 1 1 1 -1 1 1 1 1 1 -1 -1 1
## [461] 1 -1 1 1 -1 -1 1 1 1 1 1 1 -1 -1 1 -1 -1 1 1 1 -1 -1
## [484] -1 1 -1 1 -1 -1 1 1 1 1 1 1 -1 1 -1 1 1 1 1 -1 -1 1
## [507] 1 -1 -1 1 -1 -1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 -1 -1
## [530] 1 1 1 1 1 1 1 1 1 1 1 1 -1 1 1 1 1 1 -1 1 1 -1 1
## [553] -1 1 1 1 -1 -1 1 1 1 1 1 1 1 1 -1 1 1 -1 1 1 1 1
## [576] 1 1 1 1 1 1 1 1 1 -1 -1 1 -1 -1 1 1 1 1 1 1 -1 1 -1
## [599] 1 1 -1 1 1 1 1 1 1 1 1 1 1 1 1 -1 -1 -1 1 1 1 1
## [622] 1 1 -1 1 -1 1 -1 1 1 1 -1 1 1 1 1 -1 1 -1 1 -1 1 -1
## [645] 1 1 -1 1 -1 1 1 -1 1 -1 1 1 1 1 -1 1 1 1 1 1 1 1
## [668] 1 1 1 -1 1 1 1 1 1 -1 -1 -1 1 -1 -1 1 -1 1 1 -1 1 1
## [691] 1 1 1 1 1 1 1 1 1 -1 1 1 1 1 -1 1 1 1 -1 1 1 -1
## [714] 1 -1 1 -1 1 1 1 -1 -1 1 1 1 -1 1 -1 -1 1 1 -1 1 1 1
## [737] 1 1 -1 1 -1 -1 1 1 -1 -1 1 1 1 1 1 1 1 1 1 1 -1 1
## [760] 1 1 1 1 1 1 1 1 1 -1 1 1 1 1 -1 1 1 -1 1 1 1 1
## [783] 1 1 1 1 -1 -1 -1 1 1 1 -1 1 -1 1 1 1 1 1 1 -1 -1 -1
## [806] -1 -1 1 1 1 1 1 -1 -1 1 1 -1 1 1 1 1 1 1 -1 1 -1 1
## [829] 1 -1 -1 1 1 1 1 1 1 1 1 1 1 1 -1 1 1 1 -1 1 -1 -1
## [852] -1 1 -1 1 1 1 1 -1 -1 1 1 1 -1 1 1 1 1 1 1 1 -1 -1
## [875] -1 -1 1 -1 -1 1 1 1 1 1 1 -1 1 1 1 1 1 1 1 1 1 1
## [898] 1 1 1 -1 1 1 1 1 1 1 1 1 1 1 -1 -1 1 1 1 1 1 -1
## [921] -1 1 1 1 1 1 1 -1 1 -1 1 -1 1 1 1 1 1 1 1 1 1 1
## [944] 1 1 1 1 -1 1 1 1 1 1 1 1 1 1 1 1 -1 1 1 1 -1 -1
## [967] 1 1 -1 -1 1 1 1 1 1 1 1 1 1 -1 1 1 1 1 1 1 1 1
## [990] 1 1 1 1 -1 1 1 1 1 1 1 1

```

Medimos el error dentro de la muestra con regresión lineal repitiendo el experimento 1000 veces:

```

medir_Ein <- function(){
  muestra <- generar_muestra(1000)

  etiquetas <- f.etiquetado(muestra[, 1], muestra[, 2])

  ruido.indices <- sample(1000, 100)
  etiquetas[ruido.indices] <- -etiquetas[ruido.indices]

  regresion <- Regress_Lin(muestra, etiquetas)

```

```

g.recta <- -c(regresion[1], regresion[3]) / regresion[2]

g <- function(x,y){
  y - g.recta[1]*x - g.recta[2]
}

g.etiquetador <- generador_etiquetados(g)
g.etiquetas <- g.etiquetador(muestra[, 1], muestra[, 2])

E_in <- sum(g.etiquetas != etiquetas)

return(E_in / nrow(muestra))
}

num_rep <- 1000

E_in <- sum(replicate(num_rep, medir_Ein())) / num_rep
mensaje <- paste("Error medio dentro de la muestra tras", num_rep, "repeticiones:", E_in * 100, "%")
print(mensaje)

```

```
## [1] "Error medio dentro de la muestra tras 1000 repeticiones: 49.3701 %"
```

Hacemos la transformación no lineal de los datos:

```

m.x <- muestra[, 1]
m.y <- muestra[, 2]
muestra.trans <- cbind(m.x, m.y, m.x * m.y, m.x ** 2, m.y ** 2)

w <- Regress_Lin(muestra.trans, f.etiquetas)

```

Dibujamos el resultado:

Medimos el error dentro de la muestra con los datos transformados

Medimos el error fuera de la muestra con los datos transformados: