Trabajo Prácticas 1 - Aprendizaje Automático

David Criado Ramón

24 de marzo de 2017

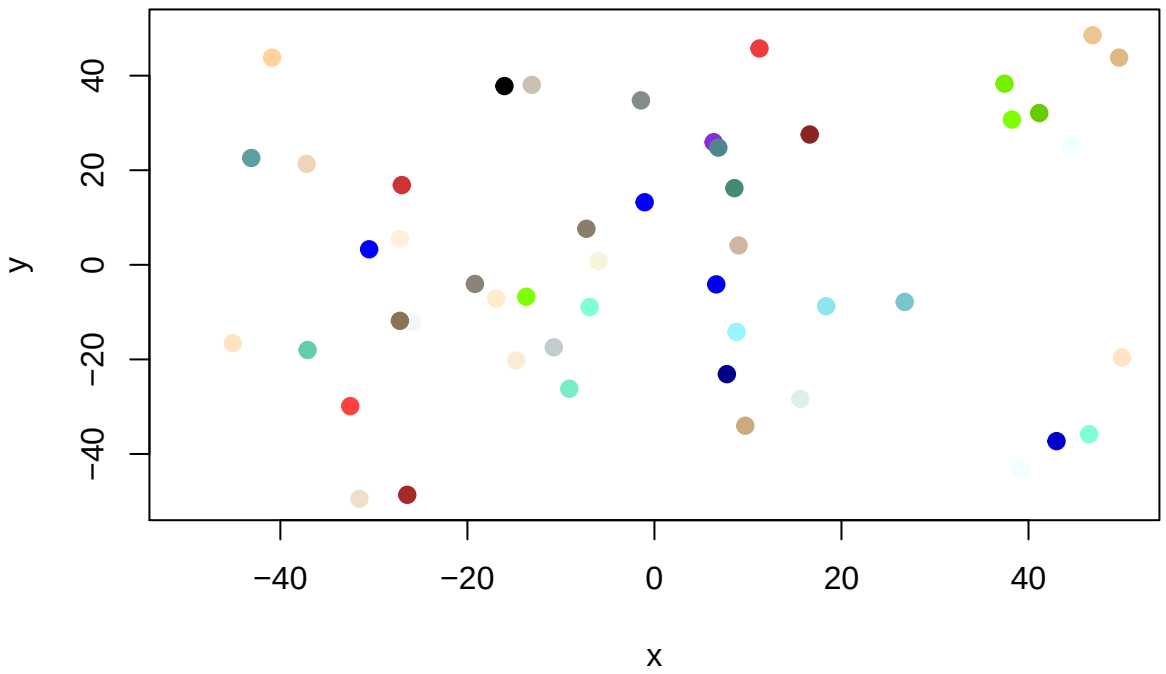
EJERCICIO SOBRE LA COMPLEJIDAD DE H Y EL RUIDO

1) Dibujar una gráfica con la nube de puntos de la salida correspondiente

a) Considere N = 50, dim = 2, rango = [-50,+50] con simula_unif(N, dims, rango)

Para realizarlo utilizamos la función simula_unif que nos ha sido proporcionada en paraTrabajo1. Dicha función nos va a devolver una matriz 50x2 en la que se encontrarán los 50 puntos 2D a representar. Las coordenadas x e y de los puntos se encontrarán en el intervalo [-50,+50] así que tendremos esos rangos en cuenta a la hora de delimitar el tamaño en la función que crea la gráfica (plot) que inicializaremos vacía con únicamente el título. Después, con la función points, añadiremos los puntos obtenidos a la gráfica con colores aleatorios (mediante la función colors()), con un ancho de 2 (lwd) y pintaremos dichos puntos como círculos sólidos (pch = 19).

```
# Función proporcionada en paraTrabajo1.R
simula_unif = function (N=2,dims=2, rango = c(0,1)){
 m = matrix(runif(N*dims, min=rango[1], max=rango[2]),
 nrow = N, ncol=dims, byrow=T)
 m
}

ejercicio1_1a = function(N = 50, dims = 2, rango = c(-50,50)) {
    # Cogemos los puntos de una distribución uniforme.
    puntos = simula_unif(N, dims, rango)
    # Creamos una gráfica vacía con el título del ejercicio.
    plot(rango, rango, xlab ="x", ylab ="y", type = "n", main = "Ejercicio 1-1a")
    # Ponemos los puntos con colores aleatorios.
    points(x = puntos, col = colors(), lwd = 2, pch = 19)
  }
  ejercicio1_1a()
```



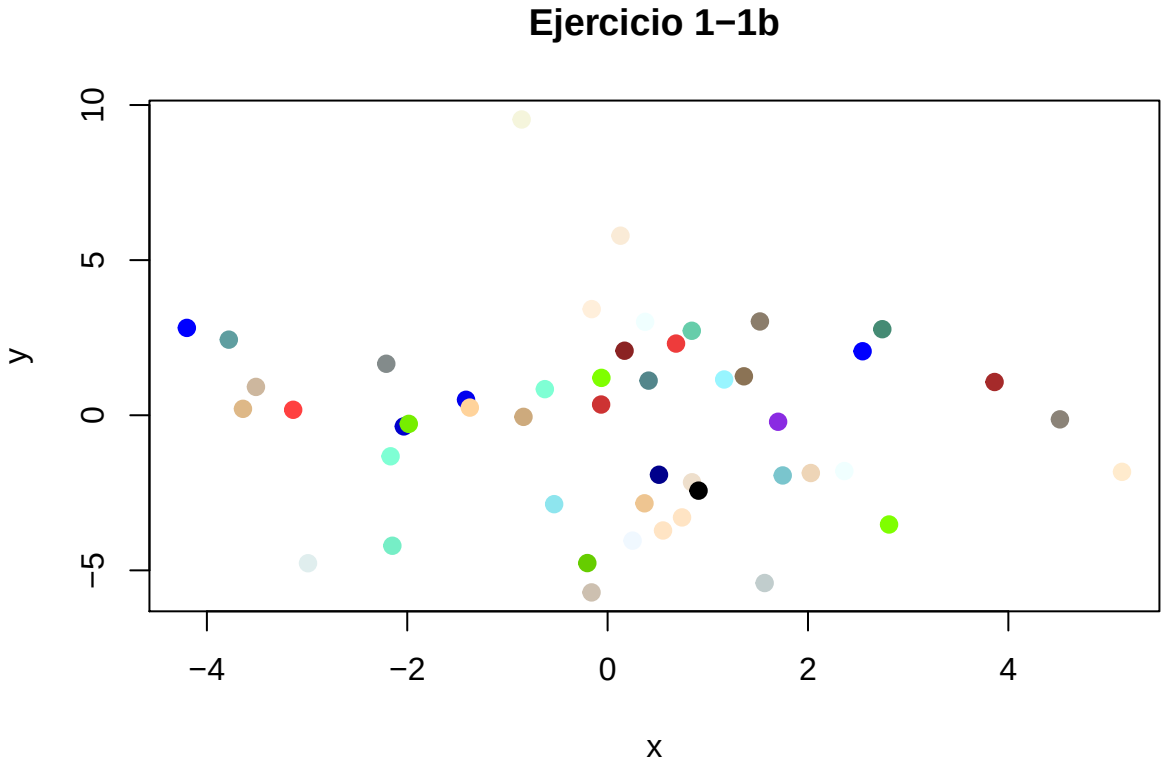
Podemos observar que en este caso la distribución, para la semilla puesta al inicio (45), ha resultado en una gráfica donde parece que hay unos pocos puntos más cerca del centro (0,0) que de los extremos, pero no es algo que deba ocurrir siempre puesto que al tratarse de una distribución uniforme todos los infintos puntos en el rango propuesto tendrían la misma probabilidad de haber salido escogidos.

b) Considere N = 50, dim = 2 y sigma = [5,7] con simula_gauss(N, dim, sigma)

En este caso actuamos de manera idéntica al caso anterior pero ahora utilizando simula_gauss (también proporcionada en Trabajo1.R) en vez de simula_unif. Cabe destacar que en este caso también utilizo range para obtener los valores mínimos y máximos de los puntos y delimitar el tamaño de la gráfica.

```
simula_gaus = function(N=2,dim=2,sigma){

  if (missing(sigma)) stop("Debe dar un vector de varianzas")
  sigma = sqrt(sigma)  # para la generación se usa sd, y no la varianza
  if(dim != length(sigma)) stop ("El numero de varianzas es distinto de la dimensiÃ³n")
  # 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
}
ejercicio1_1b = function(N = 50, dims = 2, sigma = c(5,7)) {
    # Cogemos los puntos aleatorios de la distribución gaussiana.
    puntos = simula_gaus(N, dims, sigma)
    # Creamos el gráfico vacío teniendo en cuenta el rango de los puntos.
    plot(range(puntos[,1]), range(puntos[,2]), xlab="x", ylab="y",
         type = "n", main = "Ejercicio 1-1b")
    # Pintamos los puntos con colores aleatorios.
```

```
    points(x = puntos, col = colors(), lwd = 2, pch = 19)
  }
ejercicio1_1b()
```



En este caso, como era de esperar en una distribución gaussiana de media 0, hay muchos más puntos cerca del (0,0) y menos conforme nos vamos alejando. Otra interesante propiedad de la distribución gaussiana es simétrica respecto de su media, pero en la pequeña muestra utilizada no se manifiesta con claridad dicha propiedad.

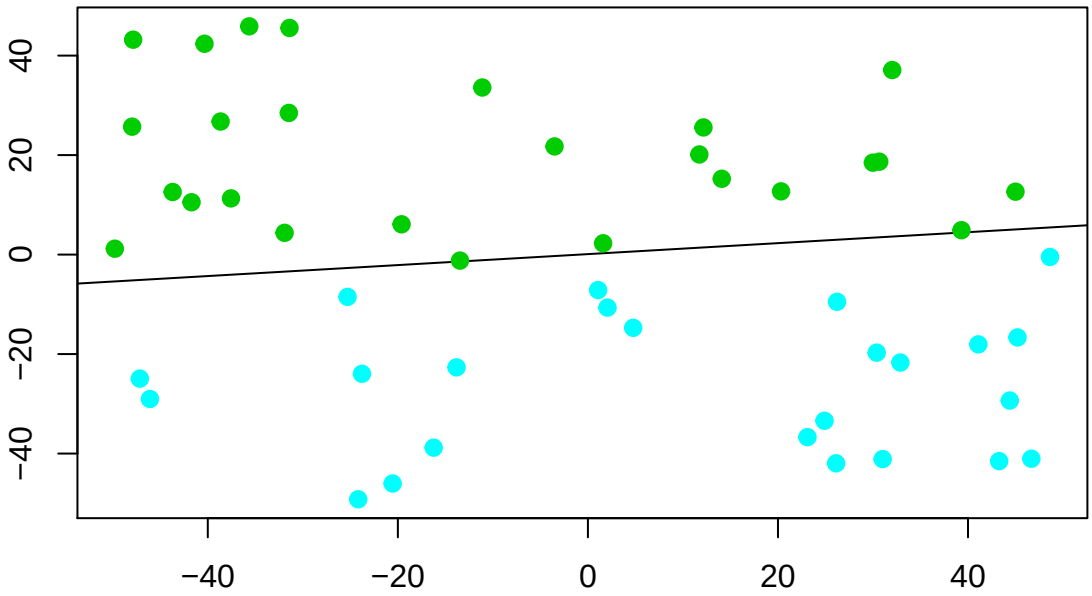
2) Con ayuda de la función simula_unif() generar una muestra de puntos 2D a los que vamos añadir una etiqueta usando el signo de la función f(x,y) = y - ax - b, es decir el signo de la distancia de cada punto a la recta simulada con simula_recta().

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

En este caso vamos a utilizar simula_recta, otra de las funciones proporcionadas en paraTrabajo1.R. Dicha función nos devuelve una recta aleatoria. Generando también con simula_unif 50 puntos aleatorios. Ambos son pintados en una gráfica. La recta es pintada usando abline y los puntos son clasificados antes de pintarse para darles colores distintos (el color es un entero calculado en función de la clasificación asociada al punto). La clasificación es realizada por la función propuesta en el ejercicio que escribo como una función en R. Utilizando apply con esa función sobre mi matriz de puntos obtengo un vector que me devuelve la clase (1 ó -1) asociada a dicho punto.

```
# Proporcionada en paraTrabajo1.R
simula_recta = function (intervalo = c(-1,1), visible=F){
   ptos = simula_unif(2,2,intervalo) # se generan 2 puntos
   a = (ptos[1,2] - ptos[2,2]) / (ptos[1,1]-ptos[2,1]) # calculo de la pendiente
```

```
   b = ptos[1,2]-a*ptos[1,1]  # calculo del punto de corte

   if (visible) {  # pinta la recta y los 2 puntos
       if (dev.cur()==1) # no esta abierto el dispositivo lo abre con plot
           plot(1, type="n", xlim=intervalo, ylim=intervalo)
       points(ptos,col=3)  #pinta en verde los puntos
       abline(b,a,col=3)   # y la recta
   }
   c(a,b) # devuelve el par pendiente y punto de corte
}

  # Apartado 2a
  # ------------------------------------------------
  # Función de clasificación: Signo de y - ax * b
  etiquetar = function(punto, recta) {
    sign(punto[2] - recta[1] * punto[1] - recta[2])
  }
  ejercicio1_2a = function(N = 50, dims = 2, rango = c(-50,50)){
    # Generamos los puntos aleatorios siguiendo una distribución uniforme
    puntos = simula_unif(N, dims, rango)
    # Generamos una gráfica vacía
    plot(range(puntos[,1]), range(puntos[,2]), type = "n", xlab = "", ylab = "",
         main = "Ejercicio 1-2a")
    # Generamos una recta aleatoria
    recta = simula_recta(intervalo = c(0,1))

    # Pintamos la recta en la gráfica
    abline(recta[2],recta[1])

    # Clasificamos los puntos basándonos en si se encuentran a un lado u otro de la recta
    etiquetas = apply(X = puntos, MARGIN = 1, FUN = etiquetar, recta = recta)
    # Pintamos los puntos en la gráfica
    points(x = puntos, col = 32 + 19 * etiquetas, lwd = 2, pch = 19)
    # Devolvemos los puntos, la recta y el etiquetado para otros ejercicios
    list(puntos = puntos, recta = recta, etiquetas = etiquetas)
  }
  ejer1_2a = ejercicio1_2a()
```



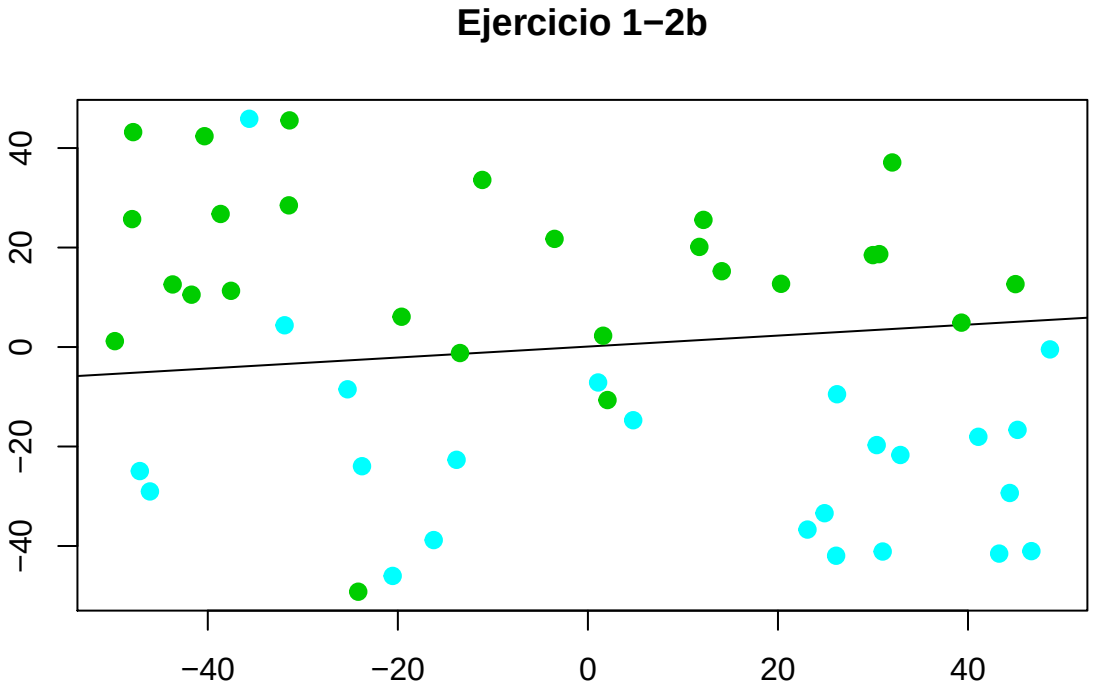
Como podemos observar todos los puntos han quedado perfectamente clasificados respecto de la recta, el único caso que podría haber ocurrido para que no saliese bien el etiquetado es que alguno de los puntos hubiese estado contenido en la recta, por lo que la diferencia sería 0 y al aplicarle sign se quedaría en la clase de etiqueta 0 en vez de ir a la 1 o la -1.

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

Para modificar un 10 % de etiquetas positivas y el mismo porcentaje de negativas, primero hemos de determinar que puntos son positivos y cuales negativos. Para ello utilizo which, que me devolverá un vector con los índices en los que la condición que indico se evalúa como verdadero siendo la condición que pertenezca a la clase 1 para los positivos y a la clase -1 para los negativos. Después utilizo la función sample para obtener una muestra de exactamente el 10 % sobre mi vector de índices. Para terminar la modificación, la clasificación de la muestra que he seleccionado de índices positivos es modificada a -1 y de la muestra negativa es modificada a 1. Así pues, con la nueva clasificación, volvemos a pintar la gráfica de los puntos con su clase y la recta de la misma manera que se hizo en el apartado anterior.

```
mete_ruido = function(etiquetas, porcentaje) {
    # Apuntamos los índice cuales son positivos y cuales son negativos
    positivos = which(etiquetas == 1)
    negativos = which(etiquetas == -1)
    # Cogemos una muestra del porcentaje de cada uno
    muestra_positivos = sample(positivos, length(positivos) * porcentaje)
    muestra_negativos = sample(negativos, length(negativos) * porcentaje)
    # Alteramos la etiqueta
    etiquetas[muestra_positivos] = -1
    etiquetas[muestra_negativos] =  1
    etiquetas
}
ejercicio1_2b = function(puntos = ejer1_2a[[1]], recta = ejer1_2a[[2]],
                         etiquetas = ejer1_2a[[3]]) {
    # Metemos el ruido del 10 % en cada clase
```

```r
  etiquetas = mete_ruido(etiquetas, 0.1)
  # Dibujamos la nueva gráfica
  plot(range(puntos[,1]), range(puntos[,2]), type = "n", xlab = "", ylab = "",
       main = "Ejercicio 1-2b")
  abline(recta[2],recta[1])
  points(x = puntos, col = 32 + 19 * etiquetas, lwd = 2, pch = 19)
  # Devolvemos el nuevo etiquetado
  etiquetas
}
ejer1_2b = ejercicio1_2b()
```



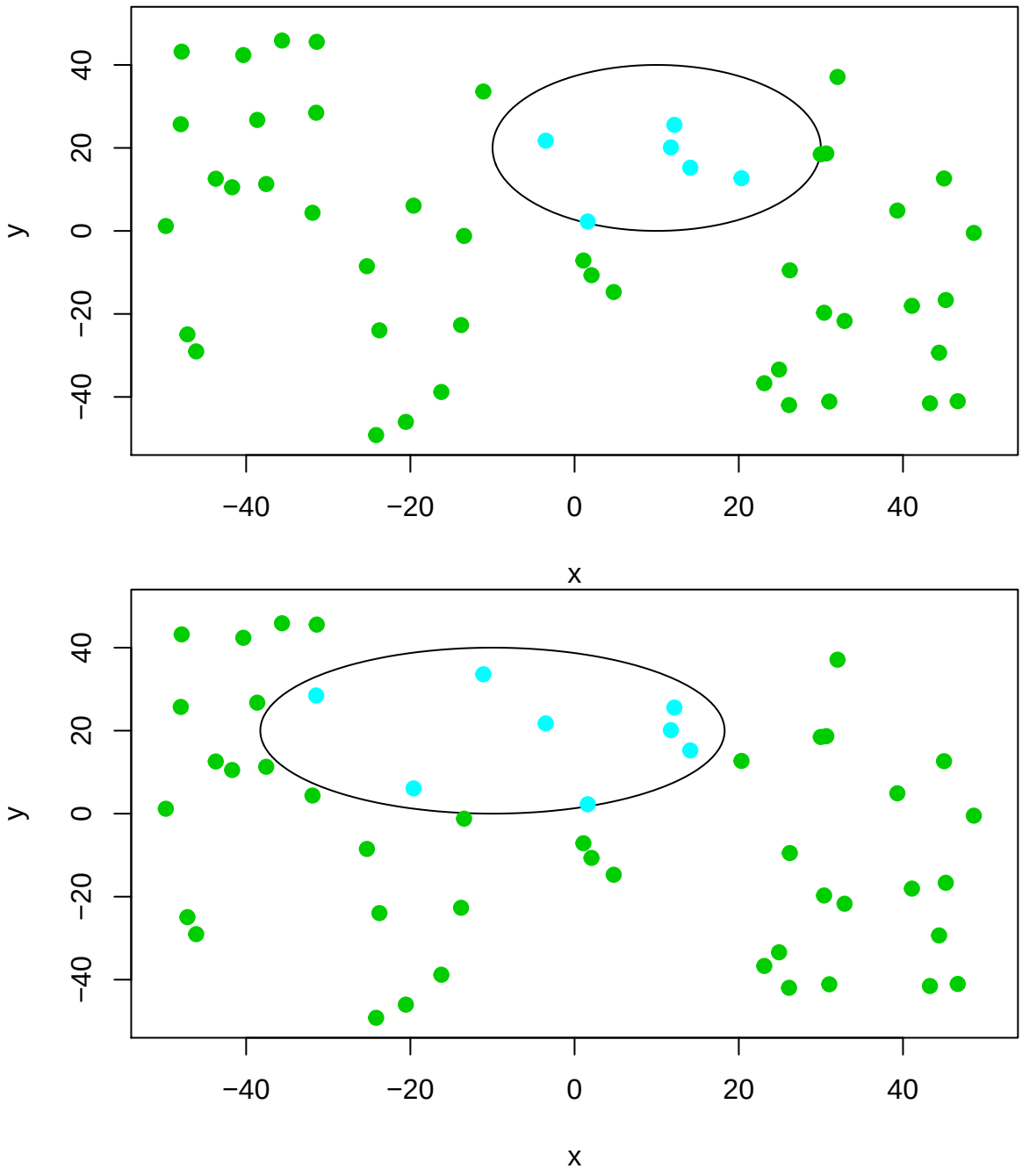
Podemos observar como un par de puntos de cada clase ha cambiado y ahora es evidente que ambas clases no pueden ser separadas mediante una recta.

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

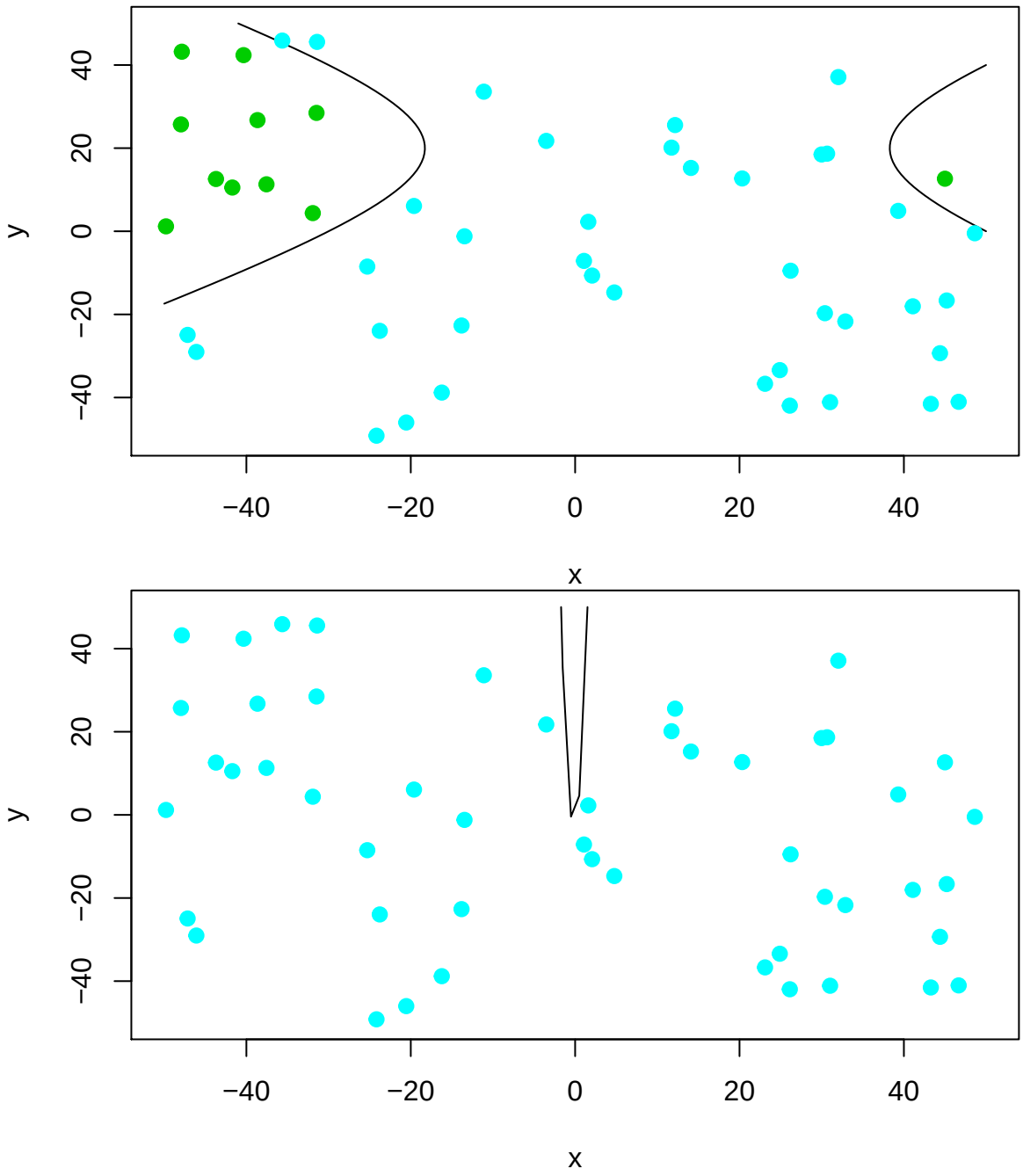
- $f(x, y) = (x - 10)^2 + (y - 20)^2 - 400$
- $f(x, y) = 0,5(x + 10)^2 + (y - 20)^2 - 400$
- $f(x, y) = 0,5(x - 10)^2 - (y + 20)^2 - 400$
- $f(x, y) = y - 20x^2 - 5x + 3$

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 utilizar una funciones más complejas que la dada por una función lineal? Explicar el razonamiento.

```
# Proporcionada en paraTrabajo1.R
pintar_frontera = function(f,rango=c(-50,50)) {
   x=y=seq(rango[1],rango[2],length.out = 100)
   z = outer(x,y,FUN=f)
  if (dev.cur()==1) # no esta abierto el dispositivo lo abre con plot
          plot(1, type="n", xlim=rango, ylim=rango)
   contour(x,y,z, levels = 0, drawlabels = FALSE,xlim =rango, ylim=rango, xlab = "x", ylab = "y")
}

f3a = function(x, y) {
  (x-10) * (x-10) + (y-20) * (y-20) - 400
}

f3b = function(x, y) {
  0.5 * (x + 10) * (x + 10) + (y - 20) * (y-20) - 400
}

f3c = function(x, y) {
  0.5 * (x-10) * (x-10) - (y-20) * (y-20) - 400
}

f3d= function(x, y) {
  y - 20 * x * x - 5 * x + 3
}

ejercicio1_3 = function(puntos = ejer1_2a[[1]]) {
  pintar_frontera(f3a)
  etiquetas = sign(mapply(f3a, puntos[,1], puntos[,2]))
  points(x = puntos, col = 32 + 19 * etiquetas, lwd = 2, pch = 19)
  pintar_frontera(f3b)
  etiquetas = sign(mapply(f3b, puntos[,1], puntos[,2]))
  points(x = puntos, col = 32 + 19 * etiquetas, lwd = 2, pch = 19)
  pintar_frontera(f3c)
  etiquetas = sign(mapply(f3c, puntos[,1], puntos[,2]))
  points(x = puntos, col = 32 + 19 * etiquetas, lwd = 2, pch = 19)
  pintar_frontera(f3d)
  etiquetas =sign(mapply(f3d, puntos[,1], puntos[,2]))
  points(x = puntos, col = 32 + 19 * etiquetas, lwd = 2, pch = 19)
}

ejercicio1_3();
```


y
40
20
0
−20
−40
−40
−20
0
20
40
x

y
40
20
0
−20
−40
−40
−20
0
20
40
x



Vemos claramente que el etiquetado basado en las distancias con la recta clasifica mucho mejor las muestras (siguiendo el etiquetado de 2b) incluso añadiendo el 10 % de ruido si intentamos utilizar las funciones propuestas para clasificar esas muestra. Aunque, en algunas ocasiones, nos pueda interesar el uso de las funciones más complejas, vendrá determinado por el problema si será o no una mejor opción utilizarlas. No nos interesa obtener una función demasiado compleja que se aproxime muy bien a la muestra de entrenamiento, puesto que puede aprender ciertas características extrañas que se dan en la muestra y ser mala para resolver el problema fuera de la muestra. Si ahora consideramos las nuevas funciones cuadráticas para el etiquetado como vemos en las gráficas, es fácil darse cuenta de que no vamos a ser capaces de pintar una recta que clasifica mejor los puntos que las funciones cuadráticas dadas.

EJERCICIO SOBRE EL ALGORITMO PERCEPTRON

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

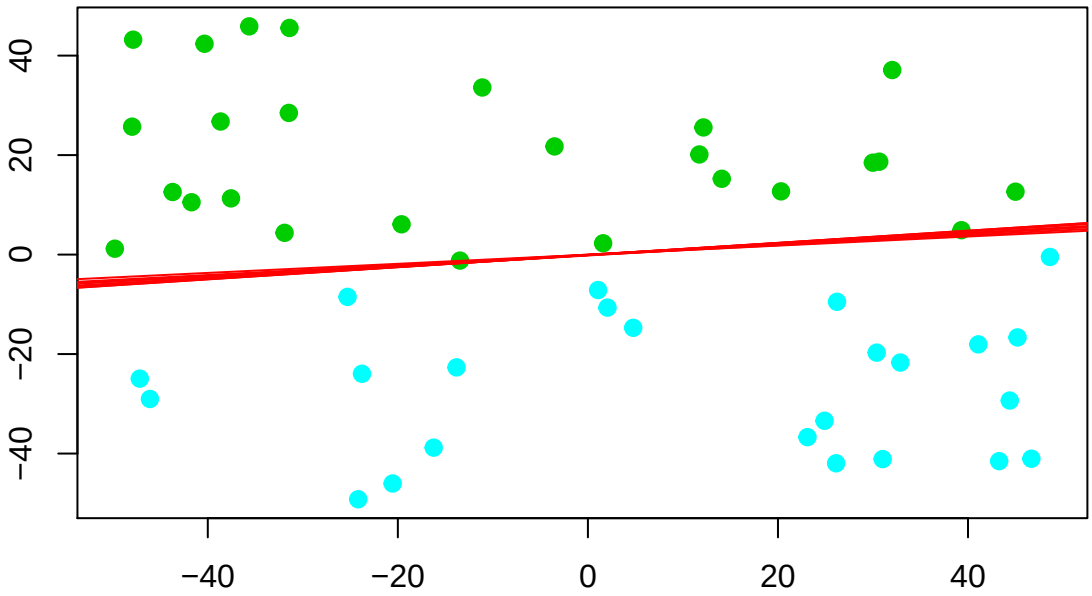
En este ejercicio, simplemente implementamos el algoritmo perceptron visto en teoría, utilizamos una variable cambio que nos permite indicar si el algoritmo no ha sido capaz de producir más cambios sobre larecta solución o hemos alcanzado el número máximo de iteraciones. Además, iteraremos sobre la muestra en orden aleatorio en vez de manera secuencial. Devolveremos junto con el ajuste de los pesos w, el número de iteraciones.

```
ajusta_PLA = function(datos, label, max_iter, vini) {
  datos = cbind(1, datos) # Añadimos la columna de 1
  i = 0 # contador para el número de iteraciones
  cambio = T # indica si se ha producido algún cambio o se debe parar
  w = vini # vector de pesos
  while (i < max_iter & cambio) {
    cambio = F # puesto que hemos empezado la iteración no se ha podido producir
               # ningún cambio
    # recorremos todos los datos
    for (x in sample(1:nrow(datos))) {
      # Aplicamos signo
      signo = sign(sum(datos[x,] * w))
      if (label[x] != signo) {
        # Si no coinciden el signo y la etiqueta cambiamos el valor del vector de pesos
        # wnew = wold + yi * xi
        w = w + datos[x,] * label[x]
        # indicamos que se ha producido un cambio
        cambio = T
      }
    }
    i = i + 1 # aumentamos el contador de iteraciones
  }
  # parámetros del hiperplano (recta en este caso) y número de iteraciones usadas
  list(w=w, iteraciones =i, vini = vini)
}

# Calcula una recta basada en el vector de pesos w y pondemos pintarla en un plot
# previamente creado
calcula_recta = function(w, pintar = F) {
  a = -w[2]/w[3]
  b = -w[1]/w[3]
  abline(b,a, col = "red")
  list(a = a, b = b)
}
```

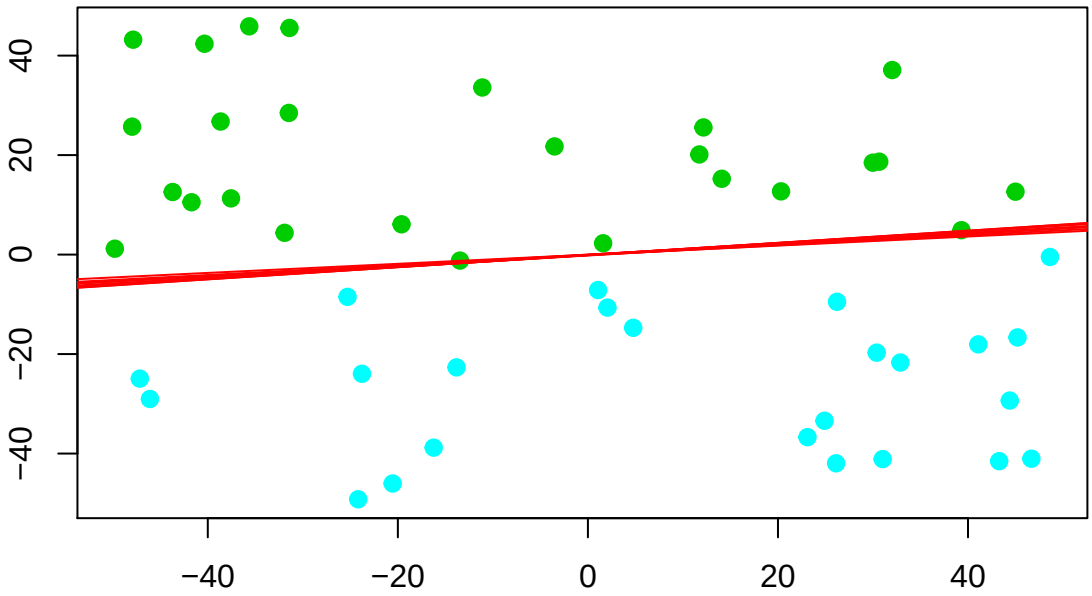
2) Ejecutar el algoritmo PLA con los datos simulados en el apartado 2a de la sección 1. Inicializar el algoritmo con: a) el vector 0, y b) con vectores de números aleatorios en [0,1] (10 veces). Anotar el número medio de iteraciones en ambos para converger. Valorar el resultado relacionando el punto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

En el caso inicializado a 0 el algoritmo variará debido a que recorre los vectores de características en un orden aleatorio. Para facilitar la compresión muestro tablas (dónde *wf* es el vector de pesos final, *wi* es el vector de pesos inicial e *iters* el número de iteraciones utilizadas por el algoritmo) y también pinto una única gráfica (una para el caso inicializado a 0 y otra para el caso aleatorio), en la que pinto todas las rectas obtenidas.

```
ejercicio2_2 = function(puntos = ejer1_2a$puntos, etiquetas = ejer1_2a$etiquetas) {
  # Caso A: Inicializando el vector de pesos a 0
  ajuste = t(replicate(10, as.numeric(unlist(
    ajusta_PLA(datos = puntos, label = etiquetas, vini = c(0,0,0),
               max_iter = 1000)))))

  # # Caso B: Inicio de vectores de pesos aleatorios
  ajusteAleatorio = t(replicate(10, as.numeric(unlist(
    ajusta_PLA(datos = puntos, label = etiquetas, vini = runif(3),
               max_iter = 1000)))))

  # Asignamos nombres a las columnas para que quede claro al hacer print
  # wf hace referencia a vector de pesos final
  # wi hacer referencia a vector de pesos inicial
  colnames(ajuste) = c("wf1", "wf2", "wf3", "iters", "wi1", "wi2", "wi3")
  colnames(ajusteAleatorio) = c("wf1", "wf2", "wf3", "iters", "wi1", "wi2", "wi3")

  # Pintamos una gráfica y mostramos la tabla para caso [0,0,0]
  plot(x = puntos, col = 32 + 19 * etiquetas, lwd = 2, pch = 19,
    type = "p", xlab = "", ylab = "", main = "PLA inicializado a 0.")
  apply(X = ajuste, MARGIN = 1, FUN = calcula_recta, pintar = T)
  print(ajuste)
  cat("De media en el caso inicializado a 0 hemos usado ", mean(ajuste[,4]), " iteraciones.\n")

  # Pintamos una gráfica y mostramos la tabla para caso aleatorio
  plot(x = puntos, col = 32 + 19 * etiquetas, lwd = 2, pch = 19,
    type = "p", xlab = "", ylab = "", main = "PLA inicializado aleatoriamente.")
  apply(X = ajuste, MARGIN = 1, FUN = calcula_recta, pintar = T)
  print(ajusteAleatorio)
  cat("De media en el caso aleatorio se han utilizado", mean(ajusteAleatorio[,4]), "iteraciones.\n")
}
ejercicio2_2()
```




```
##       wf1        wf2       wf3 iters wi1 wi2 wi3
##  [1,]   0  -7.544063  70.31231     2   0   0   0
##  [2,]   5 -11.281844  95.31346     4   0   0   0
##  [3,]   8 -10.170806  95.29552     4   0   0   0
##  [4,]  12 -17.465020 143.37774    10   0   0   0
##  [5,]  11 -14.876998 121.54580     7   0   0   0
##  [6,]   6  -9.337129  87.52860     2   0   0   0
##  [7,]   4 -13.022069 142.44608     7   0   0   0
##  [8,]  11  -9.859473  98.76226     7   0   0   0
##  [9,]  28 -17.073555 152.19999    18   0   0   0
## [10,]  13 -13.314295 111.85833     9   0   0   0
## De media en el caso inicializado a 0 hemos usado  7  iteraciones.
```



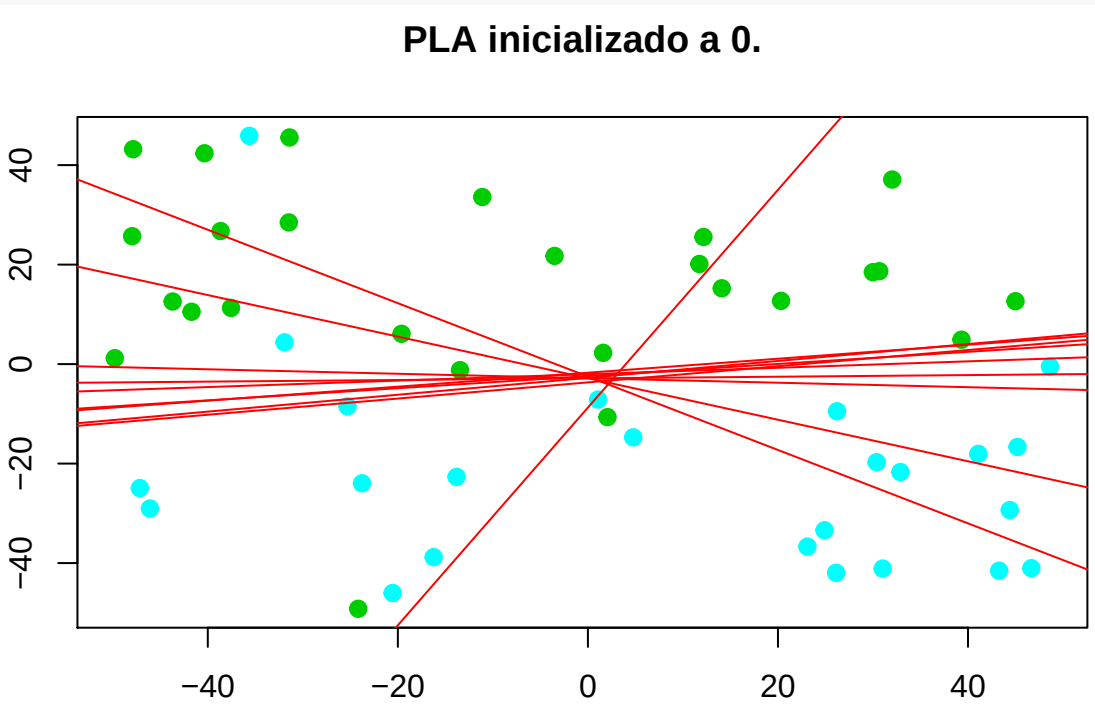

```
##             wf1        wf2       wf3 iters        wi1        wi2       wi3
##  [1,]  3.271252  -4.919702  49.39069     2 0.27125238 0.62882168 0.1365841
##  [2,]  6.022278  -9.411738  98.06117     4 0.02227833 0.27940507 0.9013155
##  [3,] 11.631299 -14.554706 122.97745     8 0.63129914 0.25141711 0.1644632
##  [4,]  4.920357 -10.346480 101.56961     6 0.92035713 0.07499839 0.1904618
##  [5,]  1.889743  -9.320633  88.94450     3 0.88974272 0.05504475 0.6348994
##  [6,]  6.368832  -4.790930  53.33063     2 0.36883202 0.16813666 0.4471289
##  [7,] -0.405776  -7.368608  71.38621     3 0.59422403 0.02347221 0.9122863
##  [8,]  5.074778  -9.947276  90.28701     4 0.07477769 0.72110019 0.4293400
##  [9,]  3.261501  -8.673867  75.80037     3 0.26150144 0.16909428 0.3950311
## [10,] 22.564403 -14.725001 155.05102    18 0.56440285 0.26461592 0.2837331
## De media en el caso aleatorio se han utilizado 5.3 iteraciones.
```

Podemos observar que, aunque todas las rectas separan las etiquetas bien, ninguna de las rectas es exactamente la misma. Aunque la media obtenida favorezca el inicio con vector de pesos inicializados a 0, viendo las muestras de las tablas no puedo concluir que el resultado sea significativo y que inicializarlo a 0 hace que converja antes que inicializándolo aleatoriamente. En ambos, hay casos en los que converje en pocas iteraciones (2 ó 3) y otros casos en los que tarda bastantes iteraciones para la complejidad del problema (18)

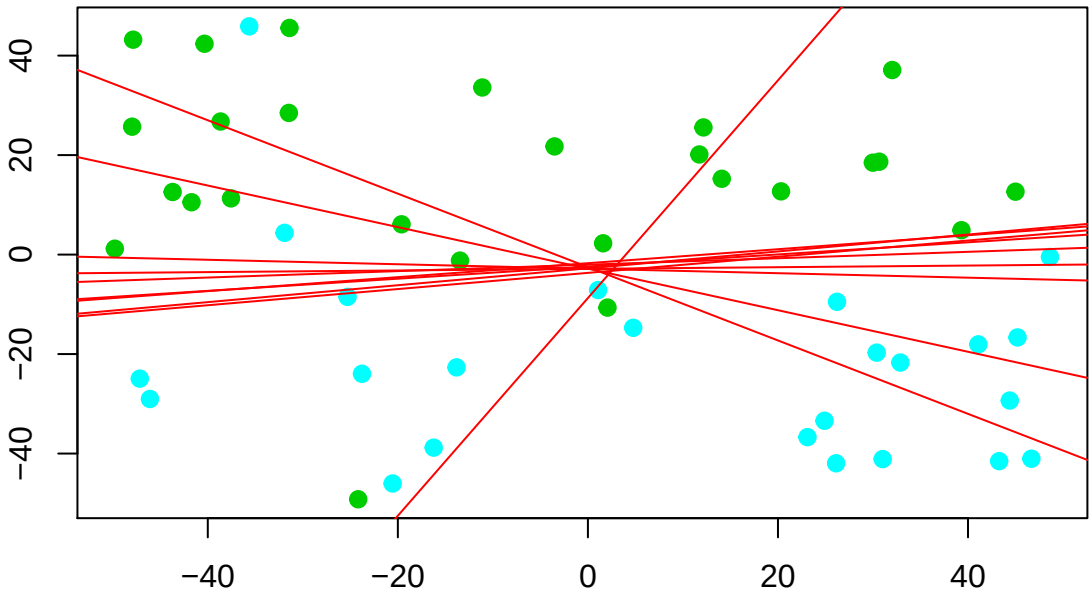
Ejercicio 2 - Apartado 3: Hacer los mismo que antes usando ahora los datos del apartado 2b de la sección 1. ¿Observa algún comportamiento diferente? En caso afirmativo diga cual y las razones para que ello ocurra.

A la hora de escribir código no debemos de añadir nada nuevo sólo llamar a la función creada en el ejercicio anterior pero ahora con las etiquetas que tenian ruido (ejer1_2b).

```
ejercicio2_2(puntos = ejer1_2a$puntos, etiquetas = ejer1_2b)
```




```
##       wf1         wf2      wf3 iters wi1 wi2 wi3
##  [1,] 115   1.8220705 40.46480  1000   0   0   0
##  [2,] 125  18.4609472 44.19640  1000   0   0   0
##  [3,]  95  -0.5441185 33.14030  1000   0   0   0
##  [4,] 127  -7.8000237 45.93463  1000   0   0   0
##  [5,] 105  -8.5211665 60.65675  1000   0   0   0
##  [6,] 113 -28.3010442 12.93329  1000   0   0   0
##  [7,] 115  -3.6597974 56.62823  1000   0   0   0
##  [8,] 110  32.3197995 43.81043  1000   0   0   0
##  [9,] 116  -5.1309476 31.55316  1000   0   0   0
## [10,] 127  -6.4292732 52.71467  1000   0   0   0
## De media en el caso inicializado a 0 hemos usado  1000  iteraciones.
```




```
##             wf1       wf2      wf3 iters         wi1       wi2        wi3
##  [1,]  94.31022  29.84855 46.06846  1000 0.310217649 0.9892447 0.02581326
##  [2,] 121.00471 -11.56227 37.81279  1000 0.004707521 0.5949787 0.79873845
##  [3,]  99.53700  39.89316 28.95278  1000 0.536996573 0.4715879 0.38652815
##  [4,] 109.51059 -20.12647 12.58630  1000 0.510586092 0.6342598 0.58439187
##  [5,] 120.60747  25.12566 72.06328  1000 0.607472484 0.4510046 0.32326139
##  [6,] 103.94065 -25.30003 64.07959  1000 0.940651824 0.1808976 0.93579270
##  [7,] 121.98879 -19.19978 81.30475  1000 0.988791194 0.9701187 0.03753606
##  [8,] 118.44803 -24.65129 49.14309  1000 0.448029505 0.7631446 0.58790715
##  [9,] 145.38710 -12.19907 31.67994  1000 0.387101737 0.3003487 0.63353983
## [10,] 137.18234  25.89039 60.08516  1000 0.182337062 0.8885039 0.44191516
## De media en el caso aleatorio se han utilizado 1000 iteraciones.
```

Podemos observar que en todos los casos el PLA utiliza todas las iteraciones propuestas sin llegar a converger. Esto se debe a que el problema inicial no es linealmente separable y nuestro hiperplano (la recta), no va a ser capaz de separar todos los datos del problema a un lado o al otro. El hecho de empezar con diferentes vectores de pesos hace que en la última iteración usada la recta obtenida sea distinta, obteniendo mejores y peores resultados en las distintas rectas, pero dichos resultados sólo son fruto de la aleatoriedad impuesta por el número finito de iteraciones y por el recorrido aleatorio del vector de características usado en la implementación y no por el hecho de empezar con el vector de pesos 0 u otro cualquiera.

EJERCICIO SOBRE REGRESIÓN LINEAL

En la web del curso se encuentran disponibles la descripción

1) Leemos datos:

- 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

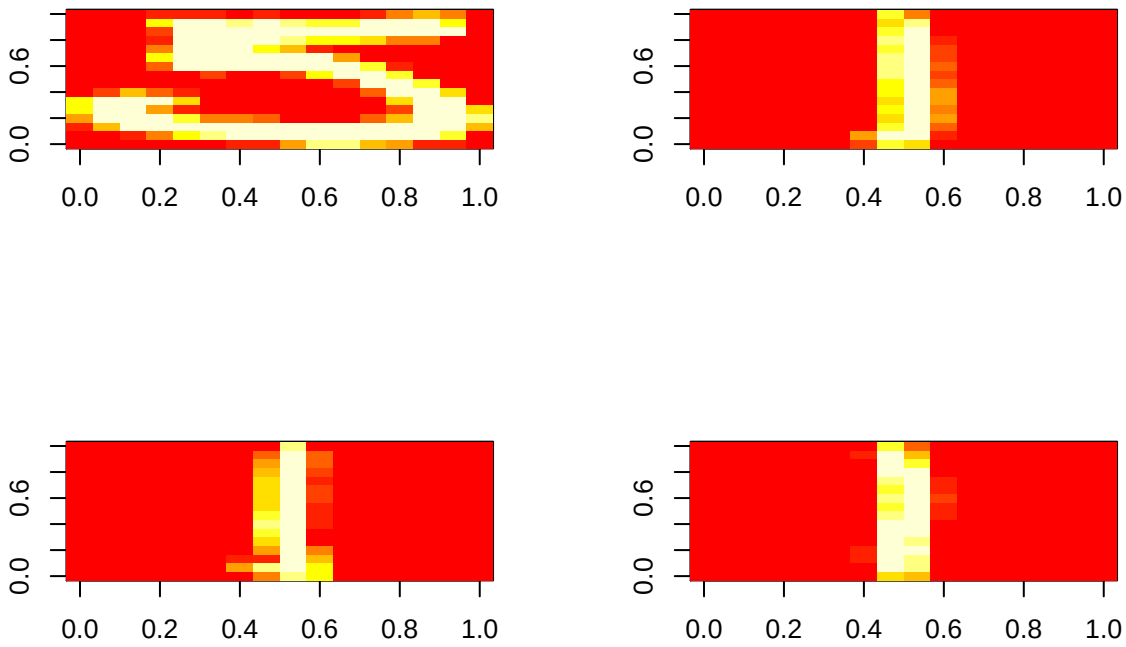
dentro de su código y visualize las imágenes (utilizando paraTrabajo1.R). Seleccione sólo las instancias de los números 1 y 5. Guárdelas como matrices de 16x16.
- También está disponible el fichero Zip.test que deberemos usar más adelante.

En el archivo en cada fila aparecen los 256 numeros que corresponden a los datos de la imagen en escala de grises y de manera similar a como se hace en el ejemplo de paraTrabajo1.R leo Zip.test (*Nota: he quitado del archivog Zip.test la última línea que causaba problemas al estar incompleta.*)

```
setwd("./datos")
# Función que lee el archivo, seleccionando instancias de 1 y 5
lectura = function(file) {
  # Leemos el archivo
  digit <- read.table(file, quote="\"", comment.char="", stringsAsFactors=FALSE)
  # Seleccionamos instancias de 1 y 5
  digit15 = digit[digit$V1 == 1 | digit$V1 ==5,]
  # Cogemos sus etiquetas
  etiquetas = digit15[,1]
  # Pasamos las etiquetas 5 a -1
  etiquetas[etiquetas == 5] = -1
  # Sacamos el tamaño del conjunto seleccionado
  n = nrow(digit15)
  # Sacamos la matriz de grises
  grises = array(unlist(subset(digit15,select=-V1)),c(n,16,16))

  # Devolvemos grises y el etiquetado original
  list (grises = grises, etiquetas = etiquetas)
}
zip_training = lectura("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
```

```
zip_test = lectura("zip.test")

par(mfrow=c(2,2))
for(i in 1:4){
  imagen = zip_training$grises[i,,16:1] # se rota para verlo bien
  image(z=imagen)
}
```




```
par(mfrow = c(1,1))
```

Podemos observar una instancia de un 5 y tres de las de uno con diferentes colores asociados a la intensidad de la escala de blancos. Las etiquetas de 5 son pasadas a -1, y las de 1 se quedan en 1 para luego procesarlas.

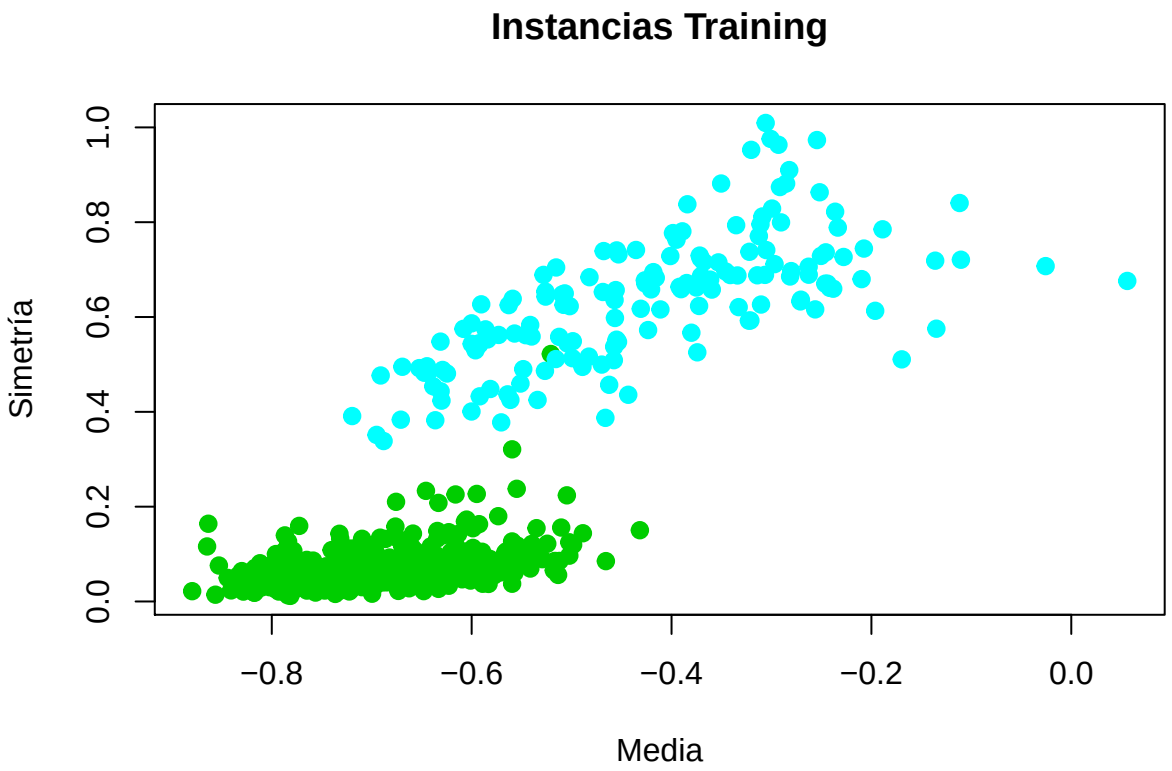
2) De cada matriz de números (imagen) vamos a extraer dos características: a) su valor medio; y b) su grado de simetría vertical

- Para calcular el grado de simetría vertical haremos lo siguiente: a) calcularemos una nueva matriz invirtiendo el orden de las columnas; b) calcularemos 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 este valor más asimétrica será la imagen.
- Representar en los ejes { X = Intesidad Promedio, Y Simetría las instancias seleccionadas de 1's y 5's}

Otra vez volvemos a utilizar el código proporcionado por paraTrabajo1.R (modificándolo para encontrar) y como hemos hecho en otros ejercicios mostramos ambos puntos.

```
fsimetria <- function(A){
  A = abs(A-A[,ncol(A):1])
  mean(A)
}

calcula_media_simetria = function(zip = zip_training, pintar = T) {
  # Calculamos la media
  media = apply(X=zip$grises, MARGIN = 1, FUN = mean)
  # Calculamos la simetría
  simetria = apply(X=zip$grises, MARGIN = 1, FUN = fsimetria)
  # Pintamos la gráfica con los puntos
  if (pintar) {
     plot(x = media, y = simetria, type = "p", xlab = "Media",
     ylab = "Simetría", main = "Instancias Training", lwd = 2, pch = 19,
     col = 32 + 19 * zip$etiquetas)
  }
```

```
  # Devolvemos la media y la simetría para el siguiente apartado
  list(media = media, simetria = simetria)
}
caracteristicas_training = calcula_media_simetria()
```



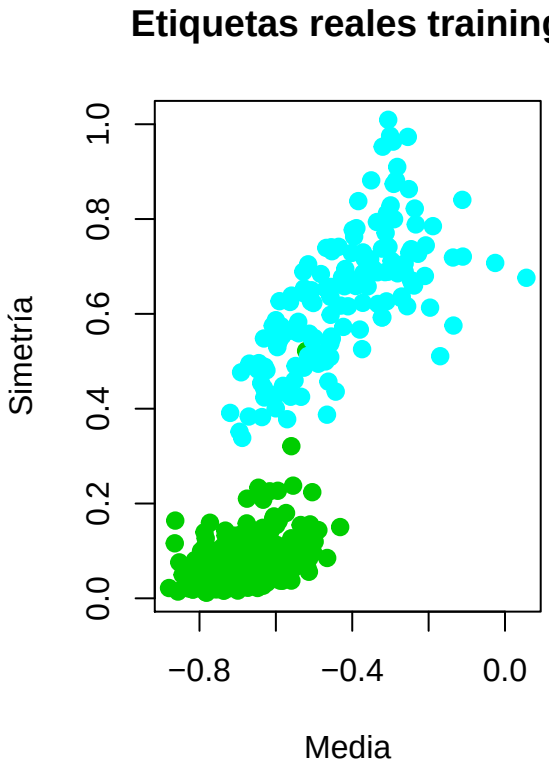
Como podemos observar el problema sería linealmente separable sino fuese por la excepción de un punto verde que crea ruido en la región azul. A partir de ahora consideramos que los puntos verdes representan las instancias de 1 (con etiquetas 1) y las instancias de 5 (con etiquetas -1) son puntos de color cian.

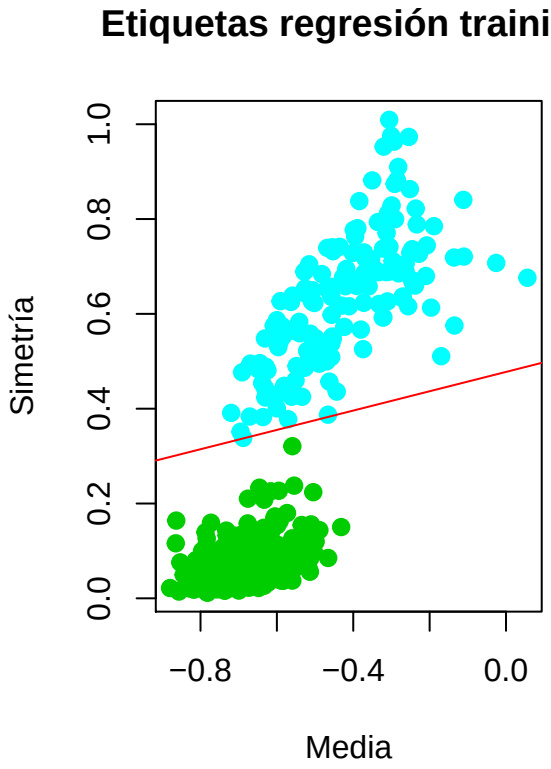
3) Ajustar un modelo de regresión lineal usando la transformación SVD sobre los datos de (Intensidad promedio, Simetría) y pintar la solución obtenida junto con los datos usados en el ajuste. Las etiquetas serán {-1,1}. Valorar la bondad del resultado usando Ein y Eout (usar Zip.test). (usar Regress_Lin(datos,label)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

```
#### 3-3
Regress_Lin = function(datos, label) {
  # Añadimos la columna 1
  x = cbind(1,datos)
  # Aplicamos svd
  svd = svd(x)
  # Sacamos la matriz ortogonal V
  V = svd$v
  # Sacamos la matriz diagonal d
  d = svd$d
  # Calculamos la pseudoinversa como la matriz diagonal de d que
  # si un dato es mayor que el epsilon de la máquina (escogido por mí como 0.0001)
  # invierte el valor
```

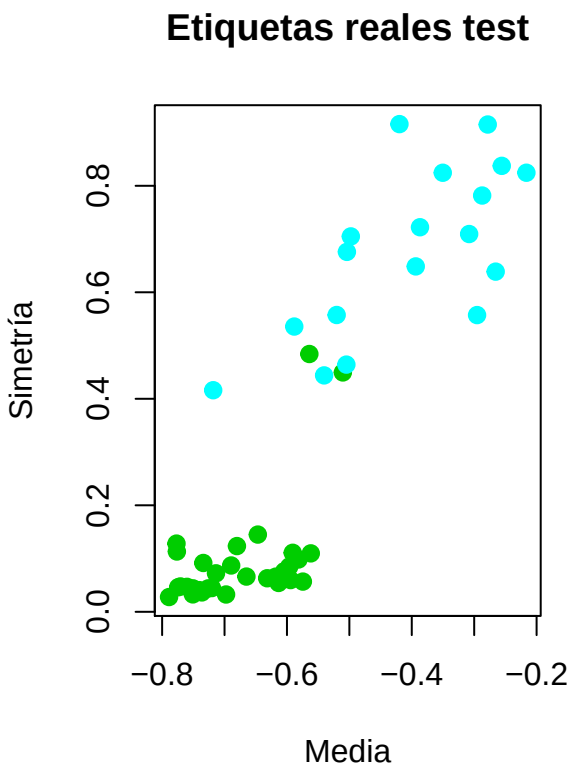
```
  D = diag(ifelse(d > 0.0001, 1/d, d))

  # Calculamos el vector de pesos (Importante D^2)
  w = V %*% D^2 %*% t(V) %*% t(x) %*% label
}

etiquetar_regresion = function(x, w) {
  x = c(1,x) # Añadimos 1 al principio del vector de características
  # La etiqueta viene dada por el signo de la sumatoria de xi * wi
  sign(sum(x*w))
}
apartado3 = function(training = caracteristicas_training) {
  # Preparamos el vector de características de training
  datos_training = cbind(training$media,training$simetria)
  # Aplicamos regresión
  w_training = Regress_Lin(datos_training, zip_training$etiquetas)
  # Usamos la regresión para etiquetar
  etiq_regr_training = apply(datos_training, 1, etiquetar_regresion, w_training)

  # Pintamos etiquetado original a izquierda y regresión a derecha (TRAINING)
  par(mfrow = c(1,2))
  plot(datos_training, xlab = "Media", ylab = "Simetría", lwd = 2, pch = 19,
     main = "Etiquetas reales training", col = 32 + 19 * zip_training$etiquetas)
  cat("\n")
  plot(datos_training, xlab = "Media", ylab = "Simetría", lwd = 2, pch = 19,
     main = "Etiquetas regresión training", col = 32 + 19 * etiq_regr_training)
  calcula_recta(w_training, pintar = T)

  # Calculamos y mostramos ein
  ein = sum(etiq_regr_training != zip_training$etiquetas)
  cat("El error dentro de la muestra de entrenamiento (Ein) es del ",
      ein/nrow(datos_training) * 100, "%\n")

  # Calculamos las características de TEST
  caracteristicas_test = calcula_media_simetria(zip = zip_test, pintar = F)
  # Las juntamos en una matriz
  datos_test = cbind(caracteristicas_test$media, caracteristicas_test$simetria)
  # Calculamos el etiquetado con respecto al ajuste de training
  etiq_regr_test = apply(datos_test, 1, etiquetar_regresion, w_training)

  # Pintamos etiquetado original a izquierda y regresión a derecha (TEST)
  plot(datos_test , xlab = "Media", ylab = "Simetría", lwd = 2, pch = 19,
      col = 32 + 19 * zip_test$etiquetas, main = "Etiquetas reales test")
  cat("\n")
  plot(datos_test , xlab = "Media", ylab = "Simetría", lwd = 2, pch = 19,
      col = 32 + 19 * etiq_regr_test, main = "Etiquetas regresión test")
  calcula_recta(w_training, pintar = T)
  eout = sum(etiq_regr_test != zip_test$etiquetas)

  cat("El error dentro de la muestra de entranmiento (en test) (Eout) es del ",
      eout/nrow(datos_test) * 100, "%\n")
  par(mfrow=c(1,1))
}
```

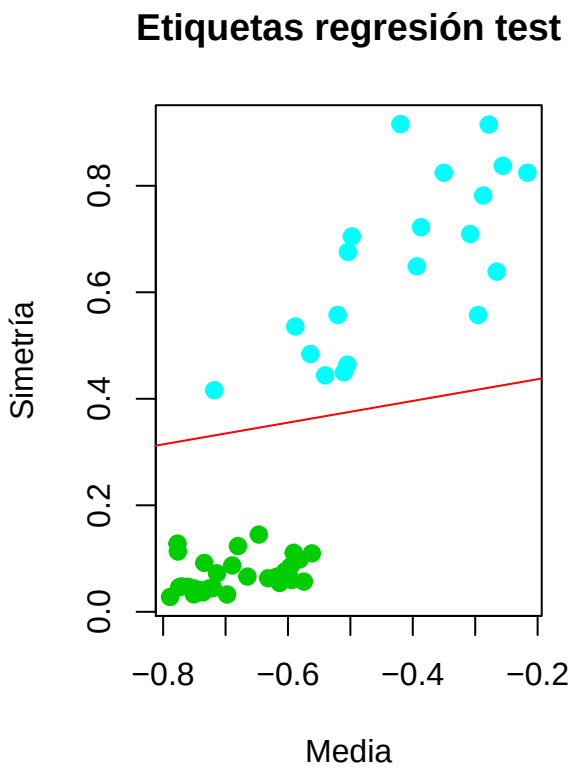
```
apartado3()
```






```
## El error dentro de la muestra de entrenamiento (Ein) es del  0.1669449 %
```





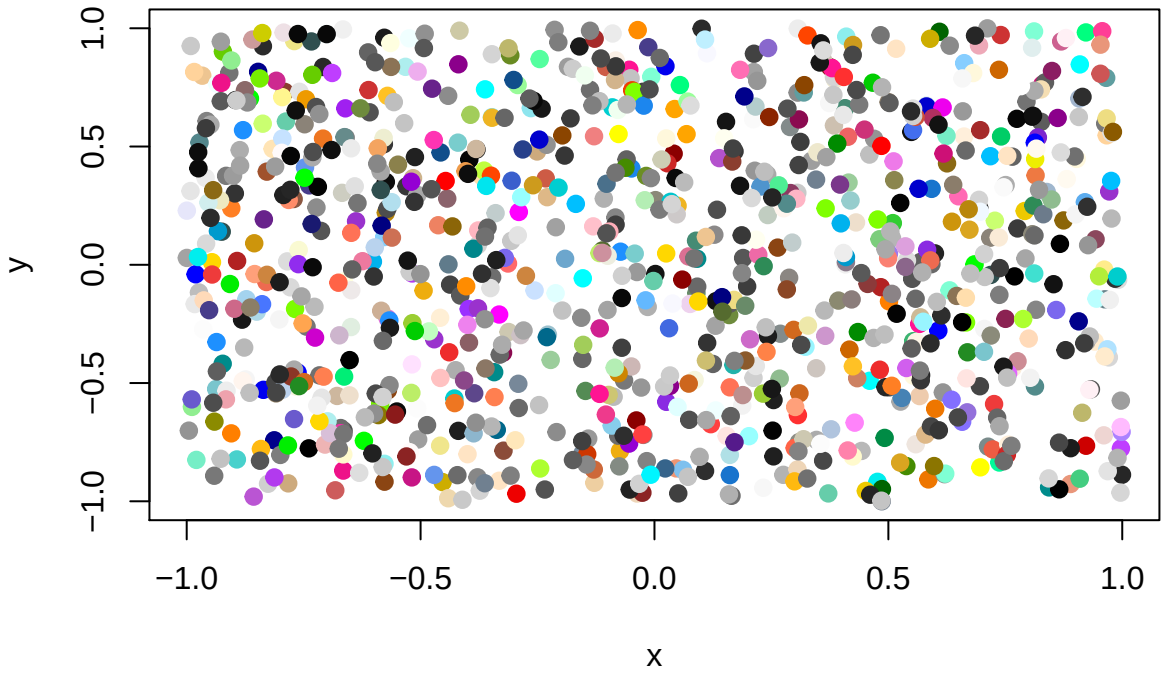

```
## El error dentro de la muestra de entranmiento (en test) (Eout) es del  4.081633 %
```

Como podíamos esperar hemos separado las regiones donde había más 1 de las de los 5 exceptuando el punto que actuaba como ruido. Al ejecutarlo sobre el conjunto de test vemos que aparece un punto más de ruido con respe cto a los obtenidos en el training. Como el conjunto de prueba es mucho más reducido el porcentaje de error E_{out} se dispara aunque sólo ha fallado en 2 puntos de los 49 (que es mucho superior a 1 de los 599 de training).

4) En este apartado exploramos como se transforman los errores Ein y Eout cuando aumentamos la complejidad del modelo lineal usado. Ahora haremos uso de la función simula_unif(N, 2, size) que nos devuelve N coordenadas 2D de puntos uniformemente muestreados dentro del cuadrado definido por [-size,size] x [-size, size]

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

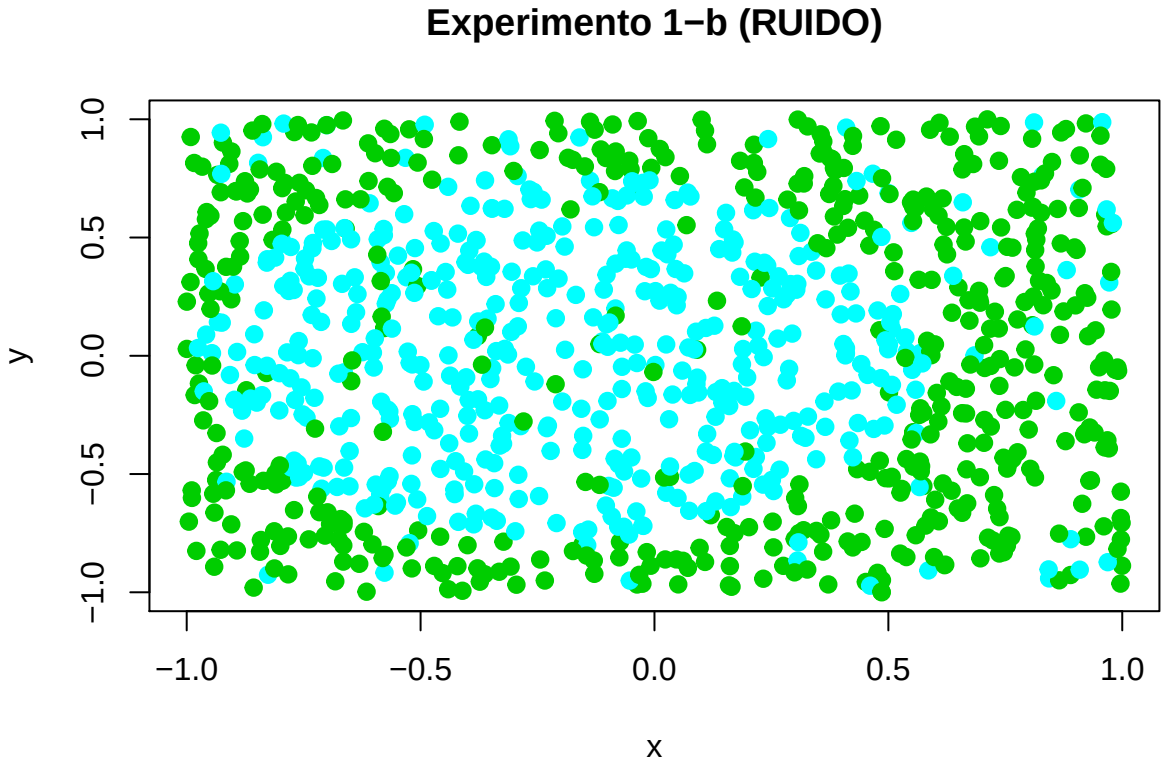
```
## EXPERIMENTO 1
etiquetado_exp1 = function (x1, x2) {
  sign((x1 + 0.2)^2 + x2^2 - 0.6)
}
exp1a = function (size, plot = T) {
  puntos = simula_unif(N = 1000, dims = 2, c(-size, size))
  if (plot) {
    # Ponemos los puntos con colores aleatorios.
    plot(puntos, xlab ="x", ylab ="y", lwd = 2, pch = 19,
         main = "Experimento 1", col = colors())
  }
  puntos
}

puntos_exp1a = exp1a(1)
```



Como podemos observar todos los puntos obtenidos ahora (1000) se encuentran dentro del cuadrado [-1,1]x[-1,1].

- b) Consideremos la función $f(x_1, x_2) = sign()$ que usaremos para asignar una etiqueta a cada punto de la muestra anterior. Introduciremos ruido sobre las etiquetas cambiando aleatoriamente el signo de un 10 % de las mismas. Pintar el mapa de etiquetas final.

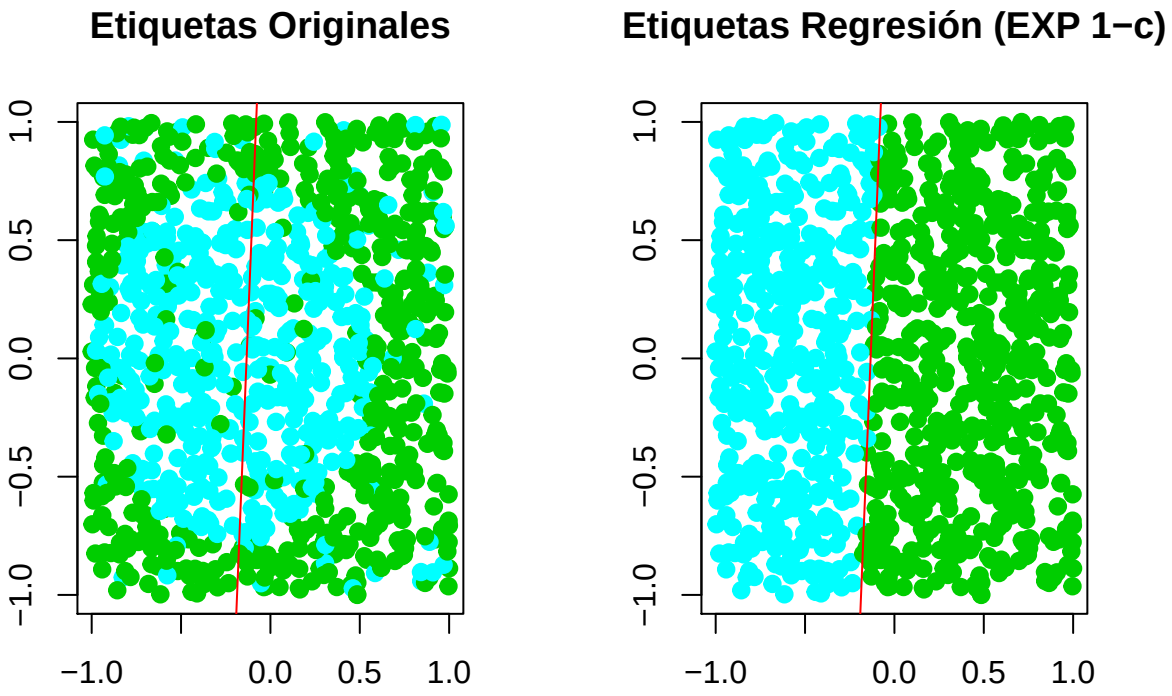
```
exp1b = function(puntos = puntos_exp1a, plot = T) {
  etiquetas = mapply(etiquetado_exp1, puntos[,1], puntos[,2])
  # Metemos ruido del 10 %
  etiquetas = mete_ruido(etiquetas, 0.1)
  if (plot) {
    plot(puntos, xlab = "x", ylab = "y", lwd = 2, pch = 19,
         main = "Experimento 1-b (RUIDO)", col = 32 + 19 * etiquetas)
  }
  # Devolvemos las nuevas etiquetas
  etiquetas
}

etiquetas_exp1b = exp1b()
```



Podemos observar que la mayoría de los puntos clasificados corresponden a la función propuesta que delimita una especie de elipse de puntos cian. La aparición de ruido hace que aparezcan puntos verdes dentro de esa elipse y puntos cian fuera de la misma.

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

```
exp1c = function(puntos = puntos_exp1a, etiquetas_orig = etiquetas_exp1b,
                 plot = T) {
  entradaAjuste = puntos
  ajuste = Regress_Lin(entradaAjuste, etiquetas_orig)
  par(mfrow=c(1,2))
  etiquetas_regr = apply(puntos, 1, etiquetar_regresion, w = ajuste)
  ein = sum(etiquetas_regr != etiquetas_orig)
  if (plot) {
    plot(puntos, xlab = "", ylab = "", lwd = 2, pch = 19,
         main = "Etiquetas Originales", col = 32 + 19 * etiquetas_orig)
    calcula_recta(ajuste, pintar = T)
    plot(puntos, xlab = "", ylab = "", lwd = 2, pch = 19,
         main = "Etiquetas Regresión (EXP 1-c)", col = 32 + 19 * etiquetas_regr)
    calcula_recta(ajuste, pintar = T)
    cat("El error Ein es del ", ein/nrow(puntos) * 100, "%\n")
  }
  par(mfrow=c(1,1))

  list(Ein = ein, w = ajuste)
}
exp1cdata = exp1c()
```




```
## El error Ein es del  41.3 %
```

Como podemos ver un modelo basado en variables lineales no es lo suficientemente bueno para darnos una buena aproximación (obteniendo un E_{in} de un 41,3 %) a la función cuadrática que creamos para etiquetar el experimento. Como podemos observar la recta de regresión intenta separar la mayor cantidad de puntos de cada tipo. La inclinación parece determinada por el ruido, especialmente por el ruido producido por los *outliers* de las esquinas.

- d) Ejecutar todo el experimento definido por (a)-(c) 1000 veces (generamos 1000 muestras diferentes) y
- Calcular el valor medio de errores E_{in} de las 1000 muestras
- Generar 1000 puntos nuevos por cada iteración y calcular con ellos el valor de E_{out} de dicha iteración. Calcular el valor medio de E_{out} de todas las iteraciones.

```
experimento1 = function(X = 1) {
  # X es el valor del cuadrado donde obtener puntos
  # Training (Uso los apartados anteriores)
  # a) Genero muestra
  puntos = exp1a(1,plot = F)
  # b) Calculo etiquetas CON RUIDO
  etiquetado_original = exp1b(puntos, plot = F)
  # c) Ajusto y calculo Ein
  ein_ajuste = exp1c(puntos, etiquetado_original, plot = F)
  # Test
  # a) Genero muestras de test
  puntos_test = simula_unif(N = 1000, dims = 2, c(-X,X))
  # b) Genero etiquetas de f SIN RUIDO
  etiquetas_test_orig = mapply(etiquetado_exp1, puntos_test[,1], puntos_test[,2])
  # c) Calculo las etiquetas con la g obtenida en TRAINING
  etiquetas_test_regr = apply(puntos_test, 1, etiquetar_regresion, ein_ajuste$w)
  # d) Calculo Eout
  Eout = sum(etiquetas_test_orig != etiquetas_test_regr)
```

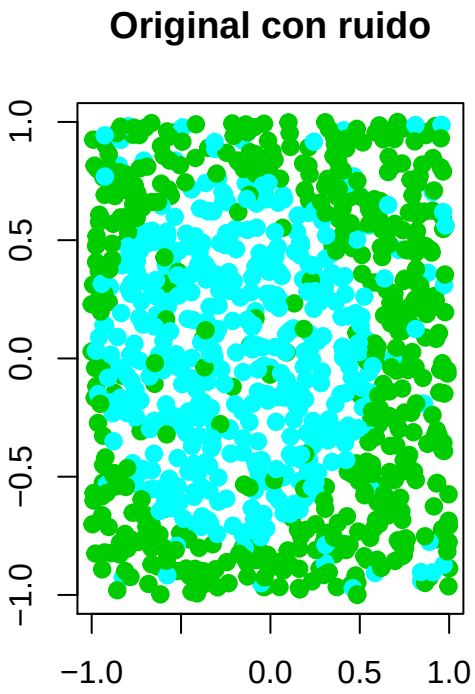
```
  # Devuelvo Ein y Eout
  c(ein_ajuste$Ein, Eout)
}

exp1d = function(iters = 1000) {
  # Repetimos el experimento 1000 veces
  resultados = replicate(n = iters, experimento1())
  # Mostramos las medias por pantalla
  cat("Media de Ein:  ", mean(resultados[1,])/ncol(resultados) * 100," %.\n")
  cat("Media de Eout: ", mean(resultados[2,])/ncol(resultados) * 100," %.\n")
}

exp1ddata = exp1d()
```

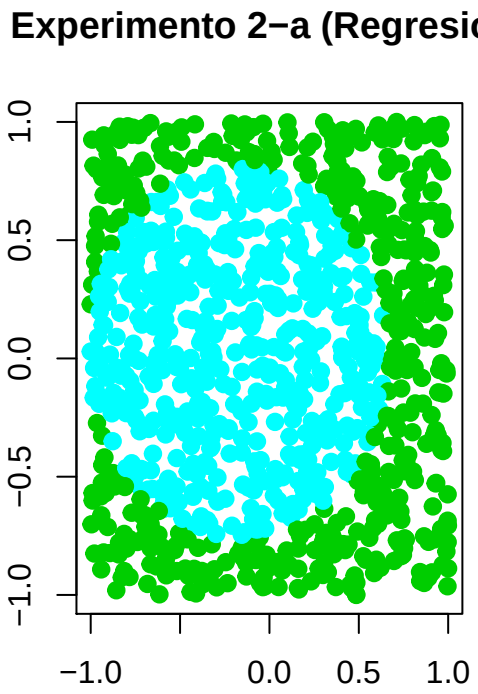
```
## Media de Ein:   39.6255  %.
## Media de Eout:  37.3203  %.
```

Como habíamos observado antes el error E_{in} es bastante elevado de por sí, aunque el caso anterior resulta ser algo más malo con respecto a la media, por lo que el error medio baja ligeramente. Obteniendo un E_{in} tan alto cabe esperar que el E_{out} sea bastante más alto, pero no es el caso. Podemos observar que el E_{out} es un 2 % más bajo que el E_{in}. Esto se debe a que el ruido que existía en el training no era lo suficientemente grande para que la regresión obtuviese una recta que se alejara demasiado de separar la mayoría de los puntos bien clasificados, por lo que se convierte en un estimador de aproximadamente igual error que el de la muestra. El hecho de que el error fuera de la muestra sea menor se debe a que en la prueba de test no hemos generado nada de ruido.

- EXPERIMENTO 2
- a) Ahora vamos a repetir el mismo experimento anterior pero usando características no lineales. Ahora usaremos el siguiento vector de características $\phi_2(x) = (1, x_1, x_2, x_1x_2, x_1^2, x_2^2)$. Ajustar el nuevo modelo de regresión lineal y calcular el nuevo vector de pesos w. Calcular el error E_{in}.

```
  exp2a = function(puntos = puntos_exp1a, etiquetas_orig = etiquetas_exp1b,
                   plot = T) {
    entradaAjuste = cbind(puntos[,1], puntos[,2], puntos[,1]*puntos[,2],
                          puntos[,1]^2, puntos[,2]^2)
    ajuste = Regress_Lin(entradaAjuste, etiquetas_orig)
    etiquetas_regr = apply(entradaAjuste, 1, etiquetar_regresion, ajuste)
    ein = sum(etiquetas_regr != etiquetas_orig)
    if (plot) {
      par(mfrow=c(1,2))
      plot(puntos, xlab = "", ylab = "", pch = 19, lwd = 2,
         main = "Original con ruido", col = 32 + 19 * etiquetas_orig)
    cat("\n")
    plot(puntos, xlab = "", ylab = "", pch = 19, lwd = 2,
         main = "Experimento 2-a (Regresión)", col = 32 + 19 * etiquetas_regr)
    cat("\n")
    par(mfrow=c(1,1))
    cat("El error Ein es del ", ein/nrow(puntos)*100, "%\n")
    }
    list(Ein = ein, ajuste = ajuste)
  }
  ein_ajuste2 = exp2a()
```






```
##
## El error Ein es del  15.1 %
```

Como podemos apreciar en las gráficas la incorporación de las variables cuadráticas hace que la regresión defina mucho mejor la función de clasificación. No obstante, el ruido introducido sigue haciendo que el E_{in} siga siendo elevado para lo deseable pero mucho menos que en el experimento anterior.

- b) Al igual que en el experimento anterior repetir el experimento 1000 veces calculando con cada muestra el vector dentro y fuera de la muestra, E_{in} y E_{out} respectivamente. Promediar los valores obtenidos para ambos errores a lo largo de las muestras.

```
experimento2 = function(X=1) {
  # Cogemos los puntos
  puntos = exp1a(X, F)
  # Adaptamos los puntos al vector de características
  caracteristicas = cbind(puntos[,1], puntos[,2], puntos[,1]*puntos[,2],
                          puntos[,1]^2, puntos[,2]^2)
  # Calculo el etiquetado original
  etiquetado_orig = exp1b(puntos, F)
  # Ajustamos con el vector de características
  ein_ajuste = exp2a(caracteristicas,etiquetado_orig, F)
  w = ein_ajuste$ajuste
  # Saco una muestra de test
  puntos_test = simula_unif(N = 1000, dims = 2, c(-1,1))
  # Obtengo las etiquetas originales de test
  etiquetas_orig = mapply(etiquetado_exp1, puntos_test[,1], puntos_test[,2])
  # Obtengo las etiquetas de regresión de test
  caracteristicas_test = cbind(puntos_test[,1], puntos_test[,2],
      puntos_test[,1]*puntos_test[,2], puntos_test[,1]^2, puntos_test[,2]^2)
  etiquetas_regr = apply(caracteristicas_test, 1, etiquetar_regresion, w)
  Eout = sum(etiquetas_orig != etiquetas_regr)
  c(ein_ajuste$Ein, Eout)
}
```

```
exp2b = function(iters = 1000) {
    resultados = replicate(iters, experimento2())
    cat("La media de Ein es de ", mean(resultados[1,])/iters*100, "%\n")
    cat("La media de Eout es de ", mean(resultados[2,])/iters*100, "%\n")
  }

  exp2b()
```

```
## La media de Ein es de  14.1684 %
## La media de Eout es de  5.6149 %
```

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

Como hemos podido observar sorprendentemente E_{out} es bastante más pequeño que E_{in}. De manera similar al experimento anterior el 10 % de ruido no ha sido suficiente para comprometer excesivamente el ajuste realizado por nuestro modelo. Otro de los posibles problemas en lo que podíamos haber caído es en el sobreaprendizaje, es decir, en que hubiese aprendido características propias del conjunto de entrenamiento que no son propias de la función f que intentamos aprender, pero de media no se manifiesta significativamente un problema grave de sobreaprendizaje.

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

Evidentemente el segundo modelo, al tener menores valores tanto de E_{in} como de E_{out} de media, es mucho mejor ya que tendremos más posibilidades de acertar la clasificación de una nueva muestra al ser el error menor.

BONUS

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 $\chi = [-10, 10]$ x $[-10, 10]$ y elegimos muestras aleatorias uniformes dentro de χ. La función f en cada caso sería una recta aleatoria que corta a X χ que asigna una etiqueta a cada punto de χ con el valor del signo de f en dicho punto. En cada apartado generaremos una muestra y le asignamos una 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 E_{in}?

```
regresion_recta_ein = function(X = 10, N = 100) {
  # Calculamos una muestra aleatoria de tamaño N
      muestras = simula_unif(N = N, dims = 2, rango = c(-X, X))
      # Calculamos una recta f aleatoria que corta el cuadrado
      f = simula_recta(intervalo = c(-X,X))
      # Calculamos las etiquetas con respecto a la recta
      etiquetas = apply(X = muestras, MARGIN = 1, FUN = etiquetar, recta = f)
```

```
      # Calculamos la función de regresión g (pesos)
      g = Regress_Lin(datos = muestras, label = etiquetas)
      # Calculamos las etiquetas con respecto a la regresión
      etiquetasRegresion = apply(muestras, 1, etiquetar_regresion, g)

      # Devolvemos ein
      sum(etiquetas != etiquetasRegresion)
}
bonus1a = function(iters = 1000) {
    Ein = replicate(iters,regresion_recta_ein())
    cat("La media de Ein es ", mean(Ein), "\n")
  }
  bonus1a()
```

```
## La media de Ein es  3.855
```

Al utilizar la recta de regresión obtenemos una recta que se pone justo en la mitad para separar las muestras, por tanto, mientras saquemos puntos de una distribeución uniforme obtendremos una recta bastante parecida a la original, obtiendo un E_{in} bastante bajo. Si conseguimos una muestra lo suficientemente elevada puesto que el problema es separable linealmente acabaríamos encontrando la recta original.

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). ¿Qué valor obtiene de E_{out}. Valore el resultado.

```
regresion_recta_eout = function(X = 10, N = 100) {
  # TRAINING
      # Calculamos una muestra aleatoria de tamaño N
      muestras = simula_unif(N = N, dims = 2, rango = c(-X, X))
      # Calculamos una recta f aleatoria que corta el cuadrado
      f = simula_recta(intervalo = c(-X,X))
      # Calculamos las etiquetas con respecto a la recta
      etiquetas = apply(X = muestras, MARGIN = 1, FUN = etiquetar, recta = f)
      # Calculamos la función de regresión g (pesos)
      g = Regress_Lin(datos = muestras, label = etiquetas)

  # TEST
      # Cogemos una nueva muestra aleatoria uniforme de tamaño N
      muestras = simula_unif(N = N, dims = 2, rango = c(-X,X))
      # Miramos las etiquetas de test con respecto de la recta
      etiquetas = apply(X = muestras, MARGIN = 1, FUN = etiquetar, recta = f)
      # Calculamos las etiquetas de test con respecto a la regresión
      etiquetasRegresion = apply(muestras, 1, etiquetar_regresion, g)

      # Devolvemos eout
      sum(etiquetas != etiquetasRegresion)
}
  bonus1b = function(iters = 1000) {
    Eout = replicate(iters, regresion_recta_eout())

    cat("La media de Eout es ", mean(Eout), "\n")
```

```
  }
  bonus1b()
```

```
## La media de Eout es  4.767
```

Puesto que la recta de regresión depende de los datos iniciales se ajustará a la distribución que se haya obtenido de dichos datos. Es esperable que haya una ligera variación con respecto al valor obtenido en E_{in} y lo más lógico es que el E_{out} sin estar bien ajustada la recta es que esta vaya a peor.

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 1000 veces ¿Cuál es el 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.

```
regresion_pla = function(N = 10, X = 10) {
  muestras = simula_unif(N = N, dims = 2, rango = c(-X, X))
  # Calculamos una muestra aleatoria de tamaño N
  muestras = simula_unif(N = N, dims = 2, rango = c(-X, X))
  # Calculamos una recta f aleatoria que corta el cuadrado
  f = simula_recta(intervalo = c(-X,X))
  # Calculamos las etiquetas con respecto a la recta
  etiquetas = apply(X = muestras, MARGIN = 1, FUN = etiquetar, recta = f)
  # Calculamos la función de regresión g (pesos)
  g = Regress_Lin(datos = muestras, label = etiquetas)
  # Aplicamos PLA
  pla = ajusta_PLA(muestras,etiquetas,1000,g)
  pla$iteraciones
}

bonus1c = function(iters = 1000) {
  resultados = replicate(1000, regresion_pla())
  print(resultados)
  cat("La media de iteraciones utilizadas por el PLA inicializado ",
      "con regresión es ", mean(resultados), "\n")
}

bonus1c()
```

```
##    [1]   1   1   1  16   1   1   1   1   1   1   1   1   1   1   1   1   8
##   [18]   1   1   1   1   5   1   1   1   1   1   1   1   1   8   1   1   1
##   [35]   1   1  60   1   1  11   1  12  40  86   1   1   1   1   1   1   1
##   [52]   8   1   7  21   1   1   1   1   1  12  16   1  12   1   1 129   1
##   [69]   1   1   1   1   1   1   7   1   1   1   1   1   1   4   8   1   1
##   [86]  11  11   1   1   1   1   1   1   1   1   1   1   1   1   1   1   1
##  [103]   1   1   1   1  44   1  12   1   1   5   1   1   1   3   1   1   1
##  [120]   1   1   9   1   1   1   1   1   1   1   1  11  10   1   1  14   1
##  [137]   1   1   1   1   1   1   1   1   1   7   1   1   5   1   1   1   1
##  [154]   1   1   1   1   1   8  14   1   1   1   1   1   9   1   1   7   1
##  [171]   1   1   1   1  41  20   1   1   1   1   1   1  28   1   1   1   1
```

```
##  [188]  10   1   1   1  25   1   1   1   1   1   1   1   1   1   1   1  24
##  [205]   1   1   1   1   3   1   1   1   1   4   1   8   1   8   1   1   1
##  [222]   1   1  16   1  25   1  15   1   1  14   8   1   1   1   1   1   1
##  [239]   1   1   1   1   1   1   1  12   8   1   1   1   1   1   6   1  17
##  [256]   1  37   1   1  95   1   1   1   1   1   1   1  21   1   1   2   1
##  [273]   5   1   1   1   1  13   1   1   1   1   1   1   1  27   1   1   1
##  [290]   1   6  12  19  43   1   1   1   1   8   1   1   1   1   1   1   1
##  [307]   1   1   1   1  18  24  20   1   1  14   1   1   1  38  19   1   1
##  [324]   1   1   1   3   1   3   1   1   1   1   8   1  10   6   1   1  17
##  [341]   1   1   1  78   1   1  72   1  80  57   8   1  19  25   1   1   1
##  [358]   1   9   1   1  17   1   1   1  13   1   1   1   1   1   1   1   1
##  [375]   1   1   1   1   8   1   1   1   1   1   9   1   1   4   1   7   1
##  [392]   1   1   1   1   1   1  23   1   1  24   1   1   1   1   1   1   1
##  [409]  13   1  11   1   1   1   1   1  58   1   1   1  82   1   1  14   1
##  [426]   1   1   1   1   1   1   1   7   1   1   1   1   1   1  37  16   1
##  [443]   1   1  41   7   1   1   1  32   1   1   1  10   1   1   1   1   1
##  [460]   1   7   1   1   1   1   1  31   1   1   1   1   1   1   1  47   1
##  [477]   1   1   1  26   1   1   3   1   1   1   1  46   1  12   1   1   1
##  [494]   6   1   1   1   1  40   1   1  71   1  15   1   1   1   1   1   2
##  [511]   1   1   1   1   4   1   1   1   1   1   6  14   1   1  12   9   1
##  [528]   1   1   7   1   1   1  13   1   1   1   1  24   1   1   1   1   1
##  [545]   1   1   1   1   1   1   1  19   1   1  12   1   1   1   1   1  40
##  [562]   5   1   8   1   1   1   1   1   1   1  10   1  26   1   1  18  36
##  [579] 101   1   1   1   6   1   1  26   1   1   1   1   4   1   1   1   1
##  [596]  13   1   1  68  14   1   1   1  12   1   1   1   1   1   1   1   1
##  [613]   1   1   1   1   1   1   1   1  28   1   1   1   8   1   8   1  18
##  [630]   4  13   1   1   1   1   1   1   1   1  13   1   1   1  25   1   1
##  [647]   1   1   1   1   1   3  27  17   1   1   1   1   2   1  14   1  80
##  [664]   1   5   1   1   1   1   1  40   1   1   1   1   1  39   3   1   1
##  [681]   1   1   1   1   1   1   1  49   1   1   1   1   1   1   1  14   1
##  [698]   1   1   9   1   1   1   1   1   1   1   1   1   4  12   1   1   1
##  [715]   1  12   1   1  24 107   1  40   1  12   1   1   1   1   1  15   1
##  [732]   1   1   1   1   1 344   1   1  29   1  56   1   1   1   1   1   1
##  [749]   1   1  13   1   1   1   1   1   1   1   8   1   1   1  18   1   1
##  [766]   1   1   1   1   1   1   1   1   3  10   1   1   1   1   1   1   1
##  [783]   1   1   1   1   1   1   1   1   1  20   1   1   1   1  25   1   1
##  [800]  19   1   1   1   1   1   1   1   1   1   1   6  23  12   1   6   1
##  [817]   1   1  65   1  34   1   1   1  15   1  62   1   1  17  44   1  24
##  [834]   8  15  32  23   1  48   1  26  16   1   1   1   1   1   1   1   1
##  [851]   1   1  13   1   1   1   1   1   1   1   1   1   1  15   1   1   1  26
##  [868]   1   1   1   4   1   1   1   1  23   1   1  54   1   1   1   1   1
##  [885]  27   1   1   1  30   1   1   1   1   1   1  22   1   1  11   1   1
##  [902]   1  36   1   1   1  16  67   1   9   1   1   1   9   1   1   1   1
##  [919]   1   1   1   1   1   1   5   1  89   1  68  11  12   1   1   1  29
##  [936]   1   1   1   6   1   1   1   1   1   1   1   1   1   1  13   1   1
##  [953]  18   1   1   1   1   1   1  14   1   1  13  38   1   1   1   1  13
##  [970]   1   1   1   1   1  28   1   1   1  58  67   1   1   6  44   1   1
##  [987]   1  22   1   1   1  18   1   1   1   1  29   1   9   1
## La media de iteraciones utilizadas por el PLA inicializado  con regresión es  6.404
```

De por sí la media no parece una gran indicadora de lo que ocurre realmente, por lo que he decidido mostrar el vector con el número de iteraciones por pantalla. Como podemos observar, la moda es que tarde 1 una única iteración en converger, puesto que al iniciar en una recta de regresión cercana que clasifica bastante bien y sabiendo que la alteración creada por el algoritmo siempre nos lleva a unos pesos que la clasifica

mejor. No obstante, también es relativamente frecuente encontrar casos en los que para encontrar la recta el algoritmo necesita de muchas iteraciones para converger, hecho que hace que la media suba.