Trabajo 1: Programación

Gema Correa Fernández

27 de Marzo de 2017

Nota: *Antes de comenzar a ejecutar todo el código, inicializamos una semilla a un valor por defecto, convenientemente a un número primo.*

```
set.seed(3) # Inicializamos la semilla a un número por defecto
```

Ejercicio sobre la Complejidad de H y el Ruido

Ejercicio 1

Dibujar una gráfica con la nube de puntos de salida correspondiente:

a) Considere $N = 50$**,** $dim = 2$**,** $rango = [-50, +50]$ **con** $simula\_unif(N, dim, rango)$**.**

Para generar la gráfica de nube de puntos, partimos de la función $simula\_unif()$ desarrollada por la profesora. En ella, se generan por defecto 2 puntos entre [0,1] de 2 dimensiones.

```
simula_unif = function (N = 2, dim = 2, rango = c(0,1)) {
  m = matrix(runif(N*dim, min=rango[1], max=rango[2]), nrow=N, ncol=dim, byrow=T)
  m
}
```

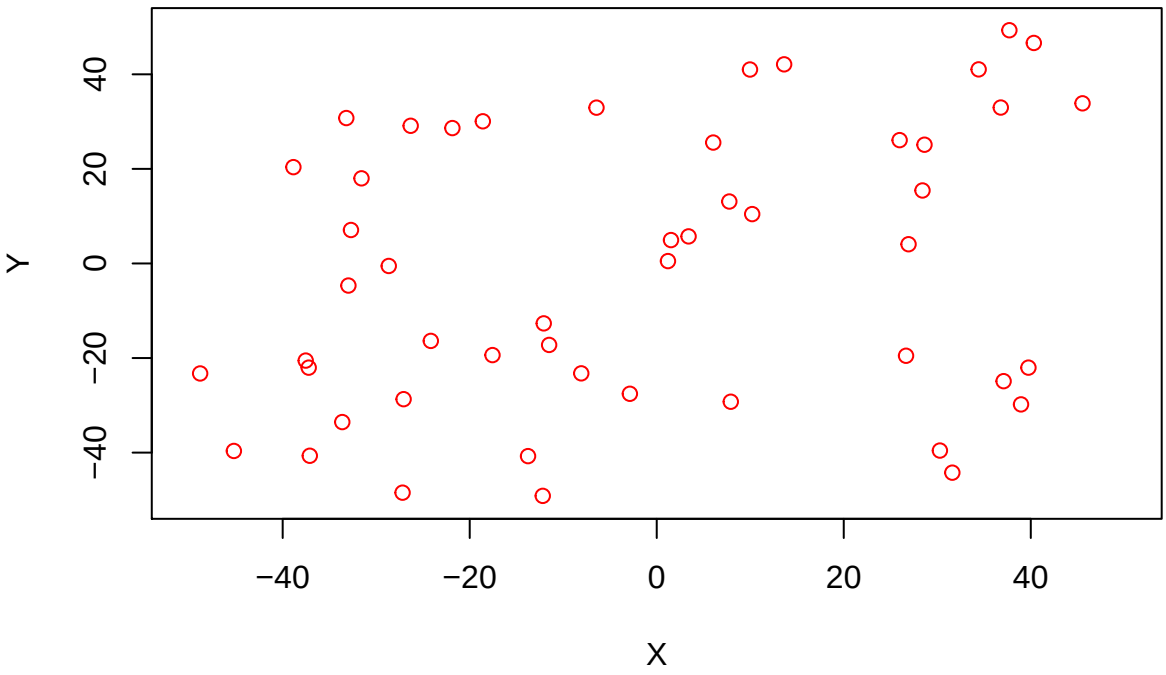
El apartado nos pide crear una gráfica, para una lista de 50 vectores de dimensión 2, donde cada vector contiene 2 números aleatorios uniformes en el intervalo $[-50, +50]$. Para pintarlo, hacemos lo siguiente:

1. Obtenemos los puntos de manera aleatoria:

```
# Cuando se utilizan números aleatorios, es recomedable establecer una semilla, para
# que al repetir la generación de los números, siempre obtengamos los mismos
set.seed(3)

# Generamos nuestros números aleatorios
unif <- simula_unif(N = 50, dim = 2, rango = c(-50, 50))
```

2. Representamos gráficamente los puntos con $plot()$:

```
# Dibujamos los puntos
plot (unif, main = "Distribución Uniforme", col = 2, xlab = "X", ylab = "Y",
      xlim = c(-50, 50), ylim = c(-50, 50))
```



En la gráfica, estamos representando los puntos que hemos obtenido de manera aleatoria, usando para ello las especificaciones citadas en el apartado. Por tanto, la distribución uniforme es una distribución de probabilidad cuyos valores comprendidos entre dos extremos, tienen la misma probabilidad.

```
# Después de crear una gráfica o iniciar un apartado paramos la ejecución 3 segundos
Sys.sleep(3)
```

b) Considere $N = 50$**,** $dim = 2$ **y** $sigma = [5, 7]$ **con** $simula\_gaus(N, dim, sigma)$**.**

Para generar la gráfica de nube de puntos, partimos de la función $simula\_gaus()$ desarrollada por la profesora. En ella, se genera un conjunto de longitud N de vectores de dimensión dim, conteniendo números aleatorios gaussianos de media 0 y varianzas, dadas por el vector $sigma$. Por defecto, genera 2 puntos de 2 dimensiones.

```
simula_gaus = function(N = 2, dim = 2, sigma) {
  if (missing(sigma))
    stop("Debe dar un vector de varianzas")

  # Para la generación se usa sd, y no la varianza
  sigma = sqrt(sigma)

  if(dim != length(sigma))
    stop ("El numero de varianzas es distinto de la dimensión")

  # Genera 1 muestra, con las desviaciones especificadas
  simula_gauss1 = function() rnorm(dim, sd = sigma)

  # Repite N veces, simula_gauss1 y se hace la traspuesta
```

```
  m = t(replicate(N,simula_gauss1()))
  m
}
```

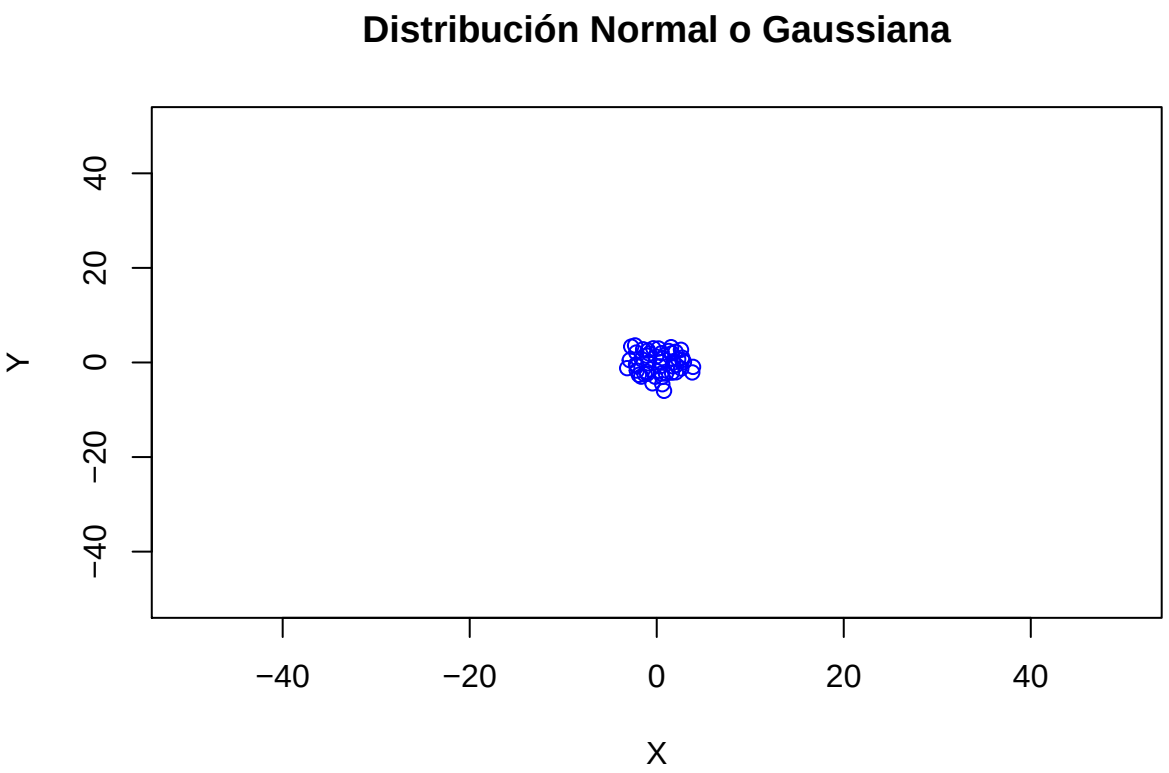
En ete apartado, se nos pide crear una gráfica para una lista de longitud con 50 vectores de dimensión 2, donde cada posición del vector contiene un número aleatorio extraído de una distribución Gaussiana de media 0 y varianza, para cada dimensión, dadas por el vector $sigma = [5, 7]$. Para ello, hacemos el mismo procedimiento que en el apartado anterior:

1. Obtenemos los puntos aleatorios:

```
# Cuando se utilizan números aleatorios, es recomedable establecer una semilla, para
# que al repetir la generación de los números, siempre obtengamos los mismos
set.seed(3)

# Generamos nuestros números aleatorios
gaus = simula_gaus(N = 50, dim = 2, sigma = c(5, 7))
```

2. Representamos gráficamente los puntos con *plot*()

```
# Dibujamos los puntos
plot (gaus, main = "Distribución Normal o Gaussiana", col = 4, xlab = "X", ylab = "Y",
      xlim = c(-50, 50), ylim = c(-50, 50))
```



En la gráfica, hemos representado los puntos de una distribución gaussiana, es decir, realmente lo que hacemos es muestrear dos gaussianas distintas, ambas com media 0 y varianzas 5 y 7. Como se aprecia, todos los puntos están concentrados en el intervalo $(0, 0)$ (la gaussiana se concentra en el centro).

```
# Después de crear una gráfica o iniciar un apartado paramos la ejecución 3 segundos
Sys.sleep(3)
```

Ejercicio 2

Con ayuda de la función $simula\_unif()$ generar una muestra de puntos $2D$ a los que vamos añadir una etiqueta usando el signo de la función $f(x, y) = y - ax - b$, es decir, el signo de la distancia de cada punto a la recta simulada con $simula\_recta()$.

Antes de nada, partimos de la función desarrollada por la profesora para la generación de la recta. La función $simula\_recta(intervalo)$ calcula los parámetros de una recta aleatoria, $y = ax + b$, que corta al plano $[-50, 50]x[-50, 50]$

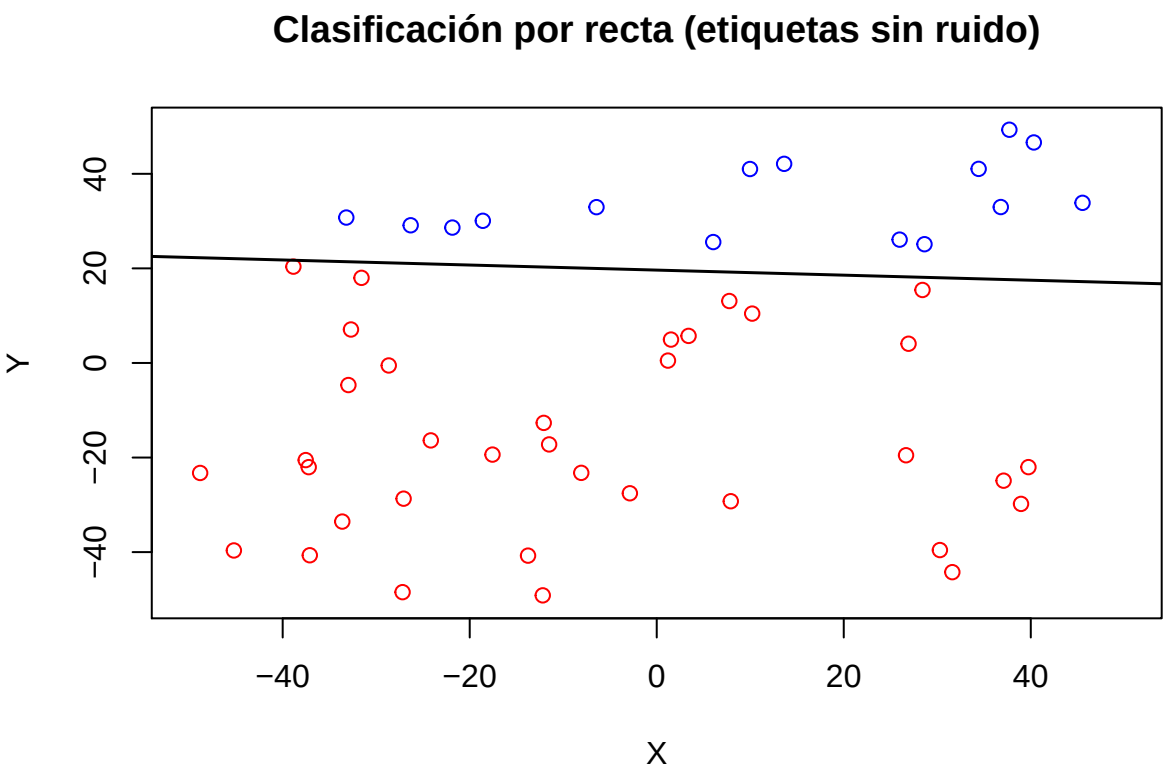
Nota: *Para calcular la recta se simulan las coordenadas de 2 puntos dentro del cuadrado y se calcula la recta que pasa por ellos.*

```
simula_recta = function (intervalo = c(-1,1), visible=F){
  # Se generan 2 puntos
  ptos = simula_unif(2,2,intervalo)

  # calculamos la pendiente
  a = (ptos[1,2] - ptos[2,2]) / (ptos[1,1]-ptos[2,1])

  # Calculamos el punto de corte
  b = ptos[1,2]-a*ptos[1,1]

  # Se pinta la recta y los 2 puntos
  if (visible) {

    # Si no esta abierto el dispositivo lo abre con plot
    if (dev.cur()==1)
      plot(1, type="n", xlim=intervalo, ylim=intervalo)

    # Pinta en verde los puntos
    points(ptos,col=3)

    # y la recta
    abline(b,a,col=3)
  }

  # Devuelve el par pendiente y punto de corte
  c(a,b)
}
```

a) Dibujar una gráfica donde los puntos muestren el resultado de su etiqueta, junto con la recta usada para ello. (Observe que todos los puntos están bien clasificados respecto de la recta)

Primero, generamos un conjunto de números aleatorios, luego, generamos una recta. Y para etiquetar los puntos según la recta, tenemos que aplicar a todos los puntos la diferencia de estos con respecto a la recta y quedarnos con el signo. Por tanto, aplicamos la función $f(x, y) = y - ax - b$ a cada punto y almacenamos el signo del resultado en una lista. Por último, representaremos la gráfica con $plot()$.

```r
# Cuando se utilizan números aleatorios, es recomedable establecer una semilla, para
# que al repetir la generación de los números, siempre obtengamos los mismos
set.seed(3)

# Generamos aleatoriamente una lista de números aleatorios
datos01 = simula_unif(N=50, dim=2, rango=c(-50,50))

# Generamos una recta
recta01 = simula_recta(intervalo=c(-50,50))

# Aplicamos la función a cada punto y almacenamos el signo del resultado en una lista
# con esto acabamos de etiquetar cada punto
etiquetas01 = sign(datos01[,2] - recta01[1]*datos01[,1] - recta01[2])

# Representamos los puntos
plot (datos01, main = "Clasificación por recta (etiquetas sin ruido)",
      col = etiquetas01+3, xlab = "X", ylab = "Y", xlim = c(-50,50), ylim = c(-50,50))

# Representamos la recta (parámetros al revés en abline)
abline(recta01[2], recta01[1], lwd = 1.5)
```



Como se observa en la gráfica, al ser los puntos linealmente separables, siempre existe una recta que los clasifica correctamente. Para encontrar que puntos están por encima y que puntos están por debajo, hemos aplicado la función a cada punto y almacenado su signo.

Los puntos que están por encima de la recta (puntos azules), tienen etiqueta 1 y los puntos que están por debajo de la recta (puntos rojos), tienen etiqueta -1.

```r
# Después de crear una gráfica o iniciar un apartado paramos la ejecución 3 segundos
Sys.sleep(3)
```

b) Modifique de forma aleatoria un 10% de etiquetas positivas y otro 10% de negativas. Dibuje de nuevo la gráfica anterior. (Ahora hay puntos mal clasificados respecto de la recta)

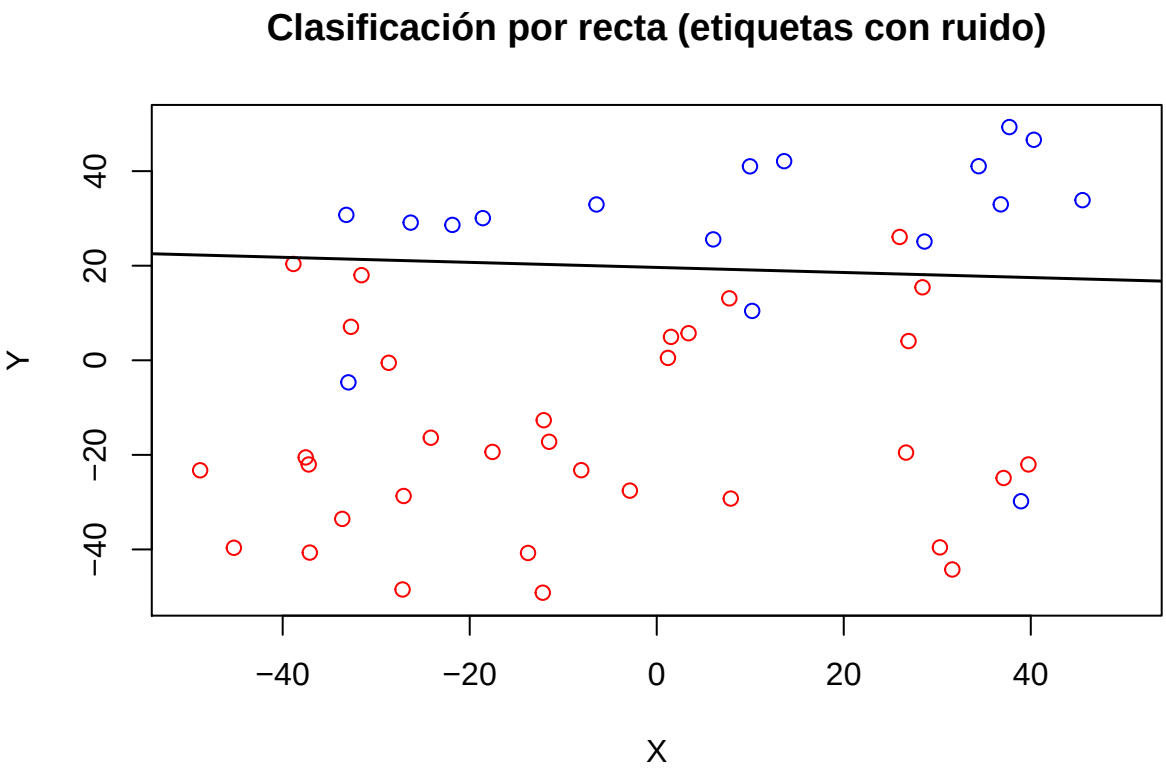
Para ello nos creamos una función que introducirá ruido en las etiquetas. Básicamente, contaremos el número de etiquetas positivas y negativas que hay, y de cada parte calcularemos el porcentaje de etiquetas a cambiar a la otra clase. Las etiquetas que se modifiquen lo harán de manera aleatoria.

```r
generar_ruido = function (etiqueta, porcentaje=10) {
  # Obtenemos el tamaño de las etiquetas
  et = length(etiqueta)

  # Obtenemos el tamaño de las etiqueas con valor 1
  et1 = sum(etiqueta == 1)

  # Obtenemos el tamaño de las etiqueas con valor -1
  et2 = et - et1

  # Obtenemos el indice de cada etiqueta
  ind1 = which(etiqueta == 1)
  ind2 = which(etiqueta == -1)

  # Introducimos el ruido en las etiquetas de manera aleatoria
  # dependiendo del porcentaje de ruido que queramos meter
  etiqueta[sample(ind1, et1*porcentaje/100)] = -1
  etiqueta[sample(ind2, et2*porcentaje/100)] = 1

  etiqueta
}
```

Una vez, realizada la función $generar\_ruido(etiqueta, porcentaje)$, realizamos el mismo proceso que en el apartado a.

```r
# Cuando se utilizan números aleatorios, es recomedable establecer una semilla, para
# que al repetir la generación de los números, siempre obtengamos los mismos
set.seed(3)

# Generamos aleatoriamente una lista de números aleatorios
datos02 = simula_unif(N=50, dim=2, rango=c(-50,50))

# Generamos una recta
recta02 = simula_recta(intervalo=c(-50,50))

# Aplicamos la función a cada punto y almacenamos el signo del resultado en una lista
etiquetas02 = sign(datos02[,2] - recta02[1]*datos02[,1] - recta02[2])

# Introducimos el ruido en las etiquetas
etiquetas_ruido01 = generar_ruido(etiquetas02, 10)

# Representamos los puntos
```

```r
plot (datos02, main = "Clasificación por recta (etiquetas con ruido)",col = etiquetas_ruido01+3,
      xlab = "X", ylab = "Y", xlim = c(-50,50), ylim = c(-50,50))

# Representamos la recta (parámetros al revés en abline)
abline(recta02[2], recta02[1], lwd = 1.5)
```



Como se ve en la gráfica, existen etiquetas mal clasificadas, ya que se ha introducido ruido en ellas. Por consiguiente, las etiquetas no se pueden separarar linealmente, es decir, no existe una recta que separe las etiquetas con valor -1 de las etiquetas con valor 1.

```r
# Después de crear una gráfica o iniciar un apartado paramos la ejecución 3 segundos
Sys.sleep(3)
```

Como se ve, en las etiquetas positivas, es decir, en las que están por encima de la recta, solo se introduce ruido en una de ellas. Sin embargo, en las etiquetas negativas se introduce ruido en tres de ellas, esto es debido a la cantidad de puntos que tiene cada clase. Para comprobar que se ha introducido el ruido correctamente, hacemos lo siguiente:

1. Calculamos el número de etiquetas de cada parte:

```r
# Calcular etiquetas positivas
sum(etiquetas02 == 1)
```

```
## [1] 15
```

```r
# Calcular etiquetas negativas
sum(etiquetas02 == -1)
```

```
## [1] 35
```

2. Como se nos pide cambiar de forma aleatoria un 10% de etiquetas positivas y un 10% de etiquetas negativas. Calculamos el porcentaje que debemos cambiar en cada clase:

```
# Porcentaje a introducir ruido en las etiquetas positivas
(sum(etiquetas02 == 1)) / 10
```

```
## [1] 1.5
```

```
# Porcentaje a introducir ruido en las etiquetas negativas
(sum(etiquetas02 == -1)) / 10
```

```
## [1] 3.5
```

Finalmente, modificamos un 1.5% de las etiquetas positivas, lo que quiere decir que se introduce una etiqueta con ruido. Mientras que en las etiquetas negativas modificamos un 3.5% de ellas, es decir, introducimos tres etiquetas con ruido.

```
# Después de crear una gráfica o iniciar un apartado paramos la ejecución 3 segundos
Sys.sleep(3)
```

Ejercicio 3

Supongamos ahora que las siguientes funciones definen la frontera de clasificación de los puntos de la muestra en lugar de una recta:

- $f(x, y) = (x - 10)^2 + (y - 20)^2 - 400$
- $f(x, y) = 0.5(x + 10)^2 + (y - 20)^2 - 400$
- $f(x, y) = 0.5(x - 10)^2 - (y + 20)^2 - 400$
- $f(x, y) = y - 20x^2 - 5x + 3$

Visualizar el etiquetado generado en 2b junto con cada una de las gráficas de cada una de las funciones. Comparar las formas de las regiones positivas y negativas de estas nuevas funciones con las obtenidas en el caso de la recta. ¿Hemos ganado algo en mejora de clasificación al usar funciones más complejas que la dada por una función lineal? Explicar el razonamiento.

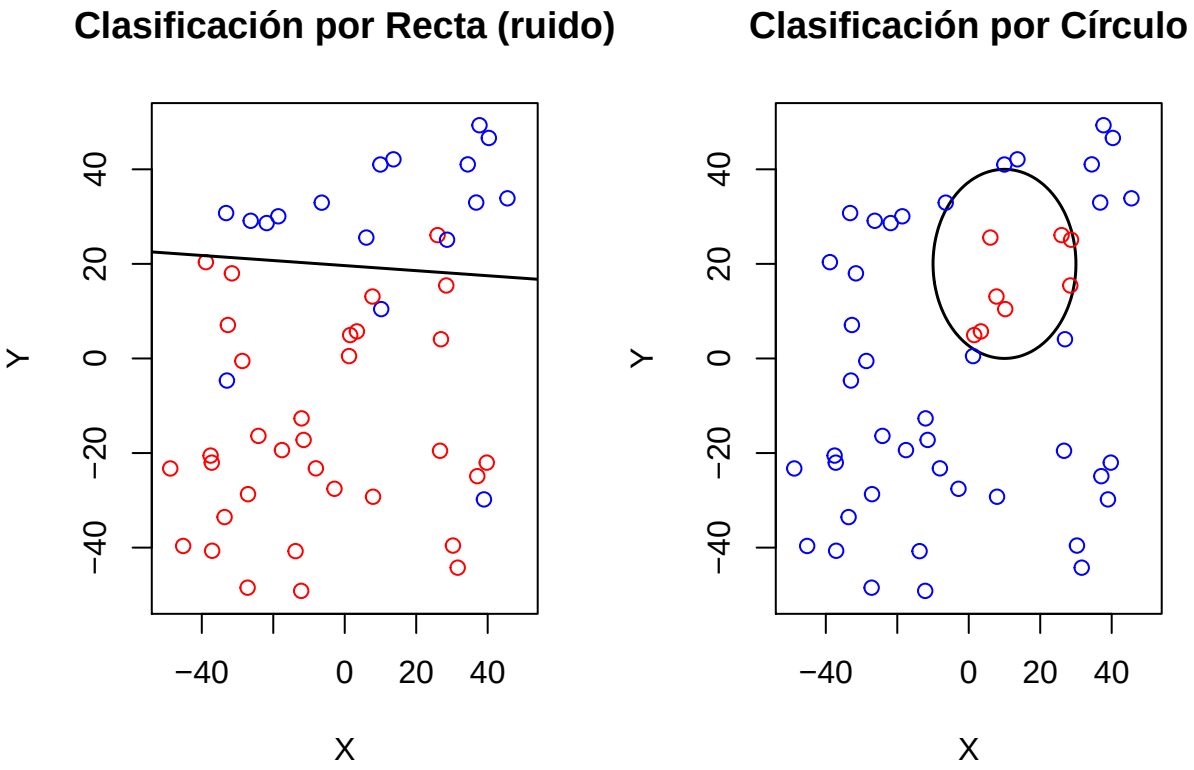
Para comenzar vamos a usar los mismos datos que en el ejercicio anterior, usando la variable *datos*02. Luego, cogemos la función *pintar_frontera*() desarrollada por la profesora, que pinta la frontera de la función que deseamos representar.

```
pintar_frontera = function(f,rango=c(-50,50)) {
  x = y = seq(rango[1],rango[2],length.out = 100)
  z = outer(x,y,FUN=f)

  # Si no está abierto el dispositivo lo abre con plot
  if (dev.cur()==1)
   plot(1, type="n", xlim=rango, ylim=rango)

  contour(x,y,z, levels = 0, drawlabels = FALSE, add = T,
          xlim = rango, ylim = rango, xlab = "x", ylab = "y", lwd = 1.5)
}
```

Básicamente lo que haremos será:
1. Crear una función para cada función dada por el ejercicio.
2. Visualizar la gráfica del etiquetado del ejercicio 2b, es decir, ver la clasificación de la recta con etiquetas de ruido.
3. Pintar la frontera definida en la función del punto 1.
4. Obtener los puntos de fuera y de dentro de la frontera del punto 1.

- $f(x,y) = (x-10)^2 + (y-20)^2 - 400$

```
# Nos creamos una función que definirá la frontera de clasificación
# de los puntos de la muestra
f1_xy <- function(x,y) {
  return ((x-10)^2 + (y-20)^2 - 400)
}

# Dividimos la región de dibujo en dos partes
par(mfrow=c(1:2))

# Visualizar el etiquetado generado en el ejercicio 2b (etiquetas con ruido)
# ----------------------------------------------------------------------------
# Representamos los puntos
plot (datos02, main = "Clasificación por Recta (ruido)",col = etiquetas_ruido01+3,
      xlab = "X", ylab = "Y", xlim = c(-50,50), ylim = c(-50,50))

# Representamos la recta (parámetros al revés en abline)
abline(recta02[2], recta02[1], lwd = 1.5)
# ----------------------------------------------------------------------------

# Abrimos otro dispositivo con plot, para que no nos cree ambas gráficas juntas
plot(1, main = "Clasificación por Círculo", type="n", xlab = "X", ylab = "Y",
     xlim=c(-50,50), ylim=c(-50,50))

# Pintamos la frontera que define la primera función
pintar_frontera(f1_xy)

# Obtenemos los puntos que van a estar fuera de la frontera
etiquetado01 <- subset(datos02, f1_xy(datos02[,1],datos02[,2]) > 0)

# Obtenemos los puntos que van a estar dentro de la frontera
etiquetado02 <- subset(datos02, f1_xy(datos02[,1],datos02[,2]) <= 0)

# Pintar los puntos en la segunda gráfica
points(etiquetado01, col = "blue")
points(etiquetado02, col = "red")
```



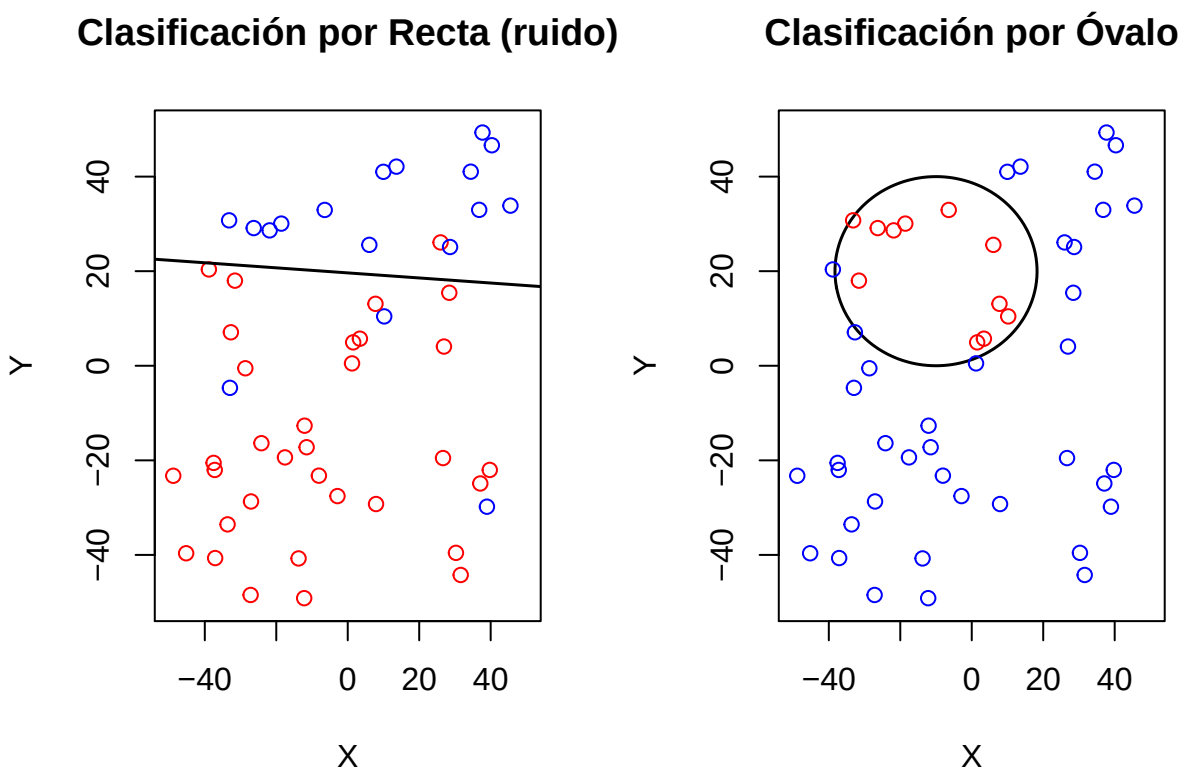
Como se puede ver en la primera gráfica, comentada en el *ejercicio 2 apartado b*, no existe una recta que separe linealmente los datos. En concreto, en ninguna gráfica existe una recta que los separe linealmente. Sin embargo, en la segunda gráfica a expensas de no existir una recta, sí existe un círculo que separa los datos.

```
# Después de crear una gráfica o iniciar un apartado paramos la ejecución 3 segundos
Sys.sleep(3)
```

- $f(x, y) = 0.5(x + 10)^2 + (y - 20)^2 - 400$

```
# Nos creamos una función que definirá la frontera de clasificación
# de los puntos de la muestra
f2_xy <- function(x,y) {
  (0.5*(x+10)^2 + (y-20)^2 - 400)
}

# Dividimos la región de dibujo en dos partes
par(mfrow=c(1:2))

# Visualizar el etiquetado generado en el ejercicio 2b (etiquetas con ruido)
# ------------------------------------------------------------------------------
# Representamos los puntos
plot (datos02, main = "Clasificación por Recta (ruido)",col = etiquetas_ruido01+3,
      xlab = "X", ylab = "Y", xlim = c(-50,50), ylim = c(-50,50))

# Representamos la recta (parámetros al revés en abline)
abline(recta02[2], recta02[1], lwd = 1.5)
# ------------------------------------------------------------------------------

# Abrimos otro dispositivo con plot, para que no nos cree ambas gráficas juntas
plot(1, main = "Clasificación por Óvalo", type="n", xlab = "X", ylab = "Y",
```

```
     xlim=c(-50,50), ylim=c(-50,50))

# Pintamos la frontera que define la primera función
pintar_frontera(f2_xy)

# Obtenemos los puntos que van a estar fuera de la frontera
etiquetado03 <- subset(datos02, f2_xy(datos02[,1],datos02[,2]) > 0)

# Obtenemos los puntos que van a estar dentro de la frontera
etiquetado04 <- subset(datos02, f2_xy(datos02[,1],datos02[,2]) <= 0)

# Pintar los puntos en la segunda gráfica
points(etiquetado03, col = "blue")
points(etiquetado04, col = "red")
```



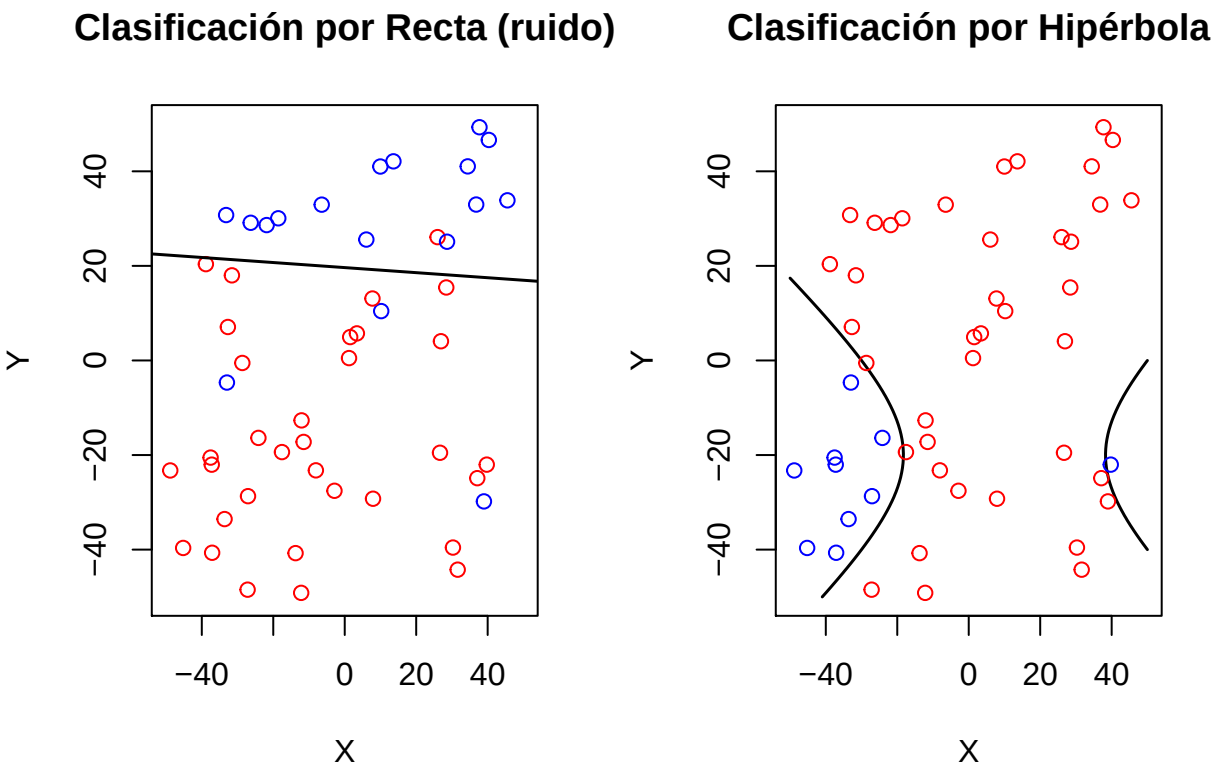
Como se puede ver en la primera gráfica, comentada en el *ejercicio 2 apartado b*, no existe una recta que separe linealmente los datos. En concreto, en ninguna gráfica existe una recta que los separe linealmente. Sin embargo, en la segunda gráfica a expensas de no existir una recta, sí existe un óvalo que separa los datos, obteniendo así una buena clasificación.

```
# Después de crear una gráfica o iniciar un apartado paramos la ejecución 3 segundos
Sys.sleep(3)
```

- $f(x, y) = 0.5(x - 10)^2 - (y + 20)^2 - 400$

```
# Nos creamos una función que definirá la frontera de clasificación
# de los puntos de la muestra
f3_xy <- function(x,y) {
   (0.5*(x-10)^2 - (y+20)^2 - 400)
}
```

```r
# Dividimos la región de dibujo en dos partes
par(mfrow=c(1:2))

# Visualizar el etiquetado generado en el ejercicio 2b (etiquetas con ruido)
# -----------------------------------------------------------------------------
# Representamos los puntos
plot (datos02, main = "Clasificación por Recta (ruido)",col = etiquetas_ruido01+3,
      xlab = "X", ylab = "Y", xlim = c(-50,50), ylim = c(-50,50))

# Representamos la recta (parámetros al revés en abline)
abline(recta02[2], recta02[1], lwd = 1.5)
# -----------------------------------------------------------------------------

# Abrimos otro dispositivo con plot, para que no nos cree ambas gráficas juntas
plot(1, main = "Clasificación por Hipérbola", type="n", xlab = "X", ylab = "Y",
     xlim=c(-50,50), ylim=c(-50,50))

# Pintamos la frontera que define la primera función
pintar_frontera(f3_xy)

# Obtenemos los puntos que van a estar fuera de la frontera
etiquetado05 <- subset(datos02, f3_xy(datos02[,1],datos02[,2]) > 0)

# Obtenemos los puntos que van a estar dentro de la frontera
etiquetado06 <- subset(datos02, f3_xy(datos02[,1],datos02[,2]) <= 0)

# Pintar los puntos en la segunda gráfica
points(etiquetado05, col = "blue")
points(etiquetado06, col = "red")
```



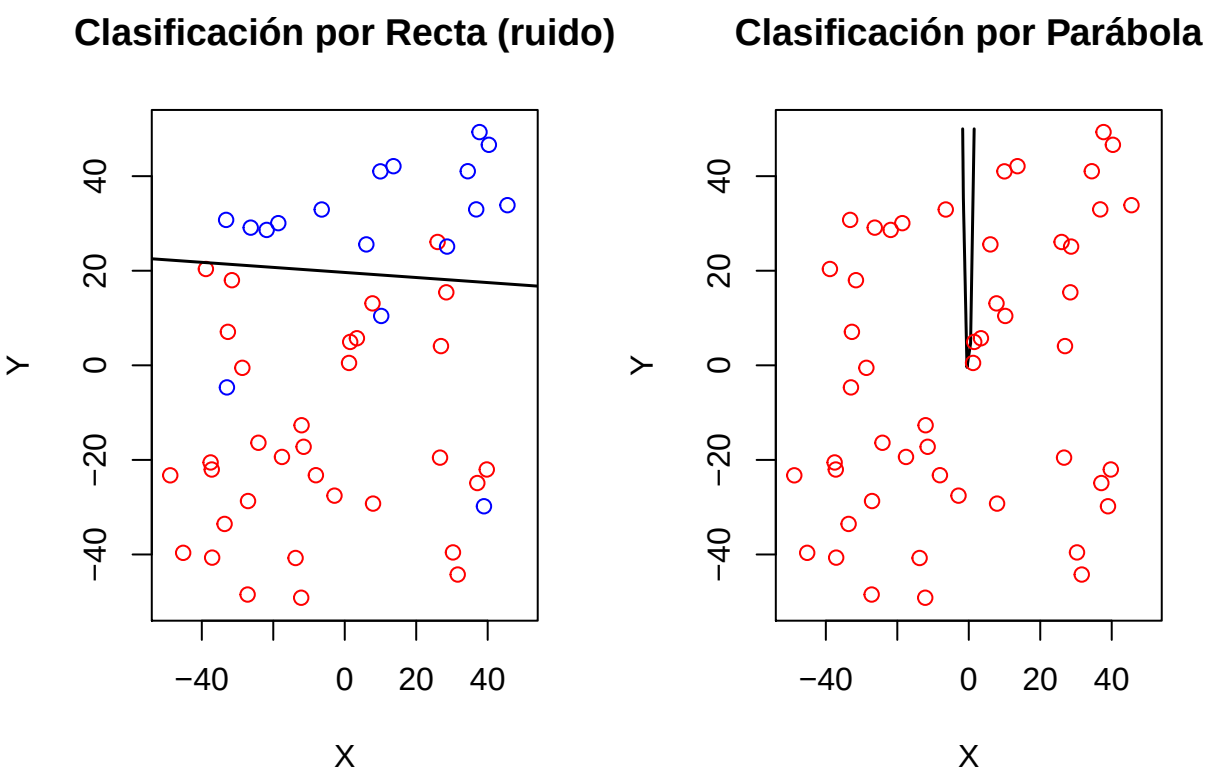
Como se puede ver en la primera gráfica, comentada en el *ejercicio 2 apartado b*, no existe una recta que separe linealmente los datos. En concreto, en ninguna gráfica existe una recta que los separe linealmente. Sin embargo, en la segunda gráfica a expensas de no existir una recta, sí existe una función hipérbola que separa los datos.

```
# Después de crear una gráfica o iniciar un apartado paramos la ejecución 3 segundos
Sys.sleep(3)
```

- $f(x, y) = y - 20x^2 - 5x + 3$

```
# Nos creamos una función que definirá la frontera de clasificación
# de los puntos de la muestra
f4_xy <- function(x,y) {
  (y - 20*x^2 - 5*x + 3)
}

# Dividimos la región de dibujo en dos partes
par(mfrow=c(1:2))

# Visualizar el etiquetado generado en el ejercicio 2b (etiquetas con ruido)
# ---------------------------------------------------------------------------------
# Representamos los puntos
plot (datos02, main = "Clasificación por Recta (ruido)",col = etiquetas_ruido01+3,
      xlab = "X", ylab = "Y", xlim = c(-50,50), ylim = c(-50,50))

# Representamos la recta (parámetros al revés en abline)
abline(recta02[2], recta02[1], lwd = 1.5)
# ---------------------------------------------------------------------------------

# Abrimos otro dispositivo con plot, para que no nos cree ambas gráficas juntas
```

```r
plot(1, main = "Clasificación por Parábola", type="n", xlab = "X", ylab = "Y",
     xlim=c(-50,50), ylim=c(-50,50))

# Pintamos la frontera que define la primera función
pintar_frontera(f4_xy)

# Obtenemos los puntos que van a estar fuera de la frontera
etiquetado07 <- subset(datos02, f4_xy(datos02[,1],datos02[,2]) > 0)

# Obtenemos los puntos que van a estar dentro de la frontera
etiquetado08 <- subset(datos02, f4_xy(datos02[,1],datos02[,2]) <= 0)

# Pintar los puntos en la segunda gráfica
points(etiquetado07, col = "blue")
points(etiquetado08, col = "red")
```



Como se puede ver en la primera gráfica, comentada en el *ejercicio 2 apartado b*, no existe una recta que separe linealmente los datos. Aunque, en la segunda gráfica, si hubiésemos creado un separador lineal, hubiéramos obtenido un ajuste de calidad, ya que siempre va a ver una recta por encima de esos valores, que nos haga clasificar bien.

En nuestro caso, al haber generado los datos de manera aleatoria y haber establecido una semilla para defecto, no se va a encontrar ningún punto dentro de la región definida por la función de la parábola. Pero puede que con otra generación de números aleatorios, encuentre algún punto dentro de la frontera, con lo cual los puntos no se podrían separar linealmente.

Entonces, en las tres primeras funciones no vamos a encontrar una recta que separe linealmente los datos, por lo que el PLA no funcionará correctamente.

```
# Después de crear una gráfica o iniciar un apartado paramos la ejecución 3 segundos
Sys.sleep(3)
```

Ejercicio sobre el Algoritmo Perceptron

Ejercicio 1

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

***Nota**: Para realizar este ejercicio, he usado la documentación subida a DECSAI. Y en particular la sesión 1 de teoría y el libro Learning from Data: A short course de Abu Mostafa, Magdon Ismail, Lin.*

Vamos a escribir una función que implemente el algoritmo Perceptron o PLA. Primero, escribiré el pseudocódigo (*Sesión 1, transparencia 33*).

Entrada: Los *datos*, las *etiquetas*, el *valor inicial del vector* y el *número de iteraciones máximas*, ya que el PLA solo va a parar si el conjunto de datos es linealmente separable.

Salida: Los *parámetros del hiperplano* y el *número de iteraciones* que tarda en clasificar correctamente.

1. Fijar $vini < -0$
2. Elegir una muestra que esté mal clasificada aleatoriamente x_i.
3. Iterar sobre las muestras D, mejorando la solución.
4. Para cada $x_i \in D$

- Si el signo de $w^T * x$ es distinto a y_i
- Actualizar w: $w_{new} = w_{old} + x_i * y_i$

5. Paramos cuando estén linealmente separables o hayamos consumido el *número de iteraciones máximas*, dadas al programa a la entrada.
6. Devolver los *parámetros del hiperplano del PLA* y el *número de iteraciones*.

A la hora de escoger los índices, los cogeré de manera aleatoria, ya que no se especifican en las transparencias de la sesión 1. Y además, cuando luego hagamos la media, nos beneficiará que lo hagamos así.

Los parámetros del hiperplano a y b son: $-(\frac{w_2}{w_3})$ y $-(\frac{w_1}{w_3})$.

Para calcularlos, partimos de los pesos $w = (term_{independiente}, term_x, term_y) = (1, x, y)$ y de la función de la recta $f(x, y) = y - ax - b$ o que es lo mismo que $0 = ax - y + b$, siendo b el término independiente y a y b las incógnitas.

Ahora obtenemos los términos de la recta y los ponemos como si fueran un vector $(ax, -y, b)$, pero como vamos a iguarlo a los pesos, escribimos el mismo orden en el que están los pesos, por tanto en la recta quedaría $(b, ax, -y)$. Luego lo igualamos a los pesos: $(b, ax, -y) = w$, con lo que tendríamos $w_1 + w_2 * x - w_3 * y = 0$.

***Nota**: Despreciamos en la fórmula a y b al ser las incógnitas a sacar.*

Entonces, despejando de la anterior fórmula la y quedaría

$$y = (\frac{w_2}{-w_3}) * x + (\frac{w_1}{-w_3})$$

Y obtendríamos que $a = -(\frac{w_2}{w_3})$ y que $b = -(\frac{w_1}{w_3})$.

```
ajusta_PLA <- function(datos, label, max_iter, vini) {

  # Asigamos a w el valor inicial del vector
  w <- vini

  # Variable que contará el número de iteraciones para parar el PLA
  # en caso de que no sea linealmente separable
  iteraciones = 1

  # Variable booleana que nos permitirá seguir o no,
  # viendo cuando se ha cambiado el valor o no
  continuar = T

  # Añadimos a la matriz de datos, una tercera columna para poder hacer
  # el producto
  datos = cbind(rep(1, nrow(datos)), datos)

  # Iterar en las muestra, mejorando la solución
  while (iteraciones <= max_iter && continuar) {
    continuar = FALSE

    # Hacemos un bucle interno iterando sobre cada dato
    # cogiendo para ello los índices aleatoriamente
    for(i in (sample(nrow(datos)))) {
      # Obtenemos el signo del producto vectorial de los datos por los pesos
      signo = sign(datos[i,] %*% w)

      # Comparamos si el signo obtenido antes, coincide con el de la etiqueta
      if (signo != label[i]) { # Sino coincide
        # Actualizamos el peso: wnew = wold + x*etiqueta
        w = w + datos[i,]*label[i]

        # Establecemos que se ha cambiado el valor
        continuar = TRUE

        # Incrementamos el número de iteraciones
        iteraciones = iteraciones + 1
      }
    }
  }

  # Devolvemos los parámetros del hiperplano del perceptrón
  # y el numero de iteraciones
  c(-w[1]/w[3], -w[2]/w[3], iteraciones)
}
```

Un ejemplo de ejecución del **algoritmo PLA** se ve en el ejercicio siguiente.

Ejercicio 2

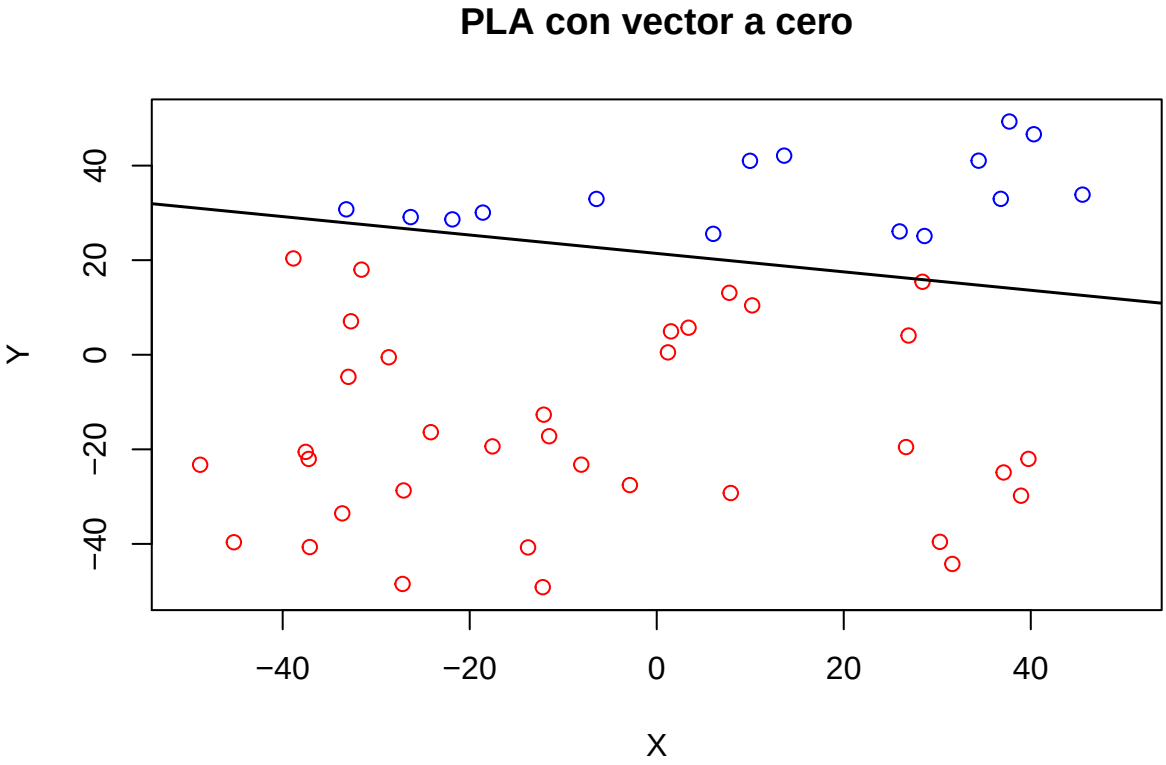
Ejecutar el algoritmo PLA con los datos simulados en los apartados 2a de la sección 1. Inicializar el algoritmo con: a) el vector cero y, b) con vectores de números aleatorios en [0, 1] (10 veces). Anotar el número medio de iteraciones necesarias en ambos para converger. Valorar el resultado relacionando el punto de inicio con el número de iteraciones.

Usaremos los datos del ejercicio 2a de la sección 1, por lo que nuestros datos a usar serán los *datos*01 y las etiquetas *etiquetas*01.

a) El vector a cero.

Inicializaremos los pesos del perceptrón a 0.

```
# Guardamos los valores del PLA en una variable
# Le pasamos al perceptrón un máximo de 2000 iteraciones y el vector inicial a 0
perceptron01 = (ajusta_PLA(datos01, etiquetas01, 2000, c(0,0,0)))

# Dibujamos los puntos en la gráfica
plot(datos01, col = etiquetas01+3, xlim=c(-50,50), ylim = c(-50,50), xlab = "X",
     ylab = "Y", main="PLA con vector a cero")

# Representamos la recta hecha por el perceptron
abline(a=perceptron01[1], b=perceptron01[2], lwd=1.5)
```



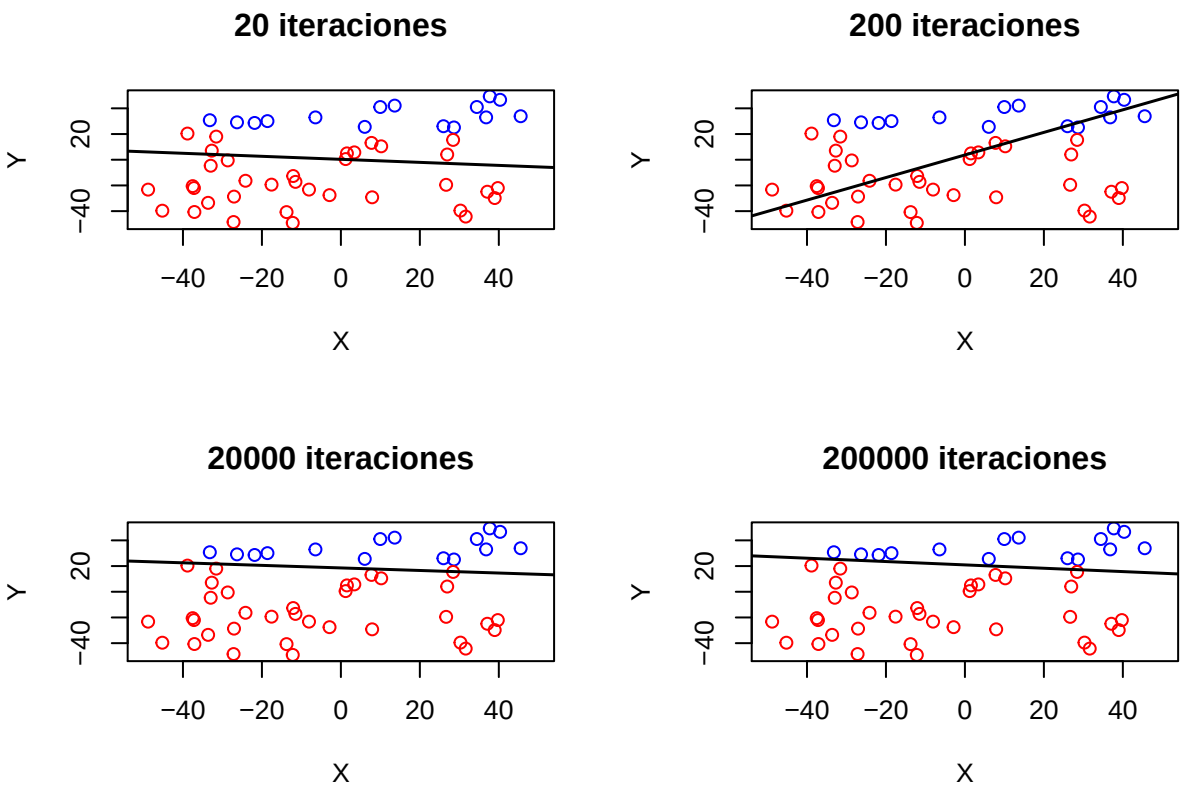

```
# Mostramos el número de iteraciones que hacen falta para converger
cat(perceptron01[3],"iteraciones para converger.")
```

```
## 193 iteraciones para converger.
```

Con los datos obtenidos no podemos sacar una conclusión firme. Para ello vamos a realizar el mismo proceso que antes pero variando el máximo de iteraciones a iterar. En concreto, probaremos con 20, 200, 20000, 200000 iteraciones. Para así, intentar llegar a una conclusión más clara. Recordad, que seguiremos asignando el vector a cero y que escogemos los índices de forma aleatoria.

```
# Después de crear una gráfica o iniciar un apartado paramos la ejecución 3 segundos
Sys.sleep(3)
```

```
# Para obtener las gráficas juntas
par(mfrow=c(2,2))

# Para un máximo de 20 iteraciones
# ------------------------------------------------------------------------------
perceptron03 = (ajusta_PLA(datos01, etiquetas01, 20, c(0,0,0)))
plot(datos01, col = etiquetas01+3, , xlab = "X", ylab = "Y", main="20 iteraciones",
     xlim = c(-50,50), ylim = c(-50,50))
abline(a=perceptron03[1], b=perceptron03[2], lwd=1.5)
# ------------------------------------------------------------------------------

# Para un máximo de 200 iteraciones
# ------------------------------------------------------------------------------
perceptron04 = (ajusta_PLA(datos01, etiquetas01, 200, c(0,0,0)))
plot(datos01, col = etiquetas01+3, , xlab = "X", ylab = "Y", main="200 iteraciones",
     xlim = c(-50,50), ylim = c(-50,50))
abline(a=perceptron04[1], b=perceptron04[2], lwd=1.5)
# Para un máximo de 20000 iteraciones
# ------------------------------------------------------------------------------
perceptron05 = (ajusta_PLA(datos01, etiquetas01, 20000, c(0,0,0)))
plot(datos01, col = etiquetas01+3, , xlab = "X", ylab = "Y", main="20000 iteraciones",
     xlim = c(-50,50), ylim = c(-50,50))
abline(a=perceptron05[1], b=perceptron05[2], lwd=1.5)
# ------------------------------------------------------------------------------

# Para un máximo de 200000 iteraciones
# ------------------------------------------------------------------------------
perceptron06 = (ajusta_PLA(datos01, etiquetas01, 200000, c(0,0,0)))
plot(datos01, col = etiquetas01+3, , xlab = "X", ylab = "Y", main="200000 iteraciones",
     xlim = c(-50,50), ylim = c(-50,50))
abline(a=perceptron06[1], b=perceptron06[2], lwd=1.5)
```




```
cat("Con un máximo de 20 iteraciones, converge a ", perceptron03[3],"iteraciones.")
```

```
## Con un máximo de 20 iteraciones, converge a  26 iteraciones.
```

```
cat("Con un máximo de 200 iteraciones, converge a ", perceptron04[3],"iteraciones.")
```

```
## Con un máximo de 200 iteraciones, converge a  206 iteraciones.
```

```
cat("Con un máximo de 20000 iteraciones, converge a ", perceptron05[3],"iteraciones.")
```

```
## Con un máximo de 20000 iteraciones, converge a  120 iteraciones.
```

```
cat("Con un máximo de 20000 iteraciones, converge a ", perceptron06[3],"iteraciones.")
```

```
## Con un máximo de 20000 iteraciones, converge a  254 iteraciones.
```

Como se aprecia en las gráficas, conforme menos iteraciones tiene para ejecutar el PLA, peor clasifica los datos. Sin embargo, cuanto mayor es el número de iteraciones, mejor ajusta. Esto no quiere decir que siempre sea así, ya que estamos cogiendo los índices aletoriamente y puede que sé de un caso donde los datos sean fáciles, y que con menos iteraciones encuentre la recta que separa los datos linealmente. Pero sí que es verdad, que existe una alta probabilidad de que conforme más iteraciones le demos, mejor ajustará.

Por tanto, si los datos son separables linealmente, sabemos que el PLA encontrará solución, pero no sabremos en que iteración lo hará.

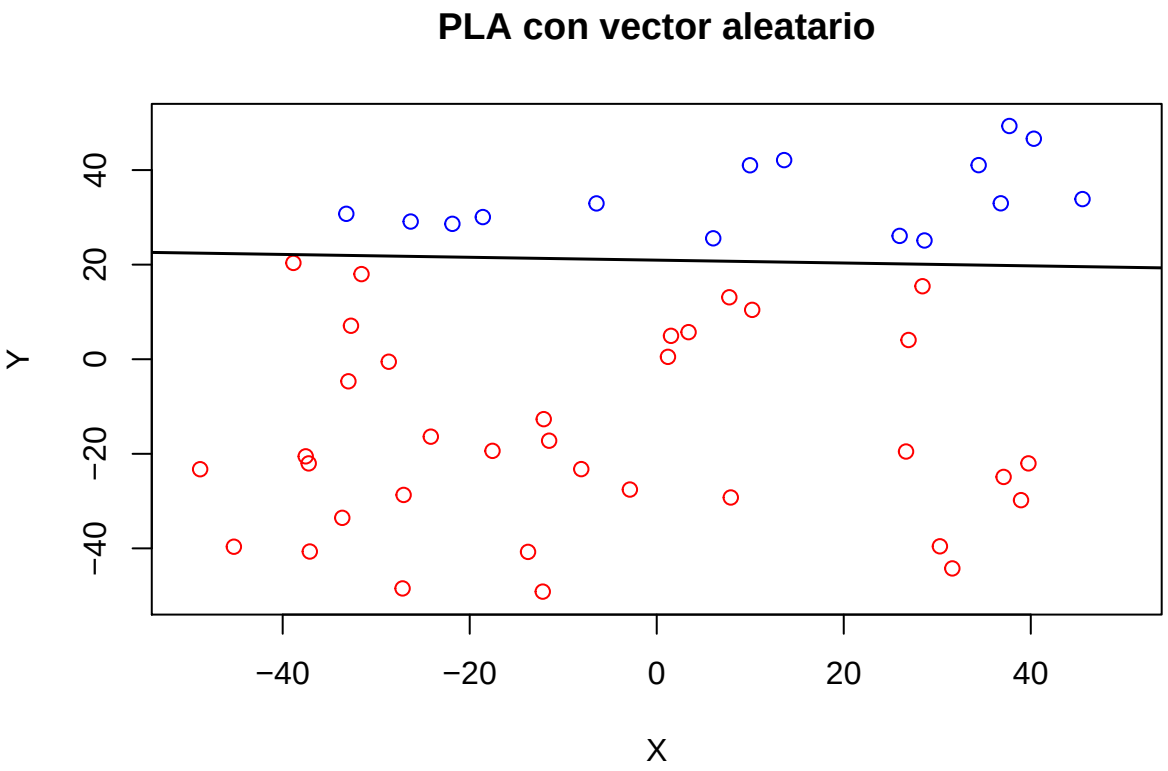
```
# Después de crear una gráfica o iniciar un apartado paramos la ejecución 3 segundos
Sys.sleep(3)
```

b) Con vectores de números aleatorios entre 0 y 1, diez veces.

Inicializaremos los pesos del perceptrón a un valor aleatorio entre 0 y 1.

Primero haremos el experimento, sin ejecutarlo 10 veces y comprobaremos el resultado que obtenemos. Luego lo ejecutaremos 10 veces y comprobaremos. Ambos serán inicializando el vector a números aletarios entre 0 y 1.

```
# Guardamos los valores del PLA en una variable
# Le pasamos al perceptrón un máximo de 2000 iteraciones
# y el vector con valores aleatorios entre [0,1]
perceptron07 = (ajusta_PLA(datos01, etiquetas01, 2000, runif(3)))

# Dibujamos los puntos en la gráfica
plot(datos01, col = etiquetas01+3, xlim=c(-50,50), ylim = c(-50,50), xlab = "X",
     ylab = "Y", main="PLA con vector aleatario")

# Representamos la recta hecha por el perceptron
abline(a=perceptron07[1], b=perceptron07[2], lwd=1.5)
```




```
# Mostramos el número de iteraciones que hacen falta para converger
cat(perceptron07[3],"iteraciones para converger.")
```

```
## 513 iteraciones para converger.
```

Ahora, hacemos el experimento ejecutando 10 veces el PLA, obtiendo para ello la media de las 10 veces. En este caso no mostraré las gráficas, ya que deberíamos ver una gráfica por ajuste, en total serían 10 gráfica. Aunque, como sabemos que converge en una iteración, siempre lo va hacer en las demás, ya que los datos son linealmente separables.

```
# Después de crear una gráfica o iniciar un apartado paramos la ejecución 3 segundos
Sys.sleep(3)
```

```
# Guardamos los valores del PLA en una variable
# Le pasamos al perceptrón un máximo de 2000 iteraciones
```

```r
# y el vector con valores aleatorios entre [0,1]
perceptron08 = replicate(10, ajusta_PLA(datos01, etiquetas01, 2000, runif(3))[3])

# Mostramos el número de iteraciones que hacen falta para converger
# Realizamos la media de las 10 iteraciones
cat(mean(perceptron08),"iteraciones para converger.")
```

```
## 544.7 iteraciones para converger.
```

Como en el anterior apartado, a partir de esto no podemos sacar una conclusión firme. Para ello vamos a realizar el mismo exprimento de antes pero variando el máximo de iteraciones a iterar. En concreto, probaremos con 20, 200, 20000, 200000 iteraciones. Para así, intentar llegar a una conclusión más clara. Recordad, que seguiremos asignando el vector a números aleatorios entre [0,1] y que escogemos los índices de forma aleatoria.

Realizamos el proceso, asignando al vector números aleatorios entre [0,1] y sin repetir 10 veces el PLA. Ya que si converge a la primera, convergerá igual iterando sobre el PLA 10 veces.

```r
# Para obtener las gráficas juntas
par(mfrow=c(2,2))

# Para un máximo de 20 iteraciones
# -----------------------------------------------------------------------------
perceptron09 = (ajusta_PLA(datos01, etiquetas01, 20, runif(3)))
plot(datos01, col = etiquetas01+3, xlim=c(-50,50), ylim = c(-50,50), xlab = "X", ylab = "Y",
     main="20 iteraciones")
abline(a=perceptron09[1], b=perceptron09[2], lwd=1.5)
cat("Con un máximo de 20 iteraciones, converge a ", perceptron09[3],"iteraciones.")
```

```
## Con un máximo de 20 iteraciones, converge a  23 iteraciones.
```

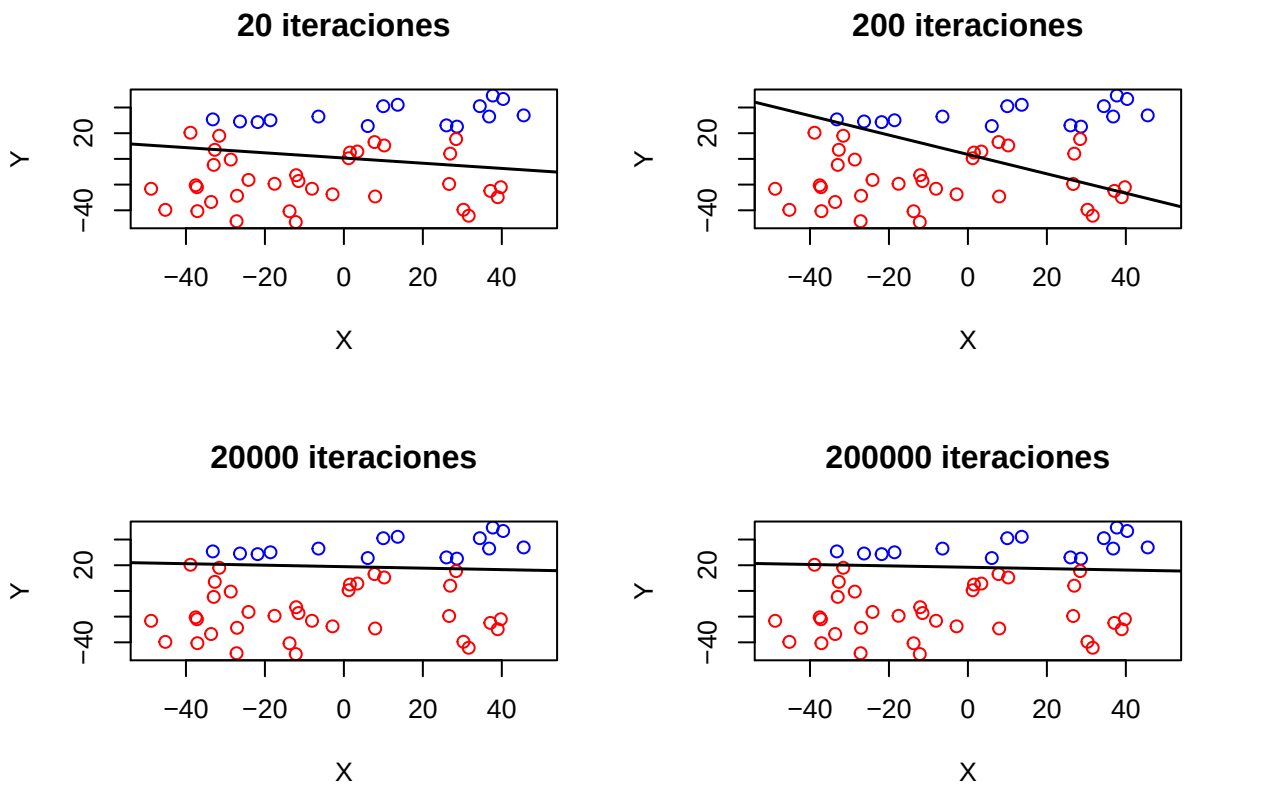
```r
# Para un máximo de 200 iteraciones
# -----------------------------------------------------------------------------
perceptron10 = (ajusta_PLA(datos01, etiquetas01, 200, runif(3)))
plot(datos01, col = etiquetas01+3, xlim=c(-50,50), ylim = c(-50,50), xlab = "X",
     ylab = "Y", main="200 iteraciones")
abline(a=perceptron10[1], b=perceptron10[2], lwd=1.5)
cat("Con un máximo de 200 iteraciones, converge a ", perceptron10[3],"iteraciones.")
```

```
## Con un máximo de 200 iteraciones, converge a  201 iteraciones.
```

```r
# Para un máximo de 20000 iteraciones
# -----------------------------------------------------------------------------
perceptron11 = (ajusta_PLA(datos01, etiquetas01, 20000, runif(3)))
plot(datos01, col = etiquetas01+3, xlim=c(-50,50), ylim = c(-50,50), xlab = "X",
     ylab = "Y", main="20000 iteraciones")
abline(a=perceptron11[1], b=perceptron11[2], lwd=1.5)
cat("Con un máximo de 20000 iteraciones, converge a ", perceptron11[3],"iteraciones.")
```

```
## Con un máximo de 20000 iteraciones, converge a  277 iteraciones.
```

```r
# Para un máximo de 200000 iteraciones
# -----------------------------------------------------------------------------
perceptron12 = (ajusta_PLA(datos01, etiquetas01, 200000, runif(3)))
plot(datos01, col = etiquetas01+3, xlim=c(-50,50), ylim = c(-50,50), xlab = "X",
     ylab = "Y", main="200000 iteraciones")
abline(a=perceptron12[1], b=perceptron12[2], lwd=1.5)
```




```
cat("Con un máximo de 200000 iteraciones, converge a ", perceptron12[3],"iteraciones.")
```

```
## Con un máximo de 200000 iteraciones, converge a  493 iteraciones.
```

Como se aprecia en las gráficas, conforme menos iteraciones se tiene para ejecutar el PLA, peor clasifica los datos. Sin embargo, cuanto mayor es el número de iteraciones, mejor ajusta. Esto no quiere decir que siempre sea así, ya que estamos cogiendo los índices aletoriamente y puede que sé de un caso donde los datos sean fáciles, y que con menos iteraciones encuentre la recta que separa los datos linealmente. Pero sí que es verdad, que conforme más iteraciones le demos, mejor ajustará. Ya que tiene más posibilidad, de encontrar la solución óptima.

```
# Después de crear una gráfica o iniciar un apartado paramos la ejecución 3 segundos
Sys.sleep(3)
```

Ahora, realizaremos el mismo proceso que antes, pero iterando 10 veces sobre el PLA. Como he comentado anteriormente, no mostraré las gráficas en este proceso, simplemente el número al que converge.

```
# Para un máximo de 20 iteraciones
# ----------------------------------------------------------------------------------
perceptron13 = replicate(10, ajusta_PLA(datos01, etiquetas01, 20, runif(3)))
cat("Con un máximo de 20 iteraciones, converge a ",
    mean(perceptron13[3]),"iteraciones.")
```

```
## Con un máximo de 20 iteraciones, converge a  25 iteraciones.
```

```
# Para un máximo de 200 iteraciones
# ----------------------------------------------------------------------------------
perceptron14 = replicate(10, ajusta_PLA(datos01, etiquetas01, 200, runif(3)))
cat("Con un máximo de 200 iteraciones, converge a ",
```

```
    mean(perceptron14[3]),"iteraciones.")
```

```
## Con un máximo de 200 iteraciones, converge a  202 iteraciones.
```

```
# Para un máximo de 20000 iteraciones
# ------------------------------------------------------------------------------
perceptron15 = replicate(10, ajusta_PLA(datos01, etiquetas01, 20000, runif(3)))
cat("Con un máximo de 20000 iteraciones, converge a ",
    mean(perceptron15[3]),"iteraciones.")
```

```
## Con un máximo de 20000 iteraciones, converge a  542 iteraciones.
```

```
# Para un máximo de 200000 iteraciones
# ------------------------------------------------------------------------------
perceptron16 = replicate(10, ajusta_PLA(datos01, etiquetas01, 200000, runif(3)))
cat("Con un máximo de 200000 iteraciones, converge a ",
    mean(perceptron16[3]),"iteraciones.")
```

```
## Con un máximo de 200000 iteraciones, converge a  415 iteraciones.
```

Como conlcusión, sabemos que si los datos son separables linealmente, el PLA encontrará una buena solución, pero no sabremos en que iteración lo hará.

```
# Después de crear una gráfica o iniciar un apartado paramos la ejecución 3 segundos
Sys.sleep(3)
```

Ejercicio 3

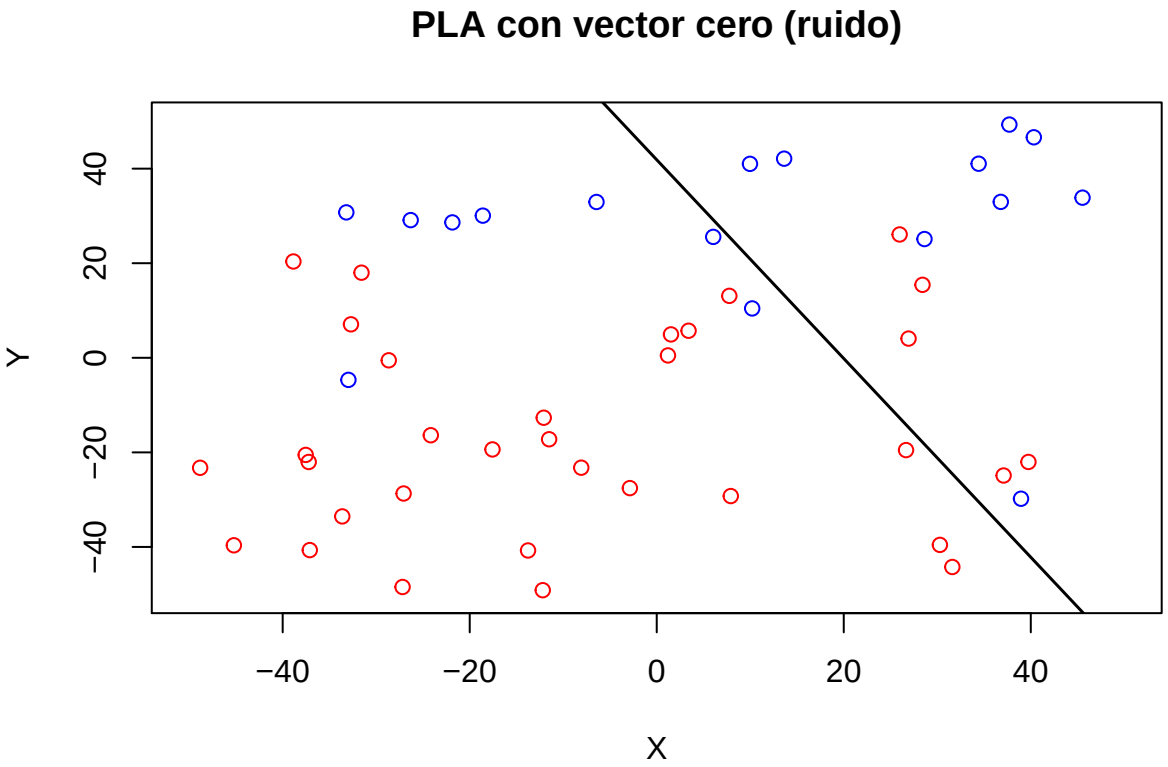
Hacer lo mismo que antes usando ahora los datos del apartado 2b de la sección.1. ¿Observa algún comportamiento diferente? En caso afirmativo diga cual y las razones para que ello ocurra.

Usaremos los datos *del ejercicio 2ab de la sección 1*, por lo que nuestro datos a usar serán los *datos*02 y las etiquetas *etiquetas*02.

a) El vector a cero.

Inicializaremos los pesos del perceptrón a 0.

```
# Dibujamos los puntos en la gráfica
plot (datos02, main = "PLA con vector cero (ruido)", col = etiquetas_ruido01+3, xlab = "X",
      ylab = "Y", xlim = c(-50,50), ylim = c(-50,50))

# Guardamos los valores del PLA en una variable
# Le pasamos al perceptrón un máximo de 2000 iteraciones y el vector inicial a 0
# Ahora usamos las etiquetas con ruido
perceptron_ruido01 = (ajusta_PLA(datos02, etiquetas_ruido01, 2000, c(0,0,0)))

# Representamos la recta hecha por el perceptron
abline(a=perceptron_ruido01[1], b=perceptron_ruido01[2], lwd=1.5)
```




```
# Mostramos el número de iteraciones que hacen falta para converger
cat(perceptron_ruido01[3],"iteraciones para converger.")
```

```
## 2005 iteraciones para converger.
```

Con los resultados obtenidos no podemos sacar una conclusión firme. Aunque más o menos, sabemos que no va a poder separar los datos linealmente. Para ello vamos a realizar el mismo experimento, variando el número máximo de iteraciones. En concreto, probaremos con 200, 20000, 200000 iteraciones. Para así, intentar llegar a una conclusión más fundamentada. Recordad, que seguiremos asignando el vector a cero y que escogemos los índices de forma aleatoria.

```
# Para obtener las gráficas juntas
par(mfrow=c(2,2))

# Para un máximo de 200 iteraciones
# ------------------------------------------------------------------------------
perceptron_ruido02 = (ajusta_PLA(datos02, etiquetas_ruido01, 200, c(0,0,0)))
plot(datos01, col = etiquetas_ruido01+3, , xlab = "X", ylab = "Y",
     main="200 iteraciones", xlim = c(-50,50), ylim = c(-50,50))
abline(a=perceptron_ruido02[1], b=perceptron_ruido02[2], lwd=1.5)
cat("Con un máximo de 200 iteraciones, converge a ",
    perceptron_ruido02[3],"iteraciones.")
```

```
## Con un máximo de 200 iteraciones, converge a  209 iteraciones.
```

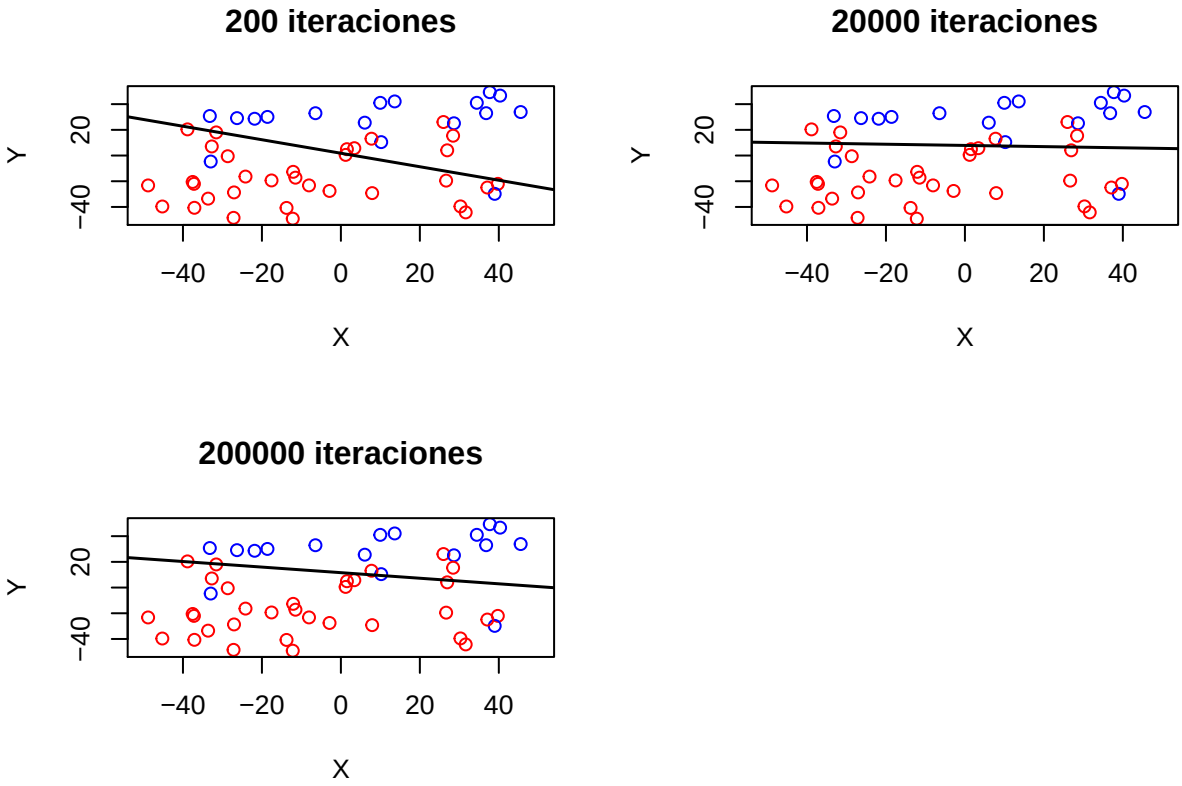
```
# Para un máximo de 20000 iteraciones
# ------------------------------------------------------------------------------
perceptron_ruido03 = (ajusta_PLA(datos02, etiquetas_ruido01, 20000, c(0,0,0)))
plot(datos01, col = etiquetas_ruido01+3, , xlab = "X", ylab = "Y",
     main="20000 iteraciones", xlim = c(-50,50), ylim = c(-50,50))
abline(a=perceptron_ruido03[1], b=perceptron_ruido03[2], lwd=1.5)
```

```r
cat("Con un máximo de 20000 iteraciones, converge a ",
    perceptron_ruido03[3],"iteraciones.")
```

```
## Con un máximo de 20000 iteraciones, converge a  20006 iteraciones.
```

```r
# Para un máximo de 200000 iteraciones
# ------------------------------------------------------------------------------
perceptron_ruido04 = (ajusta_PLA(datos02, etiquetas_ruido01, 200000, c(0,0,0)))
plot(datos01, col = etiquetas_ruido01+3, , xlab = "X", ylab = "Y",
     main="200000 iteraciones", xlim = c(-50,50), ylim = c(-50,50))
abline(a=perceptron_ruido04[1], b=perceptron_ruido04[2], lwd=1.5)
cat("Con un máximo de 20000 iteraciones, converge a ",
    perceptron_ruido04[3],"iteraciones.")
```

```
## Con un máximo de 20000 iteraciones, converge a  200005 iteraciones.
```



Como se aprecia en las gráficas, al haber introducido ruido en las etiquetas, el PLA no va a encontrar una recta que separe los datos linealmente. Por tanto, sino hubiéramos introducido un *número máximo de iteraciones*, nuestro algoritmo no pararía. Como se observa en los resultados, el PLA no converge, sino que llega hasta el final de las iteraciones establecidas por defecto.

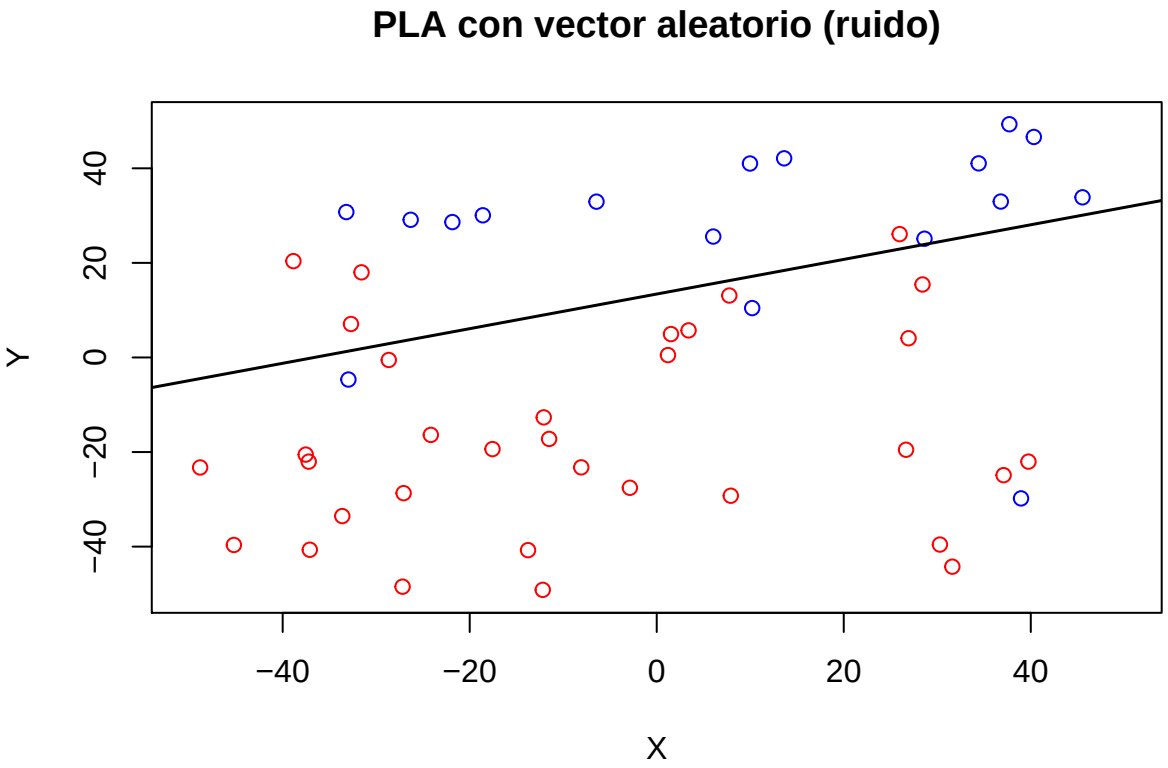
Por tanto, si los datos no son separables linealmente, sabemos que el PLA no encontrará solución, independientemente de las iteraciones.

b) Con vectores de números aleatorios entre 0 y 1, diez veces.

Inicializaremos los pesos del perceptrón a un valor aleatorio entre 0 y 1.

Primero haremos el experimento, sin ejecutarlo 10 veces y comprobaremos el resultado que obtenemos. Luego lo ejecutaremos 10 veces y comprobaremos.

```r
# Dibujamos los puntos en la gráfica
plot (datos02, main = "PLA con vector aleatorio (ruido)", col = etiquetas_ruido01+3, xlab = "X",

      ylab = "Y", xlim = c(-50,50), ylim = c(-50,50))

# Guardamos los valores del PLA en una variable
# Le pasamos al perceptrón un máximo de 2000 iteraciones y el vector inicial a 0
# Ahora usamos las etiquetas con ruido
perceptron_ruido05 = (ajusta_PLA(datos02, etiquetas_ruido01, 2000, runif(3)))

# Representamos la recta hecha por el perceptron
abline(a=perceptron_ruido05[1], b=perceptron_ruido05[2], lwd=1.5)
```




```r
# Mostramos el número de iteraciones que hacen falta para converger
cat(perceptron_ruido05[3],"iteraciones para converger.")
```

```
## 2009 iteraciones para converger.
```

```r
# Después de crear una gráfica o iniciar un apartado paramos la ejecución 3 segundos
Sys.sleep(3)
```

Ahora, hacemos el experimento ejecutando 10 veces el PLA, obtiendo la media de las 10 veces. En este caso no mostraré las gráficas, ya que como sabemos que no converge en una iteración, nunca lo va hacer en las demás, puesto que los datos no son linealmente separables.

```r
# Guardamos los valores del PLA en una variable
# Le pasamos al perceptrón un máximo de 2000 iteraciones
# y el vector con valores aleatorios entre [0,1]
perceptron_ruido06 = replicate(10, ajusta_PLA(datos02, etiquetas_ruido01, 2000, runif(3))[3])
```

```r
# Mostramos el número de iteraciones que hacen falta para converger
# Realizamos la media de las 10 iteraciones
cat(mean(perceptron_ruido06),"iteraciones para converger.")
```

```
## 2005 iteraciones para converger.
```

No hará falta, realizar más experimentos cuando las etiquetas tengan ruido. Puesto que llegamos a la conclusión de que al no ser linealmente separables, el PLA nunca va a encontrar una recta que los separa. Y siempre exprimirá hasta el *número máximo de iteraciones* que le hayamos dado.

Por tanto, en este ejercicio, da igual si el vector está iniciado a cero o aleatoriamente, puesto que los datos al no ser linealmente separables, iterará hasta que se alcance el *número máximo de iteraciones*.

```r
# Después de crear una gráfica o iniciar un apartado paramos la ejecución 3 segundos
Sys.sleep(3)
```

Ejercicio sobre Regresión Lineal

Ejercicio 1

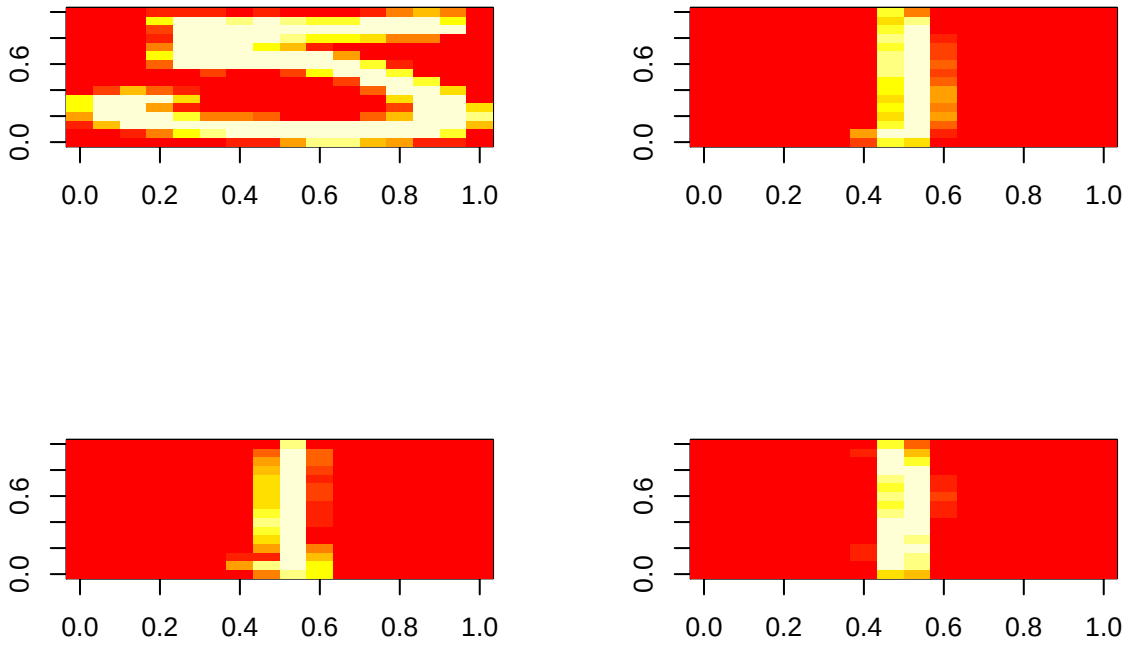
Abra el fichero Zip.info disponible en la web del curso y lea la descripción de la representación numérica de la base de datos de números manuscritos que hay en el fichero Zip.train. Lea el fichero Zip.train dentro de su código y visualice las imágenes (usando paraTrabajo1.R). Seleccione solo las instancias de los números 1 y 5. Guardelas como matrices de tamaño 16x16.

Partimos del código proporcionado por la profesora que aparece en *paraTrabajo.Rmd*.

```r
# Leemos el fichero con los datos de entrenamiento
digit.train <- read.table("datos/zip.train", quote="\"", comment.char="",
                          stringsAsFactors=FALSE)
```

```
## Warning in scan(file = file, what = what, sep = sep, quote = quote, dec =
## dec, : número de items leídos no es múltiplo del número de columnas
```

```r
# Guardamos los 1 y los 5
digitos15.train = digit.train[digit.train$V1==1 | digit.train$V1==5,]

# Etiquetas
digitos.train = digitos15.train[,1]
ndigitos.train = nrow(digitos15.train)

# Se retira la clase y se monta una matriz de tamanio 16x16
grises = array(unlist(subset(digitos15.train,select=-V1)),c(ndigitos.train,16,16))
grises.train = lapply(seq(dim(grises)[1]), function(m) {matrix(grises[m,,],16)})

# Visualizamos las imágenes
par(mfrow=c(2,2))

for(i in 1:4){
  imagen = grises[i,,16:1] # Se rotan para verlas bien
  image(z=imagen)
}
```




```
# Etiquetas correspondientes a las 4 imágenes
digitos.train[1:4]
```

```
## [1] 5 1 1 1
```

```
# Después de crear una gráfica o iniciar un apartado paramos la ejecución 3 segundos
Sys.sleep(3)
```

Ejercicio 2

De cada matriz de números (imagen) vamos a extraer dos características: a) su valor medio y b) su grado de simetría vertical.

- Para calcular el grado de simetría haremos lo siguiente: a) calculamos una nueva imagen invirtiendo el orden de las columnas; b) calculamos la diferencia entre la matriz original y la matriz invertida; c) calculamos la media global de los valores absolutos de la matriz. Conforme más alejado de cero sea el valor más asimétrica será la imagen.

- Representar en los ejes {X = Intensidad Promedio, Y = Simetría}, las instacias seleccionadas de 1's y 5's.

Primero, vamos a proceder a calcular la simetría vertical de cada imagen, es decir, si nosotros partimos el plano por la mitad, comprueba la semejanza que hay entre ambas partes. Para ello nos creamos la siguiente función:

```
simetria <- function(matriz){
  # Matriz_original
  matriz_original = matriz[1:256]

  # Calculamos una nueva imagen invirtiendo el orden de las columnas
  matriz_invertida = matriz[,ncol(matriz):1]

  # Calculamos la diferencia entre la matriz original y la matriz invertida
```

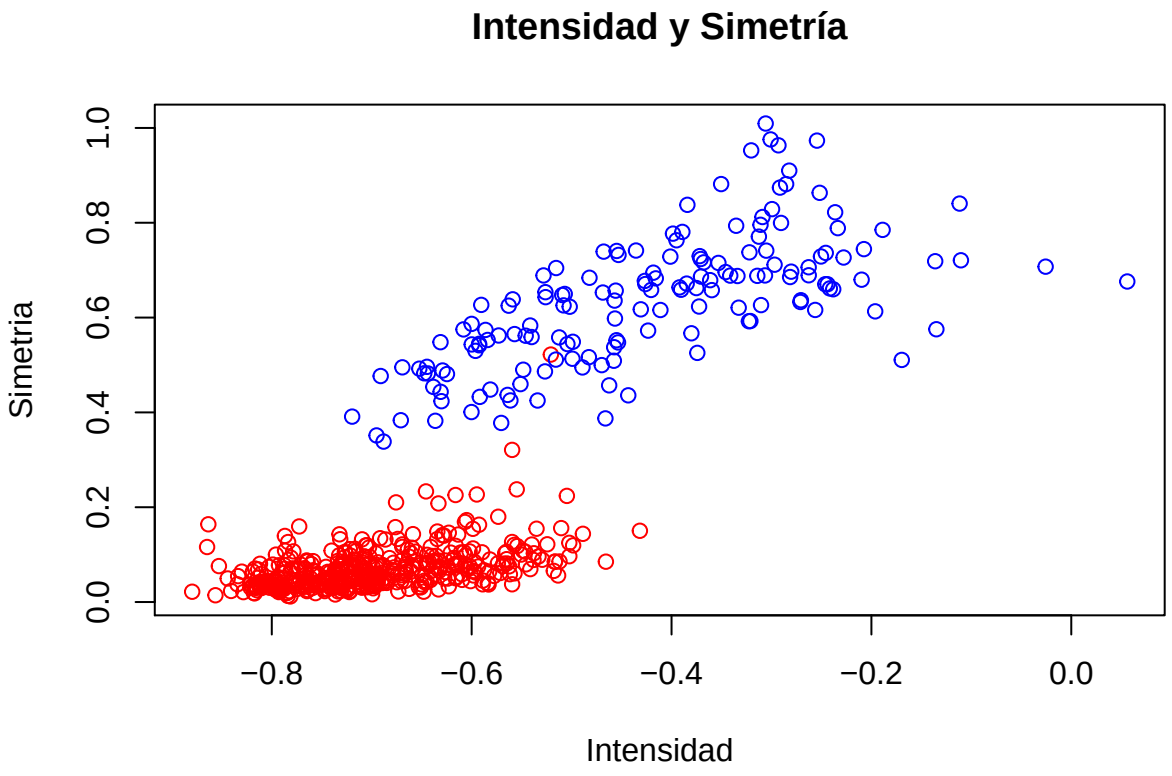
```r
  diferencia_matriz = matriz_original - matriz_invertida

  # Calculamos la media global de los valores absolutos de la matriz
  simetria = mean(abs(diferencia_matriz))

  simetria
}

# Guardamos la simetria de la imagen
simetria = unlist(lapply(grises.train, simetria))
```

Sin embargo, para calcular la intensidad, simplemente escribimos la siguiente línea, puesto que ya hay implementada en R, una función que nos devuelve la media. Ya que la intensidad, devuelve el número medio de blancos y negros (grises).

```r
# Calculamos la intensidad de la imagen
intensidad = unlist(lapply(grises.train, FUN=mean))
```

Ahora, procedemos a representar en los ejes $X = Intensidad\ Promedio$ e $Y = Simetría$, las instacias seleccionadas de 1's y 5's.

```r
# Primero convertimos las etiquetas 1 y 5 a 1 y -1, respectivamente
etiquetas03 <- digitos15.train$V1
etiquetas03 <- (etiquetas03-3)/2

# Dibujamos la nube de puntos
plot(x=intensidad, y=simetria, col=etiquetas03+3, xlab = "Intensidad", ylab = "Simetria",
     main="Intensidad y Simetría")
```



Vemos que la instancia 5, tiene el color azul y está mas dispersa que la instancia 1, que es el rojo. El 1 al ser como un palo, es más simétrico por ambas partes, por eso tiene casi 0 en simetría, es decir, que ambas partes son casi idénticas. Sim embargo, el 5 tiene una mayor relación entre simetría e intensidad.

```
# Después de crear una gráfica o iniciar un apartado paramos la ejecución 3 segundos
Sys.sleep(3)
```

Ejercicio 3

Ajustar un modelo de regresión lineal usando la transformación SVD sobre los datos de (Intensidad promedio, Simetria) y pintar la solución obtenida junto con los datos usados en el ajuste. Las etiquetas serán {-1, 1}. Valorar la bondad del resultado usando Ein y Eout (usar Zip.test). (usar Regress_Lin(datos, label) como llamada para la función).

Nos creamos una función para la regresión, para ello partimos del algoritmo descrito en el libro *Learning from Data: A short course* y de las transparencias y anotaciones de clase. Usaremos la transformación SVD.

Para calcular la pseudoinversa $X^{T_{cruz}}$, sabemos que $X^{T_{cruz}} = (X^T * X)^{-1} * X^T$, y sabemos que $X = U * D * V^T$, es decir, la descomposición de X en valores singulares. Por tanto:

$$X^T * X = V * D^{T_{cruz}} * U^T * U * D * V^T$$

Como $U^T * U = 1$, lo quitamos, y $D^{T_{cruz}} = D$, y la expresión quedaría:

$$X^T * X = V * D * D * V^T = V * D^2 * V^T$$

Por útimo, hacemos $w_{lin} = X^{T_{cruz}} * y$

Implementemos esto en una función:

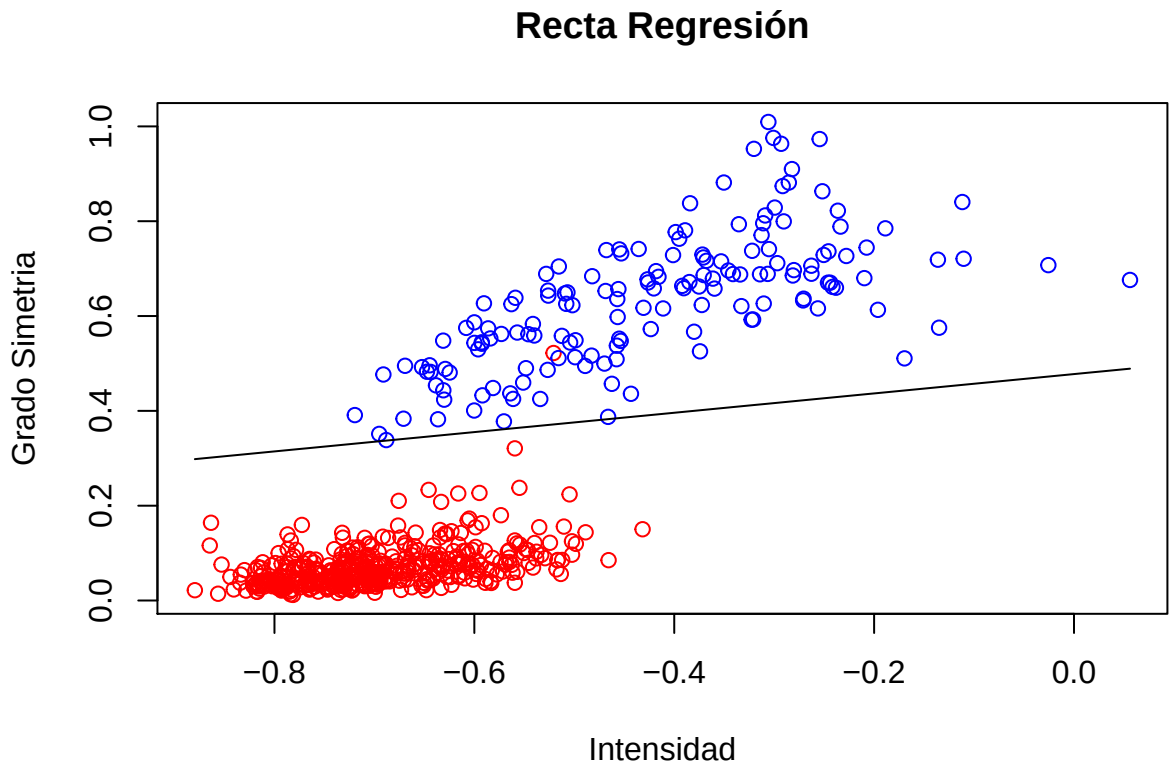
```
Regress_Lin <- function(datos, label) {
  # Añadimos una columna a los datos, para hacer el producto
  x <- cbind(1, data.matrix(datos))

  # Obtenemos la transformación SVD
  x.svd <- svd(x)

  # Descomposición de X en valores singulares
  v <- x.svd$v
  d <- x.svd$d

  # Comprobamos todos los términos de la diagonal
  diagonal <- diag(ifelse(d>0.0001, 1/d, d))

  # Calculamos (X^T) * X = V * (D^2) * (V^T)
  xx <- v %*% (diagonal^2) %*% t(v)

  # Calculamos la pseudoinversa: (X^T) * etiqueta
  pseudoinversa <- xx %*% t(x)

  # Calculamos wlin
  w <- pseudoinversa %*% as.numeric(label)
```

```
  # Devolvemos los coeficientes del hiperplano: w1 + w2*x + w3*y = 0
  c(-w[2]/w[3], -w[1]/w[3])
}

# Juntamos los valores obtenidos de simetria e intensidad
datos.train <- cbind(intensidad, simetria)

# Calculamos la recta de regresión que separa ambas clases
recta_regresion01 = Regress_Lin(datos.train, etiquetas03)

# Mostramos los coeficientes del hiperplano
recta_regresion01
```

```
## [1] 0.2038923 0.4775669
```

Como hemos obtenidos los coeficientes del hiperplano $y = (\frac{w_2}{-w_3}) * x + (\frac{w_1}{-w_3})$, ya podemos crear la recta que separará ambas clases.

```
# Dibujamos los puntos
plot(x=intensidad, y=simetria, col=etiquetas03+3, xlab="Intensidad",
     ylab="Grado Simetria", main="Recta Regresión")

# Dibujamos la recta
curve(0.2038923*x+0.4775669, add=T)
```




```
# Después de crear una gráfica o iniciar un apartado paramos la ejecución 3 segundos
Sys.sleep(3)
```

Valorar la bondad del resultado usando Ein y Eout (usar Zip.test)

Ahora, vamos a calcular el error dentro de la muestra E_{in} y fuera de la muestraE_{out}.

a) E_{in}

Para obtener el error dentro de la muestra, partimos de los datos de entrenamiento E_{in}, es decir, de las etiquetas que son distintas entre todas las etiquetas. Lo que queremos hacer es predecir cuántas veces nos equivocamos al predecir la etiqueta. Para ello, hace falta tener leído el archivo *Zip.train*.

```
# Cuando se utilizan números aleatorios, es recomedable establecer una semilla, para
# que al repetir la generación de los números, siempre obtengamos los mismos
set.seed(3)

# Generamos las etiquetas aleatoriamente
datos04 <- simula_unif(N = 100, dim = 2, rango = c(-10:10))

# Generamos una recta
recta03 <- simula_recta(intervalo = c(-10:10))

# Etiquetamos los puntos a partir de la función de la recta
etiquetas04 <- sign(datos04[,2] - recta03[1]*datos04[,1] - recta03[2])

# Calculamos la recta de regresión que separa ambas clases
recta_regresion02 <- Regress_Lin(datos04, etiquetas04)

# Etiquetamos los puntos a partir de la recta de regresión
etiquetas_regresion01 <- sign(datos04[,2] -
                              recta_regresion02[2]*datos04[,1] -
                              recta_regresion02[2])

# Calculamos Ein como las etiquetas que son distintas, partido el total de las etiquetas
Ein = sum(etiquetas04 != etiquetas_regresion01) / length(etiquetas04)

Ein
```

```
## [1] 0.28
```

Obtenemos un error dentro de la muestra de 0.28. En los datos de entrenamiento tenemos 599 muestras y como vemos en la anterior gráfica, solo comete un error, por lo tanto el error es cercano a 0.

b) E_{out}

Ahora vamos a calcular el error fuera de la muestra, básicamente, es el mismo procedimiento que para E_{in}, solo que ahora usaremos el conjunto de datos de test, usando para ello el archivo *Zip.test*

Primero procedemos a leer los datos del archivos *zip.test*.

```
# Leemos el fichero con los datos de test
digit.test <- read.table("datos/zip.test", quote="\"", comment.char="",
                         stringsAsFactors=FALSE)
```

```
## Warning in scan(file = file, what = what, sep = sep, quote = quote, dec =
## dec, : número de items leídos no es múltiplo del número de columnas
```

```
# Guardamos los 1 y los 5
digitos15.test = digit.test[digit.test$V1==1 | digit.test$V1==5,]

# Etiquetas
digitos.test = digitos15.test[,1]
ndigitos.test = nrow(digitos15.test)

# Se quita la clase y se monta una matriz 16x16
grises = array(unlist(subset(digitos15.test,select=-V1)),c(ndigitos.test,16,16))
grises.test = lapply(seq(dim(grises)[1]), function(m) {matrix(grises[m,,],16)})

# Recordamos que las etiquetas que tenemos son 5 y 1
# por tanto las pasamos a 1 y -1
etiquetas.test <- digitos15.test$V1
etiquetas.test <- (etiquetas.test-3)/2
```

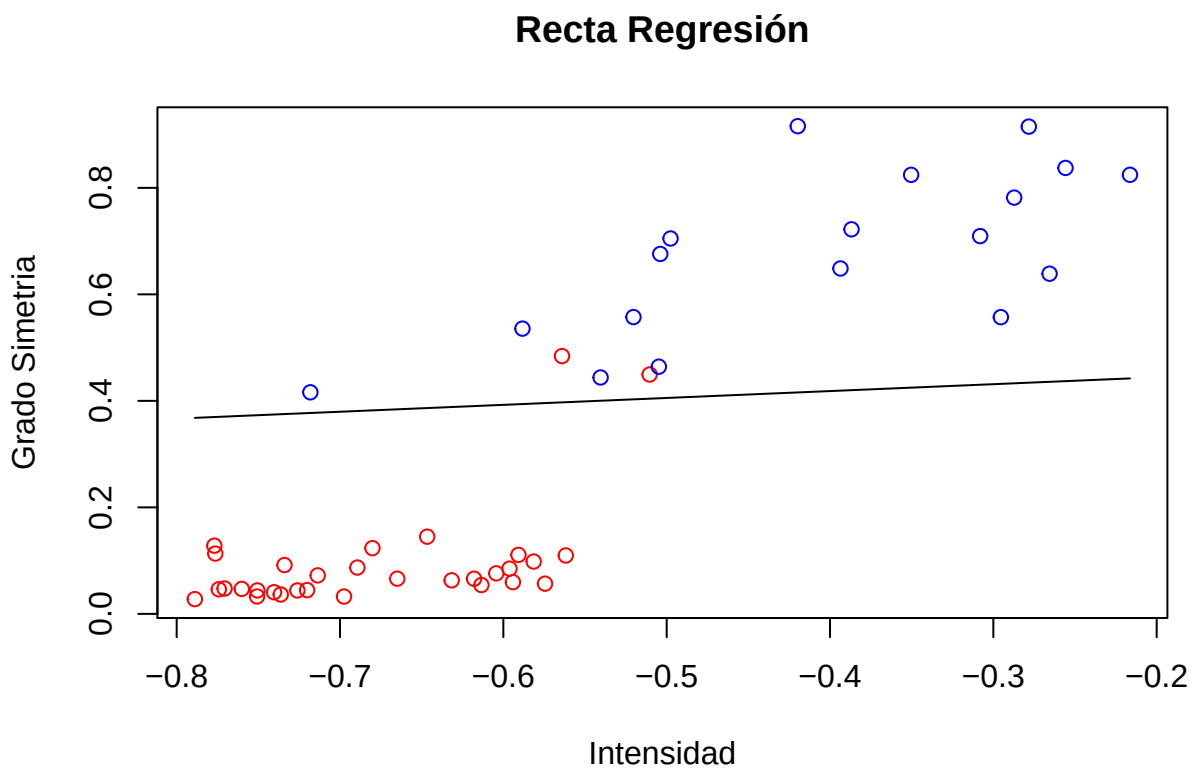
Luego, creamos una recta de regresión que corte los datos de test.

```
# Creamos un array con la media de la imagen
intensidad = unlist(lapply(grises.test, FUN=mean))

# Creamos uan función para calcular la simetría
simetria <- function(matriz){
  # Matriz_original
  matriz_original = matriz[1:256]

  # Calculamos una nueva imagen invirtiendo el orden de las columnas
  matriz_invertida = matriz[,ncol(matriz):1]

  # Calculamos la diferencia entre la matriz original y la matriz invertida
  diferencia_matriz = matriz_original - matriz_invertida

  # Calculamos la media global de los valores absolutos de la matriz
  simetria = mean(abs(diferencia_matriz))

  simetria
}

# Creamos un array con la simetría de la imagen
simetria = unlist(lapply(grises.test, simetria))

# Juntamos los valores obtenidos de simetria e intensidad
datos.test <- cbind(intensidad, simetria)

# Calculamos la recta de regresión que separa ambas clases
recta_regresion02 = Regress_Lin(datos.test, etiquetas.test)

# Mostramos los coeficientes del hiperplano
recta_regresion02
```

```
## [1] 0.1293491 0.4701312
```

Como hemos obtenidos los coeficientes del hiperplano $y = (\frac{w_2}{-w_3}) * x + (\frac{w_1}{-w_3})$, ya podemos crear la recta que separará ambas clases, es decir, el mejor ajuste que tenga.

```
# Dibujamos los puntos
plot(x=intensidad, y=simetria, col=etiquetas.test+3, xlab="Intensidad",
     ylab="Grado Simetria", main="Recta Regresión")

# Dibujamos la recta
curve(0.1293491*x+0.4701312, add=T)
```




```
# Después de crear una gráfica o iniciar un apartado paramos la ejecución 3 segundos
Sys.sleep(3)
```

Procedemos a calcular E_{out}, que es el error fuera de la muestra y se calcula a partir de los datos de train con el conjunto de test.

```
# Cuando se utilizan números aleatorios, es recomedable establecer una semilla, para
# que al repetir la generación de los números, siempre obtengamos los mismos
set.seed(3)

# Generamos los datos aleatoriamente
datos05 <- simula_unif(N = 100, dim = 2, rango = c(-10:10))

# Generamos una recta
recta04 <- simula_recta(intervalo = c(-10:10))

# Etiquetamos las etiquetas a partir de la función de la recta
etiquetas05 <- sign(datos05[,2] - recta04[1]*datos05[,1] - recta04[2])

# Usamos la recta de regresión creada en los datos de entrenamiento
etiquetas_regresion02 <- sign(datos05[,2] -
                              recta_regresion02[2]*datos05[,1] -
```

```
                                    recta_regresion02[2])
# Calculamos Ein como las etiquetas que son distintas, partido el total de las etiquetas
Eout = sum(etiquetas05 != etiquetas_regresion02) / length(etiquetas05)

Eout
```

```
## [1] 0.28
```

Al igual que en el conjunto train, hay datos con ruido, ya que se encuentran en la clase contraria. El error fuera de la muestra E_{out} ha sido 0.72, superior al error obtenido dentro de la muestra. Esto es debido a que ahora han intervenido datos nuevos. Pero, ambos errores son parecidos, por lo que se puede decir que las muestras son i.i.d (independientes e idénticamente distribuidas).

```
# Después de crear una gráfica o iniciar un apartado paramos la ejecución 3 segundos
Sys.sleep(3)
```

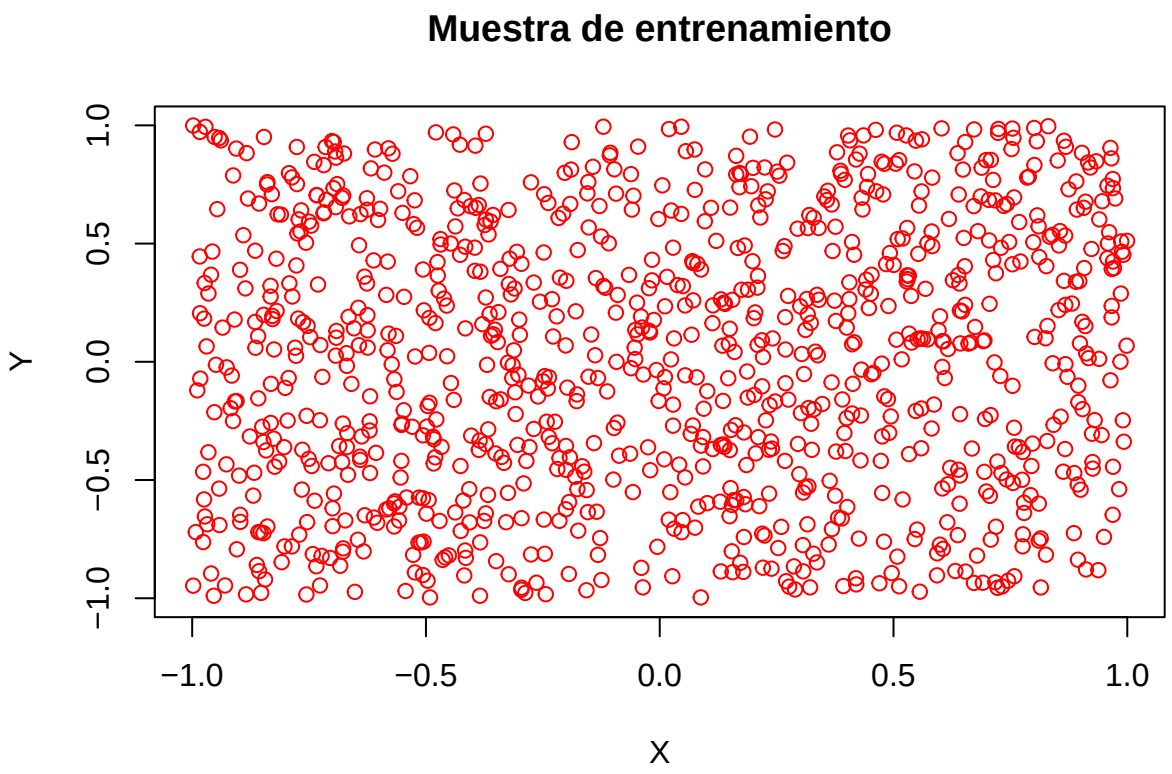
Ejercicio 4

En este apartado exploramos como se transforman los errores E_{in} y E_{out} cuando aumentamos la complejidad del modelo lineal usado. Ahora hacemos uso de la función $simula\_unif(N, 2, size)$ que nos devuelve N coordenadas de puntos uniformemente muestreados dentro del cuadrado definido por $[-size, size]$

Experimento 1:

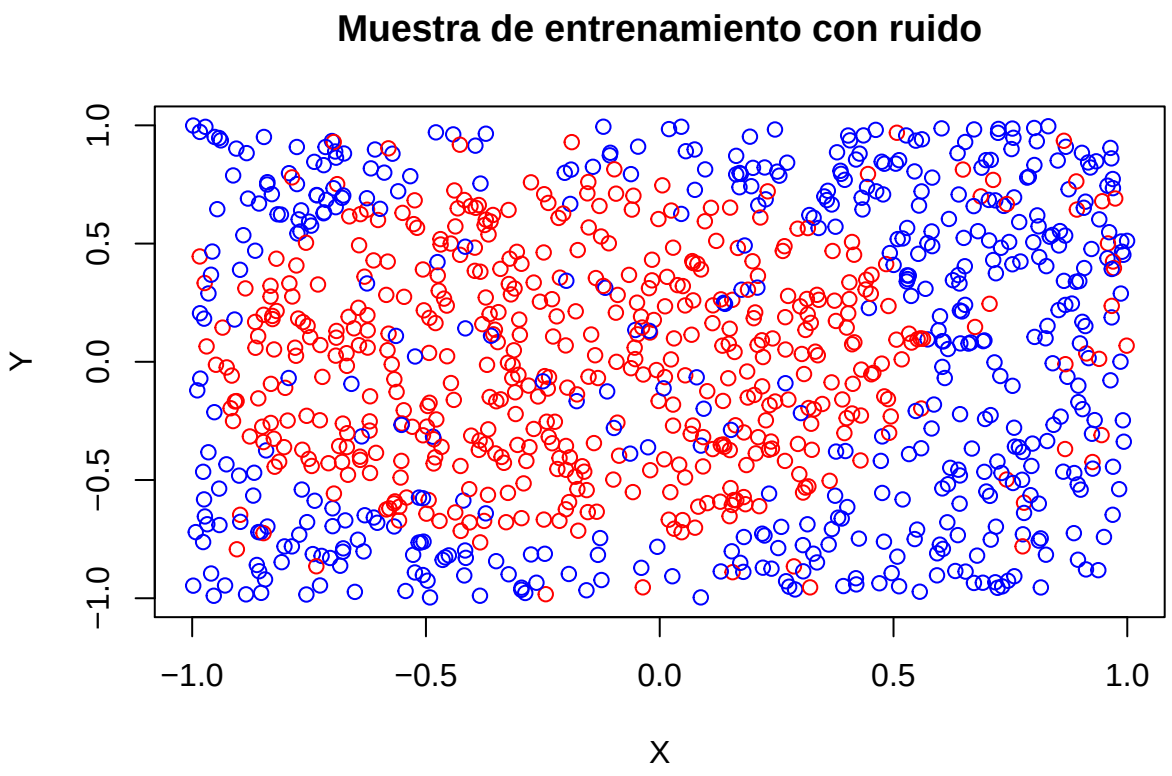
a) Generar una muestra de entrenamiento de $N = 1000$ puntos en el cuadrado $X = [-1, +1]x[-1, +1]$. Pintar el mapa de puntos $2D$.

```
# Cuando se utilizan números aleatorios, es recomedable establecer una semilla, para
# que al repetir la generación de los números, siempre obtengamos los mismos
set.seed(3)

# Generamos los datos aleatoriamente
datos06 = simula_unif(N=1000, dim = 2, rango=c(-1,1))

# Dibujamos la nube de puntos
plot (datos06, col = 2, xlab = "X", ylab = "Y", xlim =c(-1,1), ylim=c(-1,1),
      main="Muestra de entrenamiento")
```



b) Consideremos la función $(x[,1]+0.2)^2+x[,2]^2-0.6)$ que usaremos para asignar una etiqueta a cada punto de la muestra anterior. Introducimos ruido sobre las etiquetas cambiando aleatoriamente el signo de un diez por ciento de las mismas. Pintar el mapa de etiquetas final.

```
# Aplicamos la función a cada punto y almacenamos el signo del resultado en una lista
etiquetas06 = sign((datos06[,1]+0.2)^2+datos06[,2]^2-0.6)
# con esto acabamos de etiquetar cada punto

# Introducimos el ruido en las etiquetas
etiquetas_ruido02 = generar_ruido(etiquetas06,10)

# Representamos los puntos
plot (datos06, col = etiquetas_ruido02+3, xlab = "X", ylab = "Y", xlim =c(-1,1), ylim=c(-1,1),
      main="Muestra de entrenamiento con ruido")
```

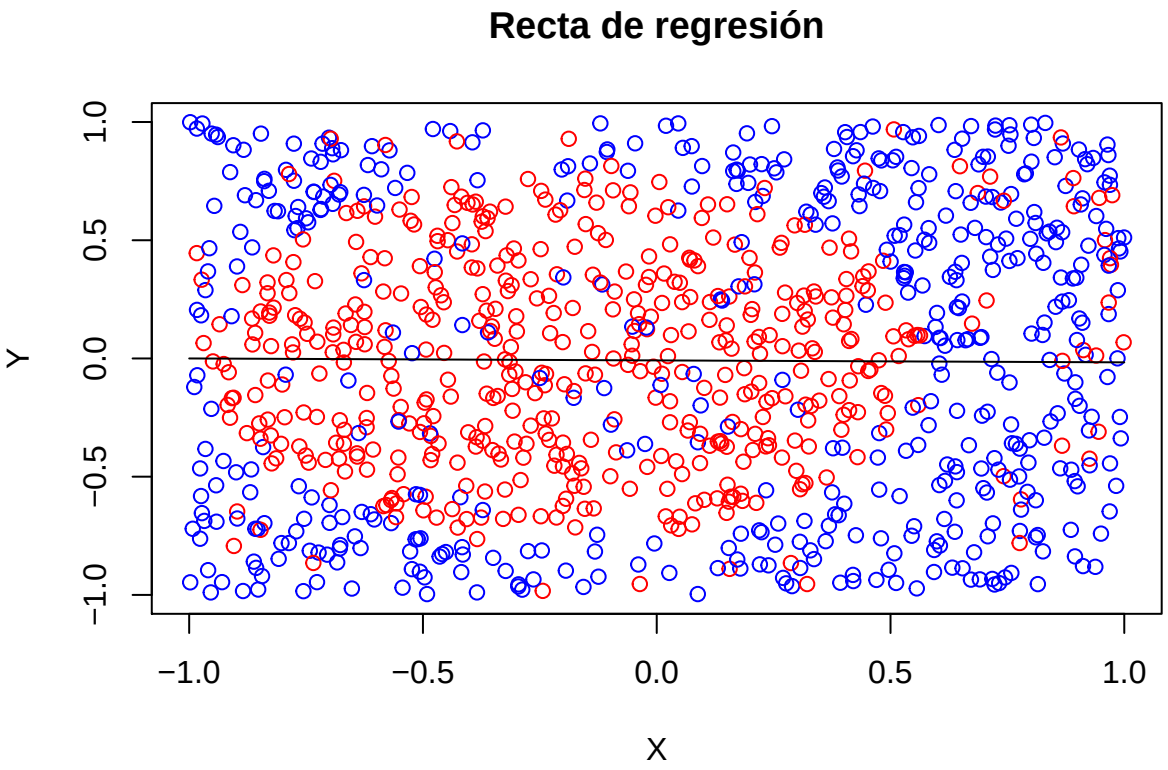


c) Usando como vector de características $(1, x_1, x_2)$ ajustar un modelo de regresion lineal al conjunto de datos generado y estimar los pesos w. Estimar el error de ajuste E_{in}.

```
# Añadimos a los datos una columna
datos07 <- cbind(1, datos06)

# Calculamos los coeficientes del hiperplano
recta_regresion04 = Regress_Lin(datos07, etiquetas_ruido02)

# Los mostramos por pantalla
recta_regresion04
```

```
## [1] -0.008209552 -0.008209552
```

```
# Dibujamos la nube de puntos
plot(datos06, col=etiquetas_ruido02+3, xlab = "X", ylab = "Y", xlim =c(-1,1), ylim=c(-1,1),
     main="Recta de regresión")

# Dibujamos la recta
curve(-0.008209552*x-0.008209552, add=T)
```



Como se observa en la gráfica, no existe un modelo lineal que separe los datos, debido a que existe ruido y el área que tiene la otra clase corresponde a una elipse. Además, los datos nos van a ser linealmente separables, debido al ruido. Pero si en los datos no hubiésemos introducido ruido, los datos se podrían separar de forma circular, no linealmente. Ya que los intenta clasificar, pero no puede.

```
# Después de crear una gráfica o iniciar un apartado paramos la ejecución 3 segundos
Sys.sleep(3)
```

Vamos a calcular el error dentro de la muestra:

```
# Etiquetamos los puntos a partir de la recta de regresión
etiquetas_regresion05 <- sign((datos07[,1]+0.2)^2+datos07[,2]^2-0.6)

# Calculamos Ein como las etiquetas que son distintas, partido el total de las etiquetas
ein1 = sum(etiquetas_ruido02 != etiquetas_regresion05) / length(etiquetas_ruido02)

ein1
```

```
## [1] 0.496
```

El error en el conjunto de entrenamiento E_{in} es de 0.496, esto quiere decir que tiene un cincuenta por ciento de error, con lo que nuestro clasificador es malo.

Ya que los datos no son linealmente seprabales, puesto que existe una gran zona en medio de datos de otra clase.

d) Ejecutar todo el experimento definido por (a)-(c) 1000 veces (generamos 1000 muestras diferentes).

Calcular el valor medio de los errores Ein de las 1000 muestras.

Generar 1000 puntos nuevos por cada iteración y calcular con ellos el valor de E_{out} en dicha iteración. Calcular el valor medio de Eout en todas las iteraciones.

Primero modificamos la función que habíamos creado de *Regresión Lineal*, ya que solo devolvíamos los coeficientes del hiperplano. Ahora simplemente, nos creamos una nueva para que devuelva los pesos, ya que ahora vamos a trabajar con otro vector de características distinto.

```
Regress_Lin1 <- function(datos, label) {
  # Añadimos una columna a los datos, para hacer el producto escalar
  x <- cbind(1, data.matrix(datos))

  # Obtenemos la transformación SVD
  x.svd <- svd(x)

  # Descomposición de X en valores singulares
  v <- x.svd$v
  d <- x.svd$d

  # Comprobamos todos los términos de la diagonal
  diagonal <- diag(ifelse(d>0.0001, 1/d, d))

  # Calculamos (X^T) * X = V * (D^2) * (V^T)
  xx <- v %*% (diagonal^2) %*% t(v)

  # Calculamos la pseudoinversa que (X^T) * etiqueta
  pseudoinversa <- xx %*% t(x)

  w <- pseudoinversa %*% as.numeric(label)

  # Devolvemos los pesos
  c(w)
}
```

Ahora procemos a calcular el error dentro y fuera de la muestra.

```
# Establecemos una semilla por defecto, para la generación de números aleatorios
set.seed(3)

# Establecemos eout a cero
e1 = 0
# Establecemos ein a cero
e2 <- 0

# Ejecutar el experimento 1000 veces
for (i in seq(1,1000)){

  # Ein
  # ------------------------------------------------------------------------
  # Generamos aleatoriamente una lista de números aleatorios
  datos06 = simula_unif(N=1000, dim=2, rango=c(-1,1))
  # Clasificamos los puntos
```

```
  etiquetas06 = sign((datos06[,1]+0.2)^2+datos06[,2]^2-0.6)
  # Introducimos el ruido en las etiquetas
  etiquetas_ruido02 = generar_ruido(etiquetas06, 10)
  # Calculamos los coeficientes del hiperplano
  recta_regresion04 = Regress_Lin1(datos07, etiquetas_ruido02)
  # Etiquetamos los puntos a partir de la recta de regresión
  recta = c(-recta_regresion04[1]/recta_regresion04[3],
            - recta_regresion04[2]/recta_regresion04[3])
  etiquetas_regresion05 <- sign(datos07[,2] - recta[1]*datos07[,1] - recta[2])
  # Calculamos el valor de Ein
  e2 = e2 + sum(etiquetas_ruido02 != etiquetas_regresion05) / length(etiquetas_ruido02)

  # Eout
  # ----------------------------------------------------------------------------
  # Generamos aleatoriamente una lista de números aleatorios
  datos06.test = simula_unif(N=1000, dim = 2, rango=c(-1,1))
  # Clasificamos los puntos
  etiquetas07 = sign((datos06.test[,1]+0.2)^2+datos06.test[,2]^2-0.6)
  # Introducimos el ruido en las etiquetas
  etiquetas_ruido03 = generar_ruido(etiquetas07, 10)
  # Etiquetamos los puntos a partir de la recta de regresión ya creada
  etiquetas_regresion05.test <- sign(datos06.test[,2] -
                                       recta[1]*datos06.test[,1] - recta[2])
  # Calculamos el valor de Eout
  e1 = e1 + sum(etiquetas_ruido03 != etiquetas_regresion05.test) /
    length(etiquetas_ruido03)
}

cat ("Ein:", e2/1000)
```

```
## Ein: 0.501221
```

```
cat("Eout: ", e1/1000)
```

```
## Eout:  0.499632
```

El error de la muestra de entrenamiento es del alrededor ciento por ciento. Es de esperar este resultado, porque sabíamos, que los datos no eran linealmente separables. Ambos dan alrededor del 50%, esto nos quiere decir que el clasificador es malo. Ya que tenemos la mitad de probabilidad de acertar.

e) Valore que tan bueno considera que es el ajuste con este modelo lineal a la vista de los valores medios obtenidos de E_{in} y E_{out}

Como hemos podido ver, el error obtenido dentro y fuera de la muestra es casi el mismo, por lo no que podemos decir mucho acerca del clasificador, a expensar de saber que los datos no son separables linealmente.

Experimento 2:

a) Ahora vamos a repetir el mismo experimento anterior pero usando características no lineales. Ahora usaremos el siguiente vector de características:. Ajustar el nuevo modelo de regresión lineal y calcular el nuevo vector de pesos w. Calcular el error E_{in}

```
ein4 <- 0
```

```r
# Cuando se utilizan números aleatorios, es recomedable establecer una semilla, para
# que al repetir la generación de los números, siempre obtengamos los mismos
set.seed(3)

# Generamos los datos
datos08 <- simula_unif(N=1000,dim = 2, c(-10,10))

# Etiquetamos cada punto, partimos de la misma función mencionada en el experimento 1
etiquetas07 = sign((datos08[,1]+0.2)^2+ datos08[,2]^2-0.6)

# Introducimos el ruido en las etiquetas
etiquetas_ruido03 = generar_ruido(etiquetas07, 10)

# Añadimos a los datos, el vector de caracteristicas mencionado en el enunciado
datos08.train <- cbind(1, datos08[,1], datos08[,2], datos08[,1]*datos08[,2],
                       datos08[,1]^2, datos08[,2]^2)

# Calculamos los coeficientes del hiperplano
recta_regresion06 = Regress_Lin1(datos08.train, etiquetas_ruido03)

etiquetas_regresion05 <- sign(sum(1*recta_regresion06[2],
                                  datos08.train[,2]*recta_regresion06[3],
                                  datos08.train[,3]*recta_regresion06[4],
                                  datos08.train[,4]*recta_regresion06[5],
                                  datos08.train[,5]*recta_regresion06[6],
                                  datos08.train[,6]*recta_regresion06[7]))

# Medimos el error
ein4 =  0.001*length(which(etiquetas_regresion05 != etiquetas_ruido03))

cat("Ein: ", ein4)
```

```
## Ein:  0.107
```

Como observamos, al tomar los nuevos datos, hemos obtenido un error del 10%, ya que hemos introducido ruido en un 20% de las muestras, es decir, 10% en cada clase. Por lo tanto, neustro datos podrán ser separados por una recta.

```r
# Después de crear una gráfica o iniciar un apartado paramos la ejecución 3 segundos
Sys.sleep(3)
```

b) Al igual que en el experimento anterior repetir el experimento 1000 veces calculando con cada muestra el error dentro y fuera de la muestra, Ein e Eout respectivamente. Promediar los valores obtenidos para ambos errores a lo largo de las muestras.

Como le hemos metido un 20% de ruido, el error dentro de la muestra debería de ser aproximadamente 0.2. Ahora, vamos a hacer el clasificador (train) que son:

$$E_{in} = etiquetas.train! = clasificador(train)/etiquetas.total$$

y a partir, de él, obtendremos nuestro error fuera de la muestra como:

$$E_{out} = etiquetas.test! = clasificador(test)/etiquetas.total$$

Para E_{out}, usaremos la recta de regresión usada en la muestra de entrenamiento.

```
# Cuando se utilizan números aleatorios, es recomedable establecer una semilla, para
# que al repetir la generación de los números, siempre obtengamos los mismos
set.seed(3)

eout3 <- 0
ein4 <-0

# Ejecutar el experimento 1000 veces
for (i in seq(1,1000)){

  # Ein
  # ------------------------------------------------------------------------
  # Generamos los datos
  datos08 <- simula_unif(N=1000,dim = 2, c(-10,10))
  # Etiquetamos cada punto, partimos de la misma función mencionada en el experimento 1
  etiquetas07 = sign((datos08[,1]+0.2)^2+ datos08[,2]^2-0.6)
  # Introducimos el ruido en las etiquetas
  etiquetas_ruido03 = generar_ruido(etiquetas07, 10)
  # Añadimos a los datos, el vector de caracteristicas mencionado en el enunciado
  datos08.train <- cbind(1, datos08[,1], datos08[,2], datos08[,1]*datos08[,2],
                         datos08[,1]^2, datos08[,2]^2)
  # Calculamos los coeficientes del hiperplano
  recta_regresion06 = Regress_Lin1(datos08.train, etiquetas_ruido03)
  etiquetas_regresion05 <- sign(sum(1*recta_regresion06[2],
                                    datos08.train[,2]*recta_regresion06[3],
                                    datos08.train[,3]*recta_regresion06[4],
                                    datos08.train[,4]*recta_regresion06[5],
                                    datos08.train[,5]*recta_regresion06[6],
                                    datos08.train[,6]*recta_regresion06[7]))

  # Medimos el error dentro de la muestra
  ein4 = ein4 + 0.001*length(which(etiquetas_regresion05 != etiquetas_ruido03))

  # EOUT
  # ------------------------------------------------------------------------
  # Generamos los datos aleatoriamente
  datos09 = simula_unif(N=1000, dim = 2, rango=c(-1,1))
  # Etiquetamos cada punto, partimos de la misma función mencionada en el experimento 1
  etiquetas08 = sign((datos09[,1]+0.2)^2+ datos09[,2]^2-0.6)
  # Introducimos el ruido en las etiquetas
  etiquetas_ruido04 = generar_ruido(etiquetas08, 10)
  # Añadimos a los datos, el vector de caracteristicas mencionado en el enunciado
  datos09.test <- cbind(1, datos09[,1], datos09[,2], datos09[,1]*datos09[,2],
                        datos09[,1]^2, datos09[,2]^2)
  # Calculamos los coeficientes del hiperplano
  etiquetas_regresion06 <- sign(sum(1*recta_regresion06[2],
                                    datos09.test[,2]*recta_regresion06[3],
                                    datos09.test[,3]*recta_regresion06[4],
                                    datos09.test[,4]*recta_regresion06[5],
                                    datos09.test[,5]*recta_regresion06[6],
                                    datos09.test[,6]*recta_regresion06[7]))
```

```
  # Calculamos el error fuera de la muestra
  eout3 = eout3 + 0.001*length(which(etiquetas_regresion06 != etiquetas_ruido04))
}

cat("Ein: ", ein4/1000)
```

```
## Ein:  0.29988
```

```
cat("Eout: ", eout3/1000)
```

```
## Eout:  0.478203
```

Al ejecutar el experimento 1000 veces obtenemos que $E_{in} = 0.29$ y que $E_{out} = 0.47$. Vemos que en el conjunto de entrenamiento, estamos cerca del 20% del ruido introducido, mientras que para el conjunto test, el porcentaje de error sube. Esto se debe, a que en el *test* existen datos nuevos que se intentan ajustar a los datos a los del *train*.

```
# Después de crear una gráfica o iniciar un apartado paramos la ejecución 3 segundos
Sys.sleep(3)
```

c) Valore el resultados de este EXPERIMENTO-2 a la vista de los valores medios de los errores E_{in} y E_{out}

Al ejecutar el experimento dos obtenemos unos porcentajes de error coherentes. A veces, cuando ajustados los datos del *test* al *train* obtenemos que el error fuera de la muestra sea más alto, ya que al ser datos nuevos, el algoritmo intenta ajusta a los datos de entrenamiento.

A la vista de los resultados de los errores promedios E_{in} y E_{out} obtenidos en los dos experimentos ¿Que modelo considera que es el más adecuado? Justifique la decisión.

Considero que el experimento 2 sería más adecuado, ya que en el primer experimento ambos errores son parecidos y con un valor más alto. Como en el experimento 1, hemos obtenido cerca del 50% de error, es como si tuviéramos la misma probabilidad que tirar una moneda. Sin embargo, en el experimento 2, los datos de *test* se ajustan mejor a los de *train*. Y el error dentro de la muestra nos dice que los datos son más separables linealmente.

Bonus

Ejercicio 1

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 y elegimos muestras aleatorias uniformes dentro de X. La función f en cada caso será una recta aleatoria que corta a X y que asigna etiqueta a cada punto de X con el valor del signo de f en dicho punto. 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 tamaño de muestra $N = 100$. Usar regresión lineal para encontrar una primera solución g y evaluar E_in, (el porcentaje de puntos incorrectamente clasificados). Repetir el experimento 1000 veces y promediar los resultados ¿Qué valor obtiene para Ein?

```r
error_in = 0

for (i in seq(1,1000)) {
  # Cuando se utilizan números aleatorios, es recomedable establecer una semilla, para
  # que al repetir la generación de los números, siempre obtengamos los mismos
  set.seed(3)

  # Generamos los datos
  datos09 <- simula_unif(N=100,dim = 2, c(-10,10))

  # Generamos la recta
  recta05 <- simula_recta(intervalo = c(-10,10))

  # Generamos las etiquetas dadas por la recta
  etiquetas08 <- sign(datos09[,2] - recta05[1]*datos09[,1] - recta05[2])

  # Homgeneizamos datos
  datos10 <- cbind(1, datos09)

  # Obtenemos vector de pesos
  recta_regresion05 <- Regress_Lin(datos10, etiquetas08)

  # Etiquetas dadas por el vector de pesos
  etiquetas_regresion06 <- sign(datos10[,2] - recta_regresion05[1]*datos10[,1] - recta_regresion05[2])

  # Medimos el error
  error_in = error_in + 0.01*length(which(etiquetas_regresion06 != etiquetas08))
  }

cat("Porcentaje ein: ", error_in/100, "%\n")
```

```
## Porcentaje ein:  7.7 %
```

El porcentaje medio de E_{in} nos devuelve que existe un 5% de error, a expensas de que los datos sean linealmente separables, existe una pequeña probabilidad de error en el algoritmo de regresión.

```r
# Después de crear una gráfica o iniciar un apartado paramos la ejecución 3 segundos
Sys.sleep(3)
```

b) Fijar el tamaño 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. ¿Qué valor obtiene de E_{out}? Valore el resultado.

```r
error_out = 0;

for (i in seq(1,1000)) {

  # Cuando se utilizan números aleatorios, es recomedable establecer una semilla, para
  # que al repetir la generación de los números, siempre obtengamos los mismos
  set.seed(3)

  # Generamos los datos de entrenamiento
```

```
  datos11 <- simula_unif(N=100, 2, c(-10,10))

  # Generamos los datos de test
  datos11.test <- simula_unif(N=1000, 2, c(-10,10))

  # Generamos recta
  recta06 <- simula_recta(intervalo = c(-10,10))

  # Etiquetas de la recta para los datos de entrenamiento
  etiquetas09 <- sign(datos11[,2] - recta06[1]*datos11[,1] - recta06[2])

  # Etiquetas de la recta para los datos de test
  etiquetas09.test <- sign(datos11.test[,2] - recta06[1]*datos11.test[,1]-recta06[2])

  # Homogeneizamos datos
  datos12 <- cbind(datos11, 1)
  datos12.test <- cbind(datos11.test, 1)

  # Calculamos el vector de pesos
  recta_regresion06 <- Regress_Lin(datos12, etiquetas09)

  # Etiquetas dadas por el vector de pesos a los datos de test
  etiquetas_regresion.test <- sign(datos11.test[,2] -
                                   recta_regresion06[1]*datos11.test[,1] -
                                   recta_regresion06[2])

  # Calculamos el error medio
  error_out = error_out + 0.01*length(which(etiquetas_regresion.test != etiquetas09.test))
  }

cat("Porcentaje de Eout: ", error_out/1000)
```

```
## Porcentaje de Eout:  9.52
```

Como se ve el error fuera de la muestra es más elevado que el error dentro de la muestra. Es normal, que los valores salgan así, ya que es desperar que el error dentro de la muestra se menor al error fuera de la muestra, puesto que hay datos que no se tienen en cuenta a la hora de aprender.

```
# Después de crear una gráfica o iniciar un apartado paramos la ejecución 3 segundos
Sys.sleep(3)
```

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 ¿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

```
# Cuando se utilizan números aleatorios, es recomedable establecer una semilla, para
# que al repetir la generación de los números, siempre obtengamos los mismos
set.seed(3)
```

```r
# Gerenamos datos
datos13 <- simula_unif(10, 2, c(-10,10))

# Calculamos la recta
recta07 <- simula_recta(intervalo = c(-10,10))

# Obtenemos las etiquetas dadas por la muestra
etiquetas10 <- datos13[,2] - recta07[1]*datos13[,1] - recta07[2]

# Calculamos el vector de pesos dado por la regresion
recta_regresion07 <- Regress_Lin(datos13, etiquetas10)

recta_regresion07
```

```
## [1]  -0.7866537 -13.9696222
```

```r
iteraciones = replicate(1000, ajusta_PLA(datos13, etiquetas01, 1000,
                                         c(-0.7866537, -13.9696222, 1)))

cat(mean(iteraciones[3]),"iteraciones para converger.")
```

```
## 124 iteraciones para converger.
```

Es normal, que salga un número bajo de iteraciones, ya que está separando solo 10 datos.