Trabajo 1 | Programación

Antonio Miguel Morillo Chica

17/04/2018

0. Introducción

Para esta práctica nos haremos uso de las funciones ya usadas para la práctica anterior que, además, fueron aportada por la profesora. Analizaremos aparece el ruido en las etiquetas a la hora de elegir una función adecuada. Para ello lasfunciones:

- simula_unif(N,dim,rango): que calcula una lista de N vectores de dimensión dim. Cada vector contiene dim números aleatorios uniformes en el intervalo rango.

```
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
}
```

- simula_gaus(N, dim, sigma):que calcula una lista de longitud N de vectores de dimensión dim, donde cada posición del vector contiene un número aleatorio extraido de una distribucción Gaussiana de media 0 y varianza dada, para cada dimension, por la posición del vector sigma.

```
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 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
}
```

- simula_recta(intervalo): que simula de forma aleatoria los parámetros, v = (a,b) de una recta, y = ax + b, que corta al cuadrado [-50,50]x[-50, 50].

```
simula_recta = function (intervalo = c(-1,1), visible=F){

  ptos = simula_unif(2,2,intervalo)
  a = (ptos[1,2] - ptos[2,2]) / (ptos[1,1]-ptos[2,1])
  b = ptos[1,2]-a*ptos[1,1]

  if (visible) {
    if (dev.cur()==1)
      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
  }
```

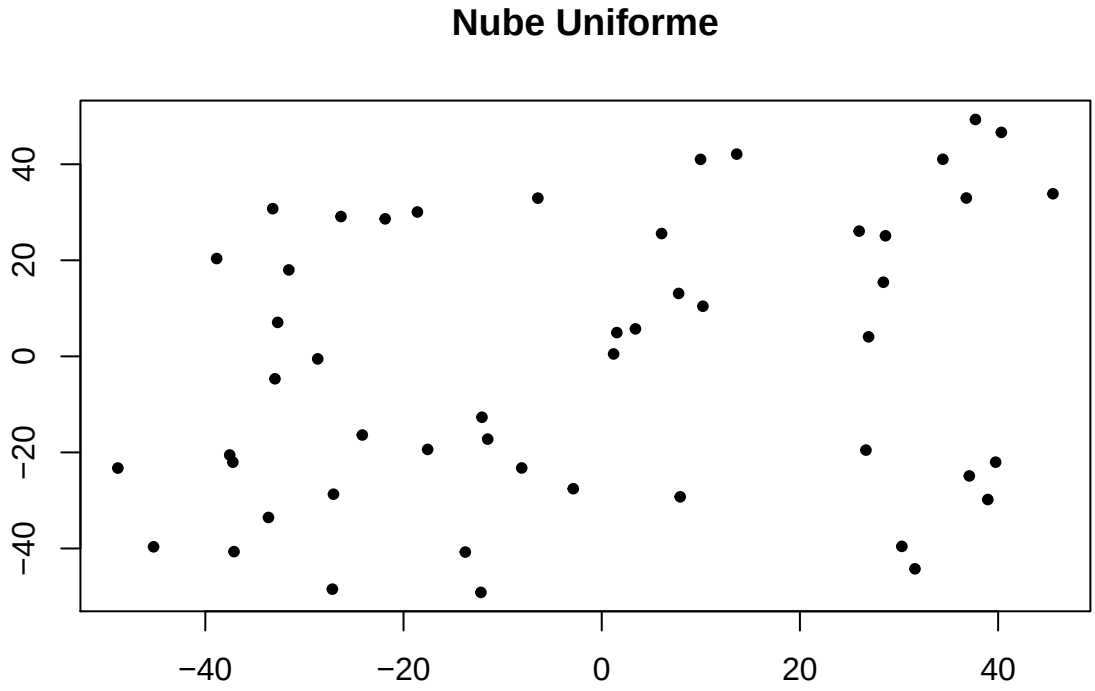
```
  c(a,b) # devuelve el par pendiente y punto de corte
}
```

1. Ejercicio sobre la complejidad de H y el ruido

1.1. Dibujar una gráfica con la nube de puntos de salida correspondiente.

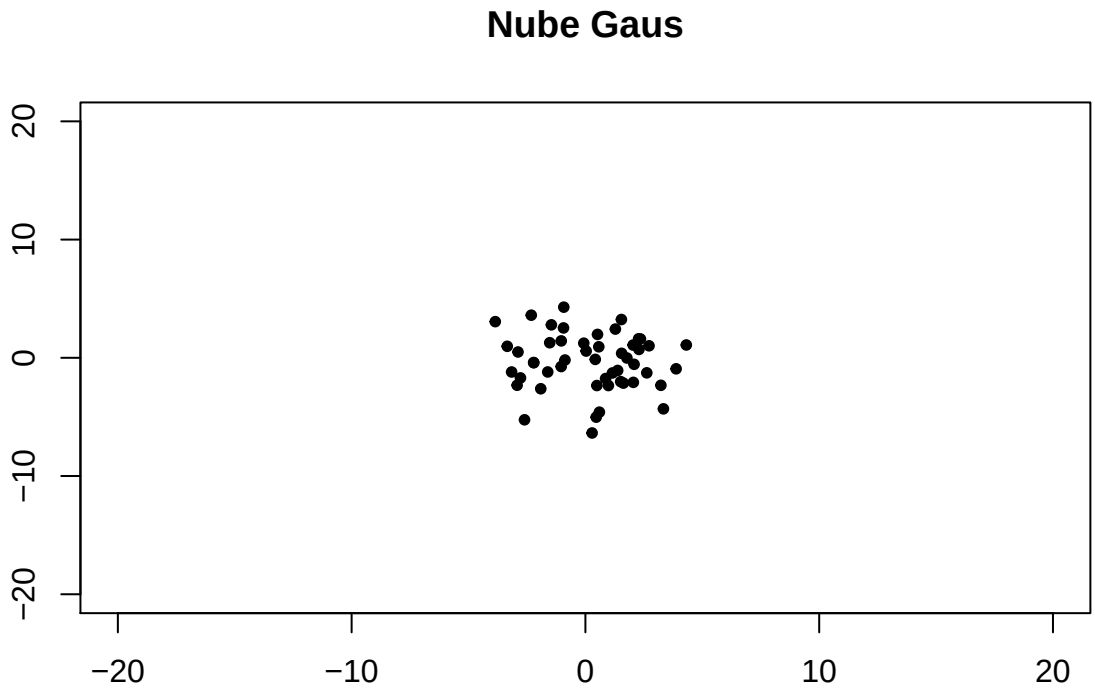
a) Considere N = 50, dim = 2, rango = [-50, +50] con simula_unif (N, dim, rango).

```
datos1 <- simula_unif(50,2,c(-50,50))
plot(datos1, xlab = " ", ylab =" ", pch=20, main = "Nube Uniforme")
```



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

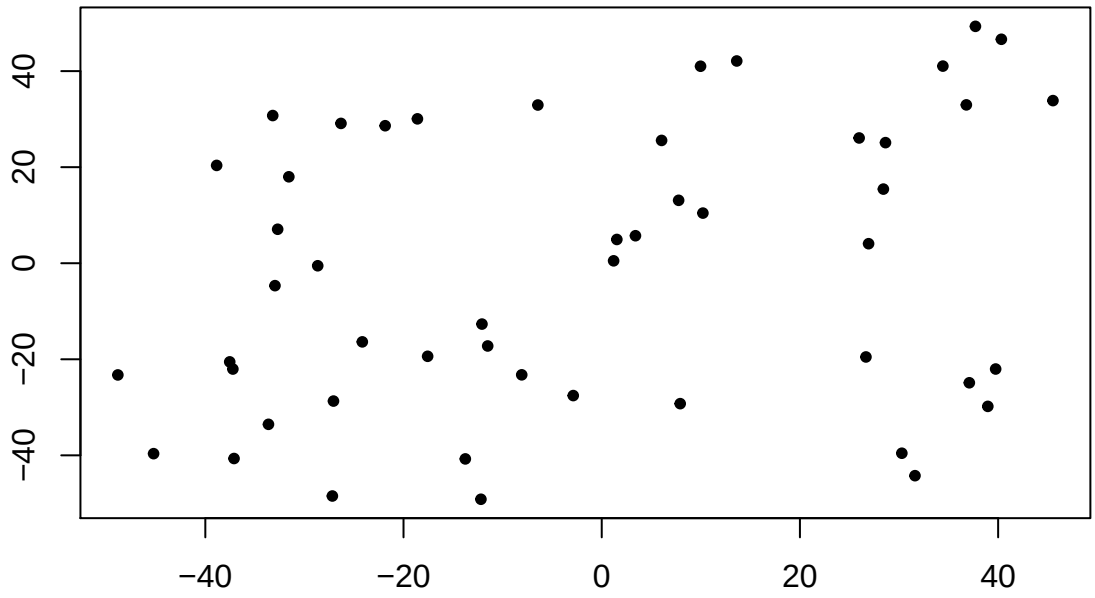
```
datos2 <- simula_gaus(50,2,c(5,7))
plot(datos2, xlab = " ", ylab =" ", pch=20, xlim = c(-20,20),
     ylim = c(-20,20), main = "Nube Gaus")
```



1.2 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().

Lo primero que hacemos es generar nuestra muestra con 50 puntos y con un rango de -50 a 50. Posterioremente necesitremos definir la función signo para, de esta forma, poder clasificar los puntos que están por encima o por debajo de la recta.

```
muestra2D = simula_unif(50,2,c(-50,50))
plot(datos1, xlab = " ", ylab =" ", pch=20, main = "Nube Uniforme muestra2D")
```

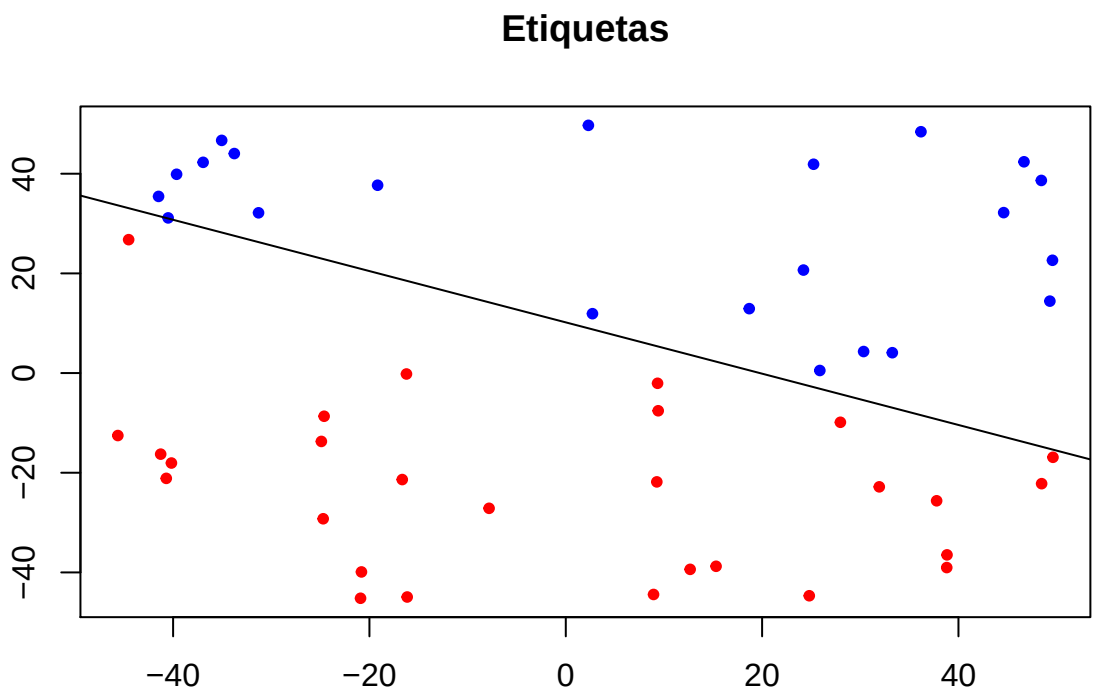



```
ab <- simula_recta(c(-50,50))

signo <- function(xy){
  sign(xy[2]-ab[1]*xy[1]-ab[2])
}
```

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)

Para hacer la distinción entre las etiquetas he creado una función, generaEtiquetas lo que hacemos es cogernos nuestra muestra de datos y nuestra recta y aplicamos la función para averiguar el signo, así los que estén por cencima aparecerán rosaceos y los que estén por debajo, azules, como podemos ver acontinuación:

```
generarEtiquetas <- function(datos = muestra2D, recta = ab ){
  etiquetas <- apply(X = datos, FUN = signo, MARGIN = 1)
  plot(datos, col=etiquetas+3, pch=20, xlab = " ", ylab =" ", main="Etiquetas" )
  abline(ab[2],ab[1])
  etiquetas
}

# Ejemplo propio para la práctica
etiquetas1 <- generarEtiquetas(muestra2D,ab)
```

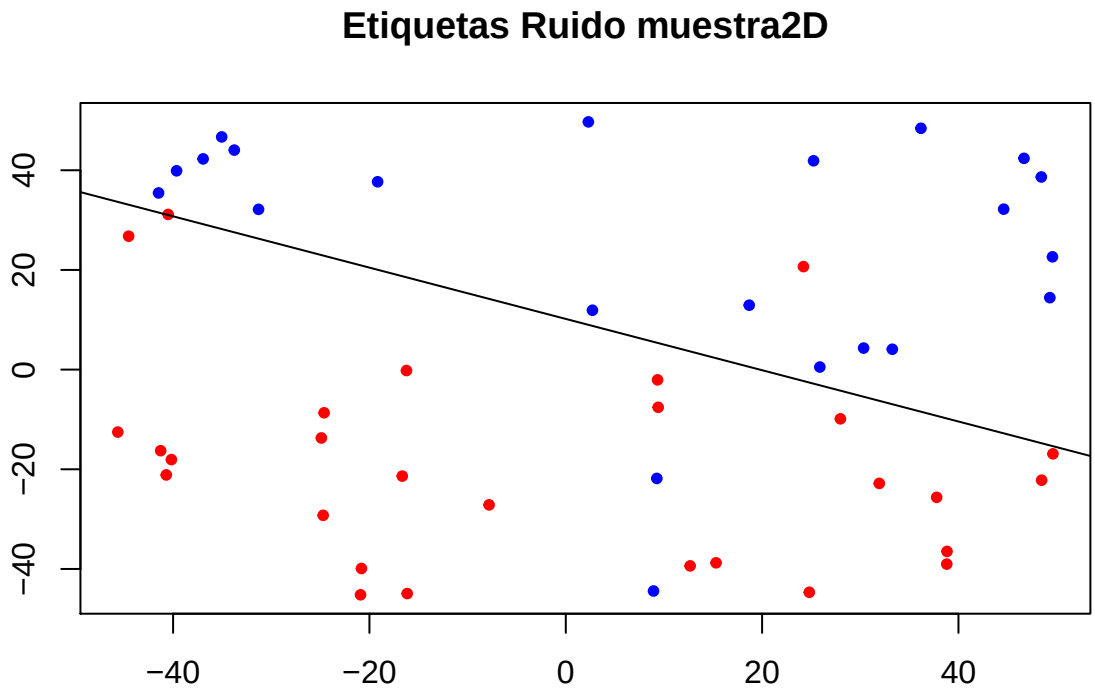


b) Modifique de forma aleatoria un 10 % etiquetas positivas y otro 10 % de negativasy guarde los puntos con sus nuevas etiquetas. Dibuje de nuevo la gráfica anterior. (Ahora hay puntos mal clasificados respecto de la recta)

Para este apartado necesitamos modificar el 10% de las etiquetas para ello he creado una función que la llamaremos "ruido" Esta función nos devolverá el vector de etiquetas con el ruido aplicado.

```
ruido1 <- function(etiquetas=etiquetas1, porcentaje=10){
  # Recogemos las etiquetas positivas y negativas
  etiq_positivas = etiquetas[etiquetas>0]
  etiq_negativas = etiquetas[etiquetas<0]

  # Aplicamos un ruido del 10 porciento
  ruidoX = sample(1:length(etiq_positivas), (length(etiq_positivas)*porcentaje/100))
  ruidoY = sample(1:length(etiq_negativas), (length(etiq_negativas)*porcentaje/100))

  etiq_positivas[ruidoX] = 1;
  etiq_negativas[ruidoY] = -1;

  c(etiq_positivas,etiq_negativas);
}

ruido <- function(etiquetas=etiquetas1, porcentaje=10){
  et = length(etiquetas)
  et1 = sum(etiquetas == 1)
  et2 = sum(etiquetas != 1)

  ind1 = which(etiquetas == 1)
  ind2 = which(etiquetas == -1)
```

```
  etiquetas[sample(ind1,et1*porcentaje/100)] = -1
  etiquetas[sample(ind2,et2*porcentaje/100)] = 1
  etiquetas
}

# Aplicamos ruido a las etiquetas
etiquetasRuido1 <- ruido(etiquetas1,10)
# Pintamos la muestra con el ruido
plot(muestra2D, col=etiquetasRuido1+3, pch=20, xlab = " ", ylab =" ",
     main="Etiquetas Ruido muestra2D" )
abline(ab[2],ab[1])
```



Como podemos ver la cantidad de puntos azules y rojos cambiados es 5 con lo que corresponde al 10% de 50 puntos. Estos puntos están mal clasificados y esa es la idea ver como el ruido afecta a la clasificación de la recta.

En los apartados siguientes veremos como usar otro tipo de funciones para clasificar los datos.

1.3 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 ¿Son estas funciones más complejas mejores clasificadores que la función lineal? ¿En que ganan a la función lineal? Explicar el razonamiento.

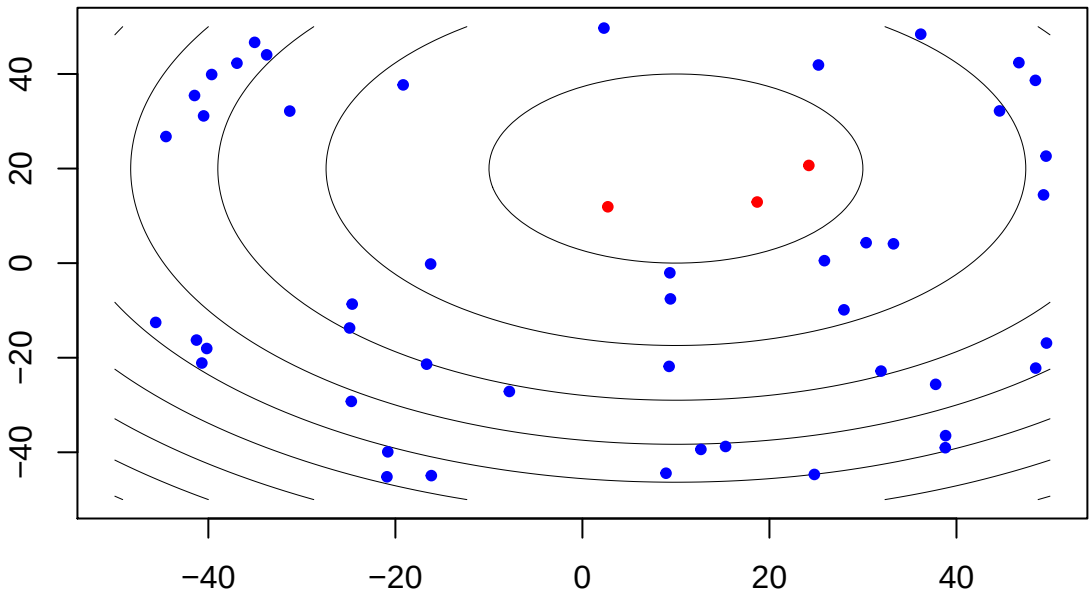
Lo primero que he hecho ha sido definir las funciones aportadas, además usaremos la función pintar_frontera aportada en la práctica anterior por la profesora.

```
f1 <- function(x,y){
  resultado <- ((x-10)^2 + (y-20)^2 - 400)
  resultado
}

f2 <- function(x,y){
  resultado <- ((0.5*(x+10)^2) + (y-20)^2 - 400)
  resultado
}

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

f4 <- function(x,y){
  resultado <- (y - 20*x^2 - (5*x) + 3)
  resultado
}

pintar_frontera = function(f,rango=c(-50,50)) {
  x=y=seq(rango[1],rango[2],length.out = (rango[2]-rango[1]))
  z = outer(x,y,FUN=f)
  if(dev.cur() == 1)
    plot(1,pch = 20, type="n",xlim=rango,ylim=rango)
  contour(x,y,z,drawlabels = FALSE , xlim =rango, ylim=rango, xlab = "", ylab = "",
lwd = 0.5)
}
```

Una vez definidas las funciones pasamos a pintar las diferentes gráficas de las diferentes funciones de clasificación para los datos obtenidos.

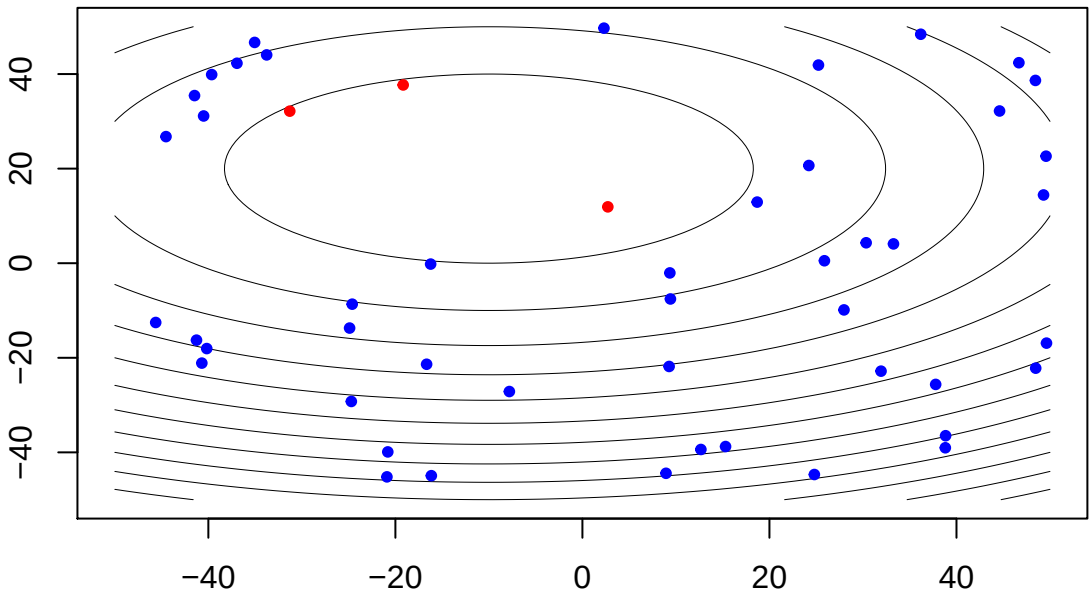
- f(x,y) = $(x-10)^2 + (y-20)^2 - 400$

```
pintar_frontera(f1)
puntos_fuera <- subset(muestra2D,f1(muestra2D[,1],muestra2D[,2])>0)
puntos_dentro <- subset(muestra2D,f1(muestra2D[,1],muestra2D[,2])<0)

points(puntos_fuera, col = "blue",pch = 20)
points(puntos_dentro, col = "red",pch = 20)
```



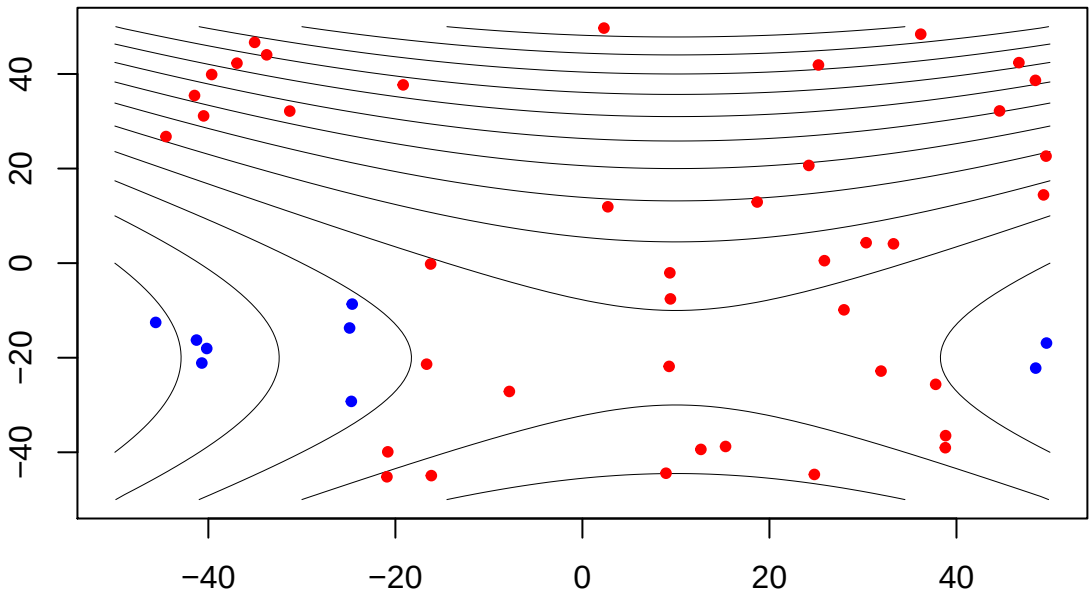
- $f(x,y) = 0{,}5(x+10)^2 + (y-20)^2 - 400$

```
pintar_frontera(f2)
puntos_fuera <- subset(muestra2D,f2(muestra2D[,1],muestra2D[,2])>0)
puntos_dentro <- subset(muestra2D,f2(muestra2D[,1],muestra2D[,2])<0)

points(puntos_fuera, col = "blue",pch = 20)
points(puntos_dentro, col = "red",pch = 20)
```



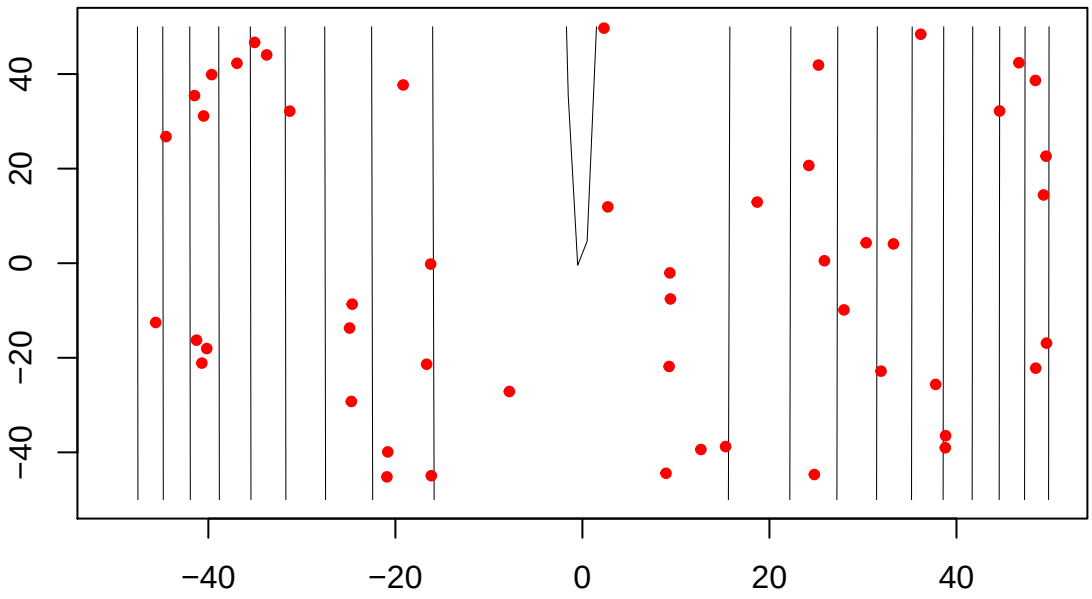
- $f(x,y) = 0{,}5(x-10)^2 - (y+20)^2 - 400$

```
pintar_frontera(f3)
puntos_fuera <- subset(muestra2D,f3(muestra2D[,1],muestra2D[,2])>0)
puntos_dentro <- subset(muestra2D,f3(muestra2D[,1],muestra2D[,2])<0)

points(puntos_fuera, col = "blue",pch = 20)
points(puntos_dentro, col = "red",pch = 20)
```



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

```
pintar_frontera(f4)
puntos_fuera <- subset(muestra2D,f4(muestra2D[,1],muestra2D[,2])>0)
puntos_dentro <- subset(muestra2D,f4(muestra2D[,1],muestra2D[,2])<0)

points(puntos_fuera, col = "blue",pch = 20)
points(puntos_dentro, col = "red",pch = 20)
```



Las diferencias son significativas, la recata divide el espacio en dos partes que separa el conjunto del espacio en dos mides sin embargo la primera y segunda funciones dividen el espacio de la solución en dos también pero una de las regiones está limitada, una por así decirlo, la parte exterior es “infinita” y la del circulo/óvalo es “finita”. La tercera función divide el espacio en 3 zonas donde los puntos mayores de 0 son las partes de la derecha e izquierda de la parábola y el resto del plano de solución es la clasificación para menores de 0. Es curioso ver como en esta función la clasificación para los mayores de 0 se “parten” en dos conjuntos que se encuentran en el espacio de solución opuestos el uno del otro. Por último la última función no separa los datos.

2. Ejercicios Modelos Lineales

2.1 Algoritmo Perceptron: 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 item con su etiqueta está representado por una fila de la matriz, label el vector de etiquetas (cada etiqueta es un valor +1 o -1), max_iter es el número máximo de iteraciones permitidas y vini el valor inicial del vector. La función devuelve los coeficientes del hiperplano.

Lo primero que hacemos antes de entrar en los apartados es definir la función ajusta_PLA. Esta función implementa el algoritmo del Perceptron visto en clase de teoría.

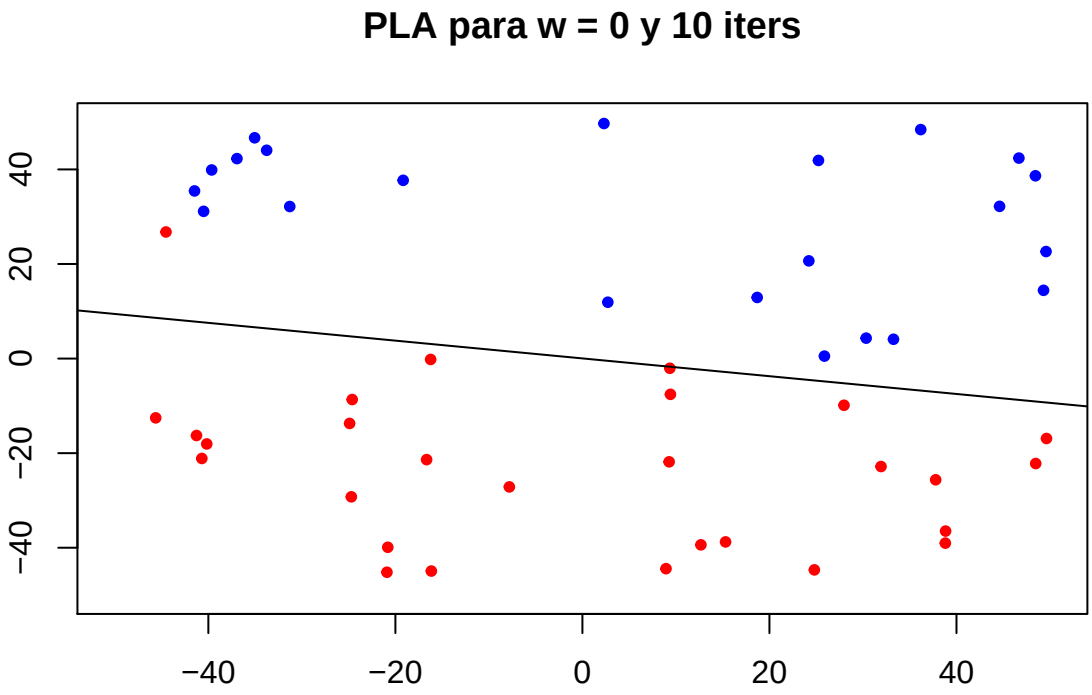
Como *entrada* tenemos: Los datos, las etiquetas, numero de iteracciones y valor inicial. Las iteracciones son necesarias ya que PLA solo para si el conunto de datos es linealmente separable. Como *salida* tenemos: Los parámetros del hiperplano y el número de iteracciones que necesitará.

```
# Algoritmo perceptron
ajusta_PLA <- function(datos, label, max_iter=2000, vini) {
  # Inicializamos el vector de pesos y el contador de iteracciones
  w = vini
  iters = 1
  seguir = TRUE
  # Añadimos una columna para el calculo
  datos = cbind(rep(1,nrow(datos)),datos)
  # Si no llegamos a las iteracciones y debemos seguir...
  while(iters < max_iter & seguir) {
    seguir = FALSE
    # Iteramos sobre cada dato y calculamos su signo ...
    for(i in sample(nrow(datos))) {
      signo = sign(datos[i,] %*% w)
      # Si el signo es distinto a como lo clasificamos,
      # cambiamos el vector de pesos para ajustarnos más
      if(signo != label[i]) {
        w = w + datos[i,]*label[i]

      }else{
        seguir = TRUE
      }
      iters = iters + 1
    }
  }
  # Devolvemos el hiperplano, en este caso,
  # una recta y el num de iters para siguientes apartados
  c(-w[1]/w[3], -w[2]/w[3], iters)
}
```

a) 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.

a) El vector w inicializado a 0 con distinto numero de iteracciones, lo provaremos con 10, 100, 1000, 10000 y 20000

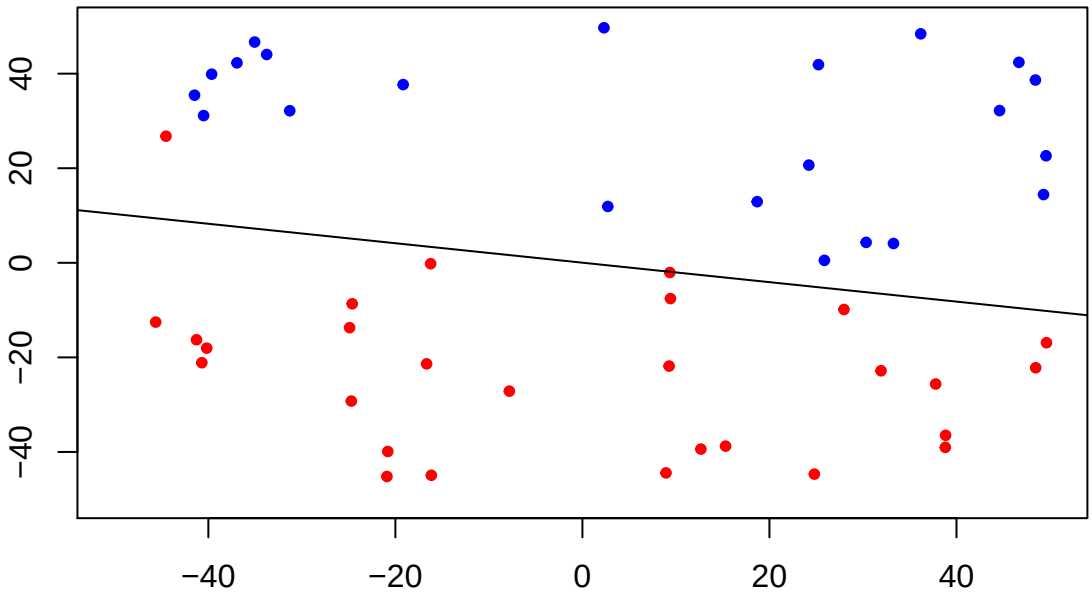
```
# Con 10 iters
# ----------------------------------------------------------------------
perceptron_a <- ajusta_PLA(muestra2D, etiquetas1, 10, c(0,0,0))
plot(muestra2D, main = "PLA para w = 0 y 10 iters", col = etiquetas1+3,
     xlim = c(-50,50), ylim = c(-50,50), xlab = "", ylab = "",pch=20)
abline(perceptron_a[1],perceptron_a[2])
```




```
cat("\nCon ", 10, " -> Num iters necesarias: ", perceptron_a[3])
```

```
##
## Con  10  -> Num iters necesarias:  51
```

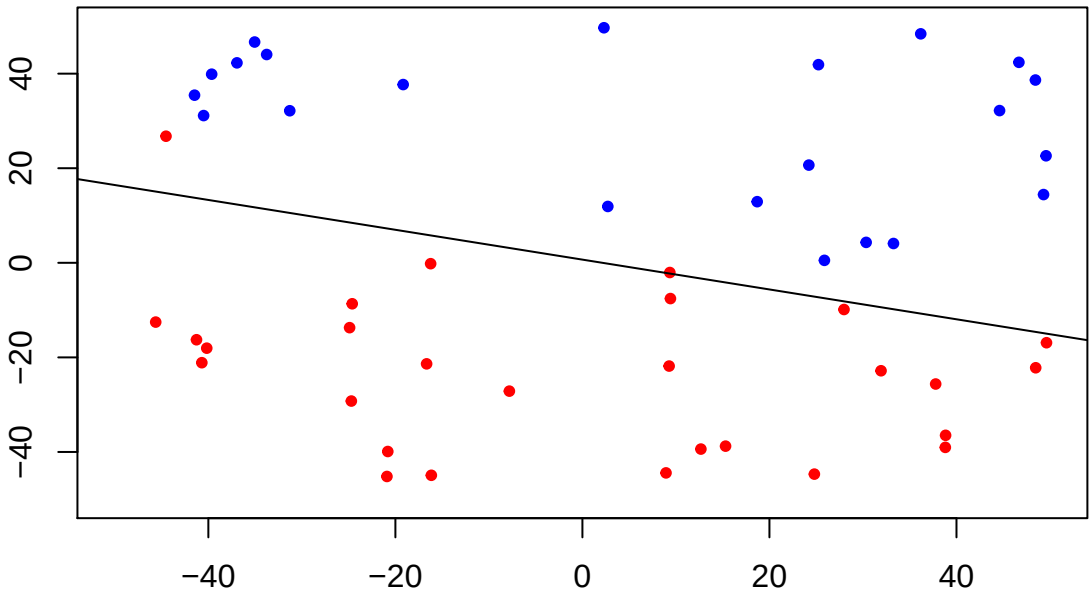
```
#----------------------------------------------------------------------

# Con 100 iters
# ----------------------------------------------------------------------
perceptron_a <- ajusta_PLA(muestra2D, etiquetas1, 100, c(0,0,0))
plot(muestra2D, main = "PLA para w = 0 y 100 iters", col = etiquetas1+3,
     xlim = c(-50,50), ylim = c(-50,50), xlab = "", ylab = "",pch=20)
abline(perceptron_a[1],perceptron_a[2])
```




```
cat("\nCon ", 100, " -> Num iters necesarias: ", perceptron_a[3])
```

```
##
## Con  100  -> Num iters necesarias:  101
```

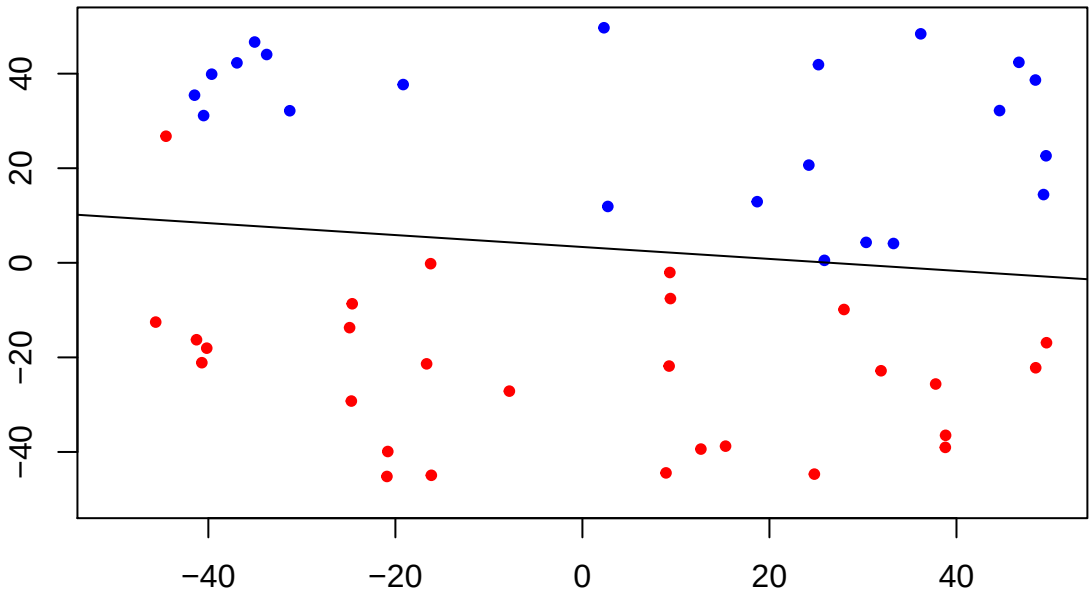
```
#-------------------------------------------------------------------------

# Con 1000 iters
# -----------------------------------------------------------------------
perceptron_a <- ajusta_PLA(muestra2D, etiquetas1, 1000, c(0,0,0))
plot(muestra2D, main = "PLA para w = 0 y 1000 iters", col = etiquetas1+3,
     xlim = c(-50,50), ylim = c(-50,50), xlab = "", ylab = "",pch=20)
abline(perceptron_a[1],perceptron_a[2])
```




```
cat("\nCon ", 1000, "-> Num iters necesarias: ", perceptron_a[3])
```

```
##
## Con  1000 -> Num iters necesarias:  1001
```

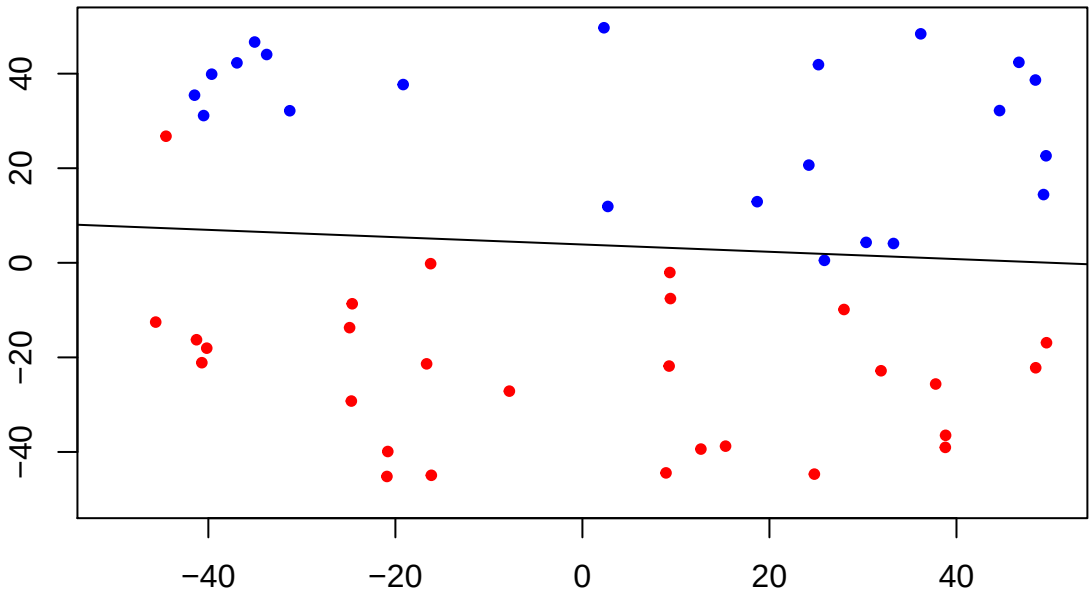
```
#-------------------------------------------------------------------

# Con 10000 iters
# -------------------------------------------------------------------
perceptron_a <- ajusta_PLA(muestra2D, etiquetas1, 10000, c(0,0,0))
plot(muestra2D, main = "PLA para w = 0 y 10000 iters", col = etiquetas1+3,
     xlim = c(-50,50), ylim = c(-50,50), xlab = "", ylab = "",pch=20)
abline(perceptron_a[1],perceptron_a[2])
```




```
cat("\nCon ", 10000, " -> Num iters necesarias: ", perceptron_a[3])
```

```
##
## Con  10000  -> Num iters necesarias:  10001
```

```
#-------------------------------------------------------------------

# Con 20000 iters
# -------------------------------------------------------------------
perceptron_a <- ajusta_PLA(muestra2D, etiquetas1, 20000, c(0,0,0))
plot(muestra2D, main = "PLA para w = 0  y 20000 iters", col = etiquetas1+3,
     xlim = c(-50,50), ylim = c(-50,50), xlab = "", ylab = "",pch=20)
abline(perceptron_a[1],perceptron_a[2])
```




```
cat("\nCon ", 20000, " -> Num iters necesarias: ", perceptron_a[3])
```

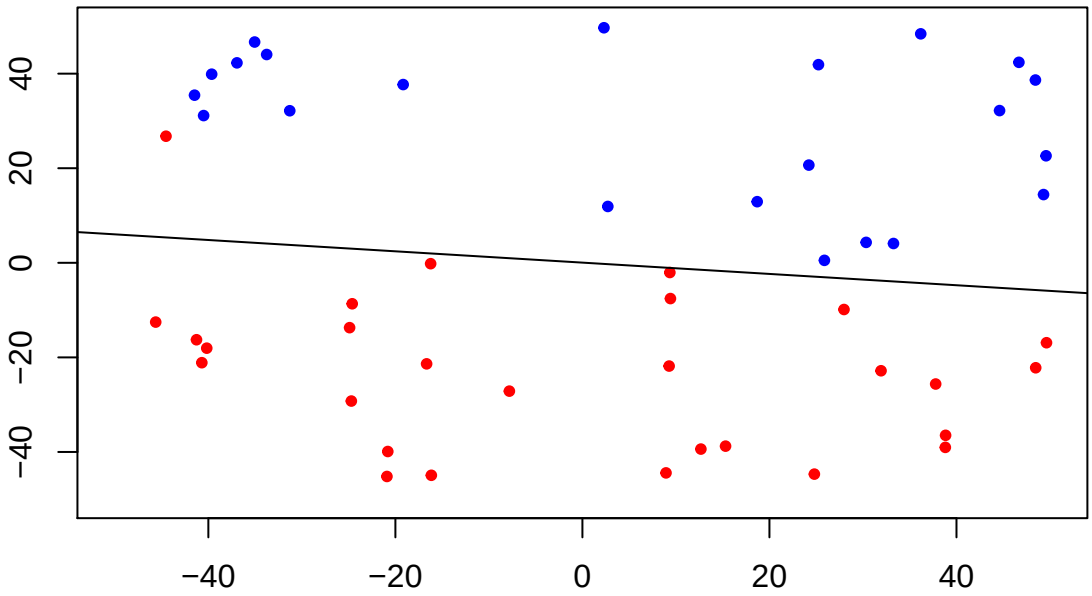
```
##
## Con  20000  -> Num iters necesarias:  20001
```

```
#-----------------------------------------------------------------------
```

Como podemos ver la clasificación para un número <10000 iteracciones produce malas clasificaciones en el algoritmo, le falta iters para poder seguir ajustando más la recta para la clasificación de los datos.

b) El vector w inicializado a valores aleatorios

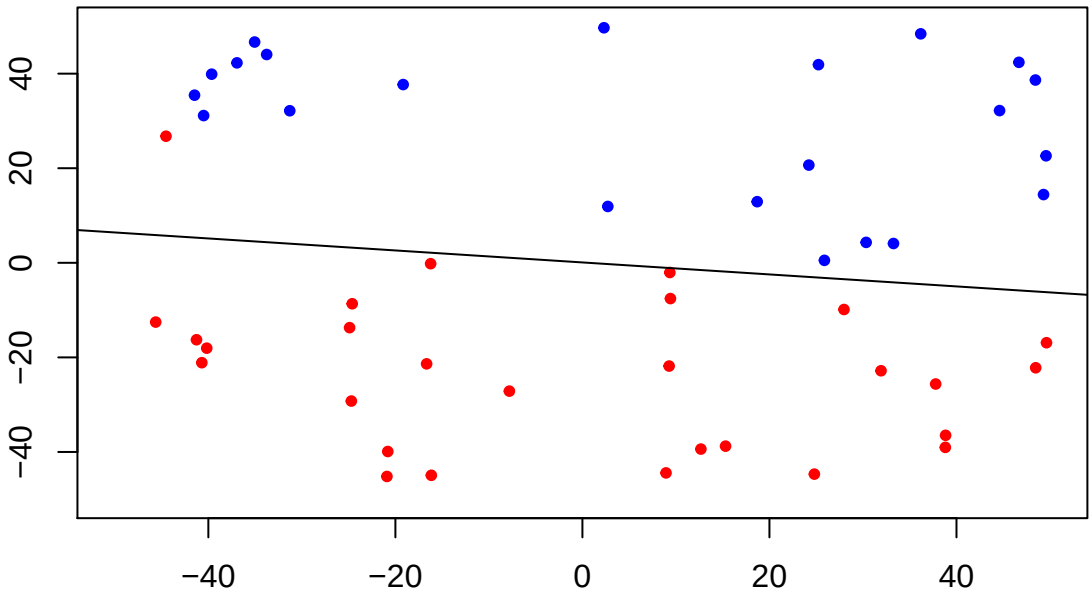
```
# Con 10 iters
# -----------------------------------------------------------------------
perceptron_b <- ajusta_PLA(muestra2D, etiquetas1, 10, runif(3,0,1))
plot(muestra2D, main = "PLA para w Random Para 10 iters", col = etiquetas1+3,
     xlim = c(-50,50), ylim = c(-50,50), xlab = "", ylab = "",pch=20)
abline(perceptron_b[1],perceptron_b[2])
```




```
cat("\nCon ", 10, " -> Num iters necesarias: ", perceptron_b[3])
```

```
##
## Con  10  -> Num iters necesarias:  51
```

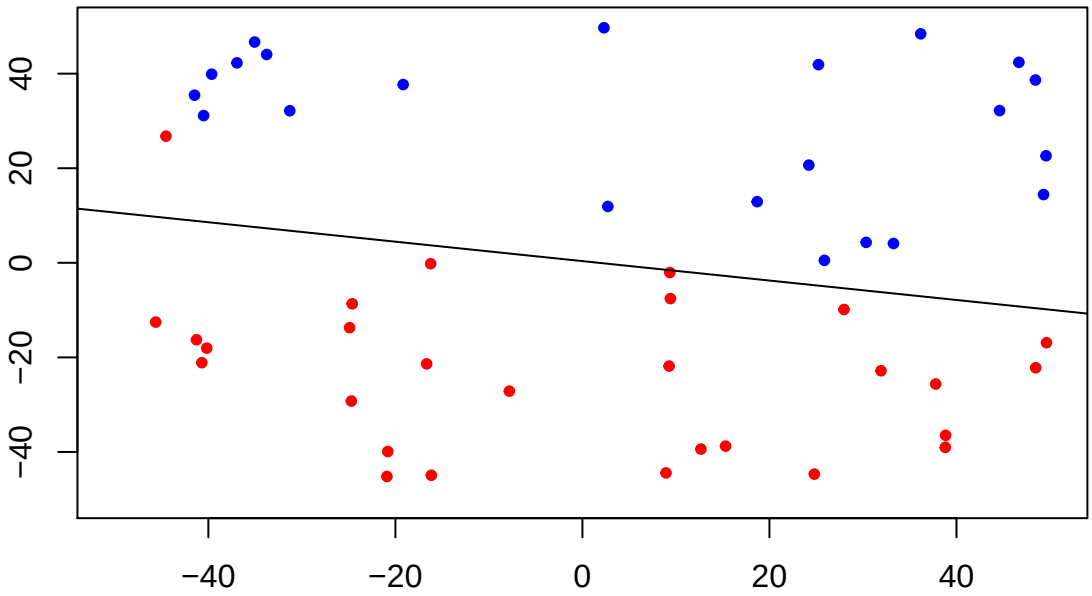
```
# ----------------------------------------------------------------------


# Con 100 iters
# ----------------------------------------------------------------------
perceptron_b <- ajusta_PLA(muestra2D, etiquetas1, 100, runif(3,0,1))
plot(muestra2D, main = "PLA para w Random Para 100 iters", col = etiquetas1+3,
     xlim = c(-50,50), ylim = c(-50,50), xlab = "", ylab = "",pch=20)
abline(perceptron_b[1],perceptron_b[2])
```




```
cat("\nCon ", 100, " -> Num iters necesarias: ", perceptron_b[3])
```

```
##
## Con  100  -> Num iters necesarias:  101
```

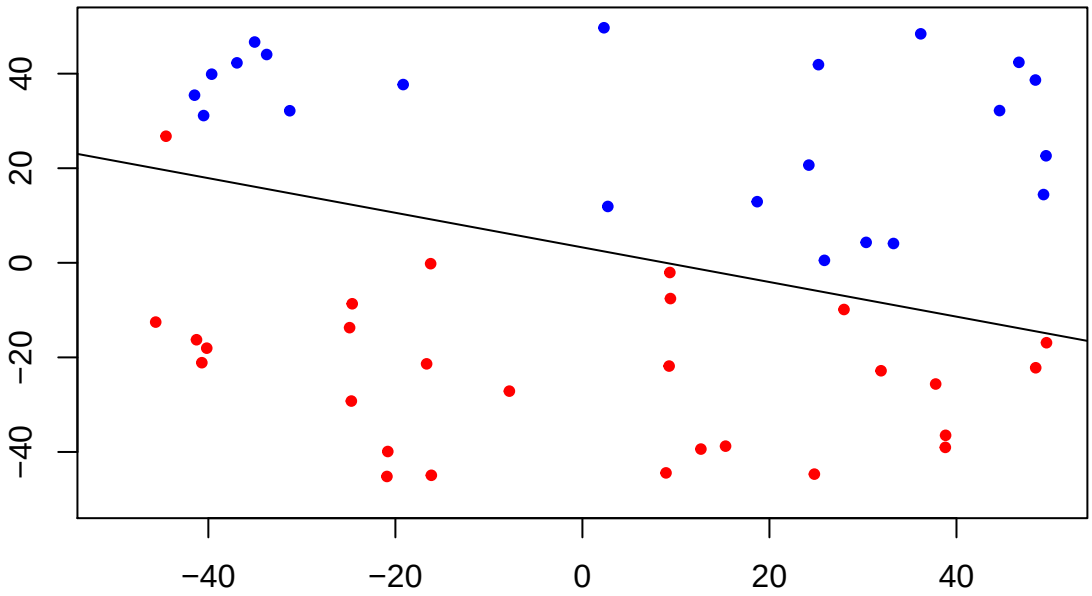
```
# ----------------------------------------------------------------------


# Con 1000 iters
# ----------------------------------------------------------------------
perceptron_b <- ajusta_PLA(muestra2D, etiquetas1, 1000, runif(3,0,1))
plot(muestra2D, main = "PLA para w Random Para 1000 iters", col = etiquetas1+3,
     xlim = c(-50,50), ylim = c(-50,50), xlab = "", ylab = "",pch=20)
abline(perceptron_b[1],perceptron_b[2])
```




```
cat("\nCon ", 1000, " -> Num iters necesarias: ", perceptron_b[3])
```

```
##
## Con  1000  -> Num iters necesarias:  1001
```

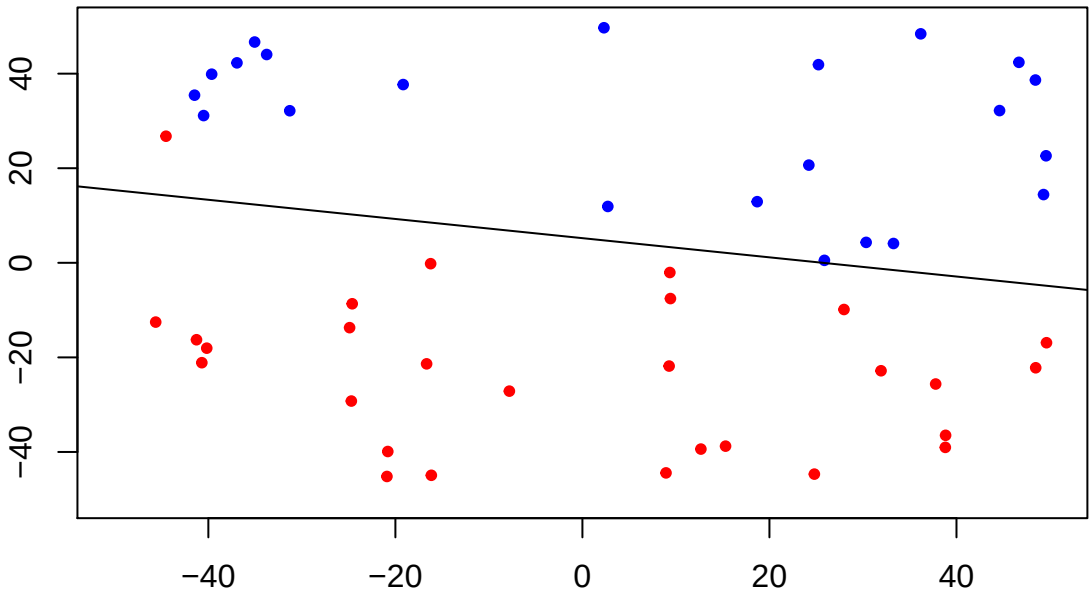
```
# ------------------------------------------------------------------------

# Con 10000 iters
# ------------------------------------------------------------------------
perceptron_b <- ajusta_PLA(muestra2D, etiquetas1, 10000, runif(3,0,1))
plot(muestra2D, main = "PLA para w Random Para 10000 iters", col = etiquetas1+3,
     xlim = c(-50,50), ylim = c(-50,50), xlab = "", ylab = "",pch=20)
abline(perceptron_b[1],perceptron_b[2])
```




```
cat("\nCon ", 10000, " -> Num iters necesarias: ", perceptron_b[3])
```

```
##
## Con  10000  -> Num iters necesarias:  10001
```

```
# ------------------------------------------------------------------------

# Con 20000 iters
# ------------------------------------------------------------------------
perceptron_b <- ajusta_PLA(muestra2D, etiquetas1, 20000, runif(3,0,1))
plot(muestra2D, main = "PLA para w Random Para 20000 iters", col = etiquetas1+3,
     xlim = c(-50,50), ylim = c(-50,50), xlab = "", ylab = "",pch=20)
abline(perceptron_b[1],perceptron_b[2])
```




```
cat("\nCon ", 20000, " -> Num iters necesarias: ", perceptron_b[3])
```

```
##
## Con  20000  -> Num iters necesarias:  20001
```

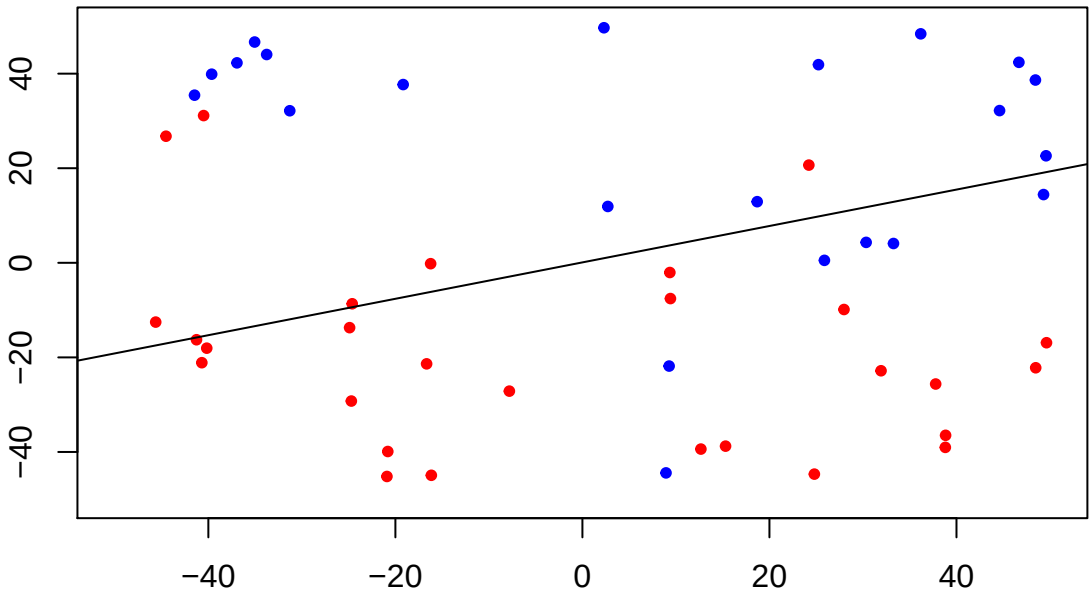
```
# ----------------------------------------------------------------------
```

Como vemos con pesos aleatorios suele tardar algunar iteracciones más pero sin embargo los resultados son muy parecidos al anterior, con iteracciones <10000 se produce una mala clasificación.

b) 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.

a) El vector w inicializado a 0

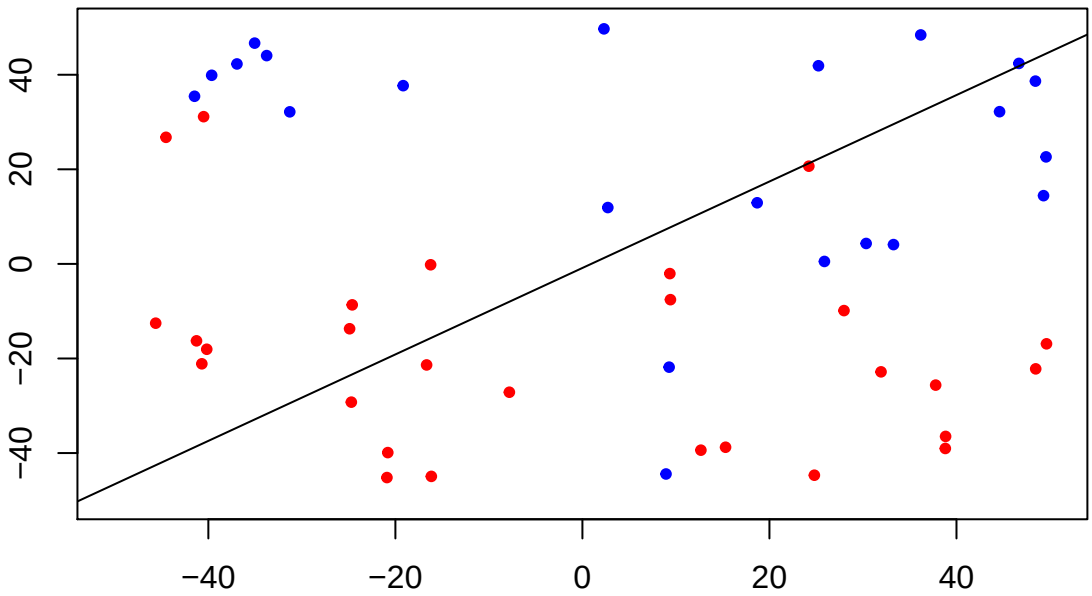
```
perceptron2_a <- ajusta_PLA(muestra2D, etiquetasRuido1, 10, c(0,0,0))
plot(muestra2D, main = "PLA ruido para w = 0 y 10 iters", col = etiquetasRuido1+3,
     xlim = c(-50,50), ylim = c(-50,50), xlab = "", ylab = "",pch=20)
abline(perceptron2_a[1],perceptron2_a[2])
```




```
cat("\nCon: ", 10, " ->Num iters necesarias: ", perceptron2_a[3])
```

```
##
## Con:  10  ->Num iters necesarias:  51
```

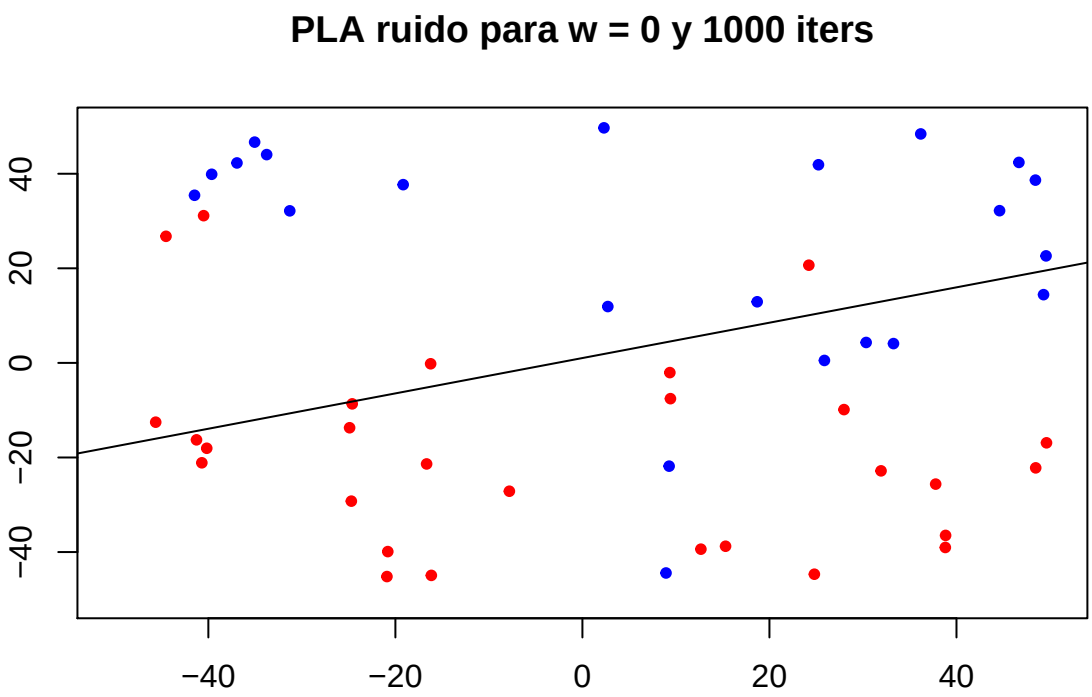
```
perceptron2_a <- ajusta_PLA(muestra2D, etiquetasRuido1, 100, c(0,0,0))
plot(muestra2D, main = "PLA ruido para w = 0 y 100 iters", col = etiquetasRuido1+3,
     xlim = c(-50,50), ylim = c(-50,50), xlab = "", ylab = "",pch=20)
abline(perceptron2_a[1],perceptron2_a[2])
```



```
cat("\nCon: ", 100, " ->Num iters necesarias: ", perceptron2_a[3])
```

```
##
## Con:  100  ->Num iters necesarias:  101
```

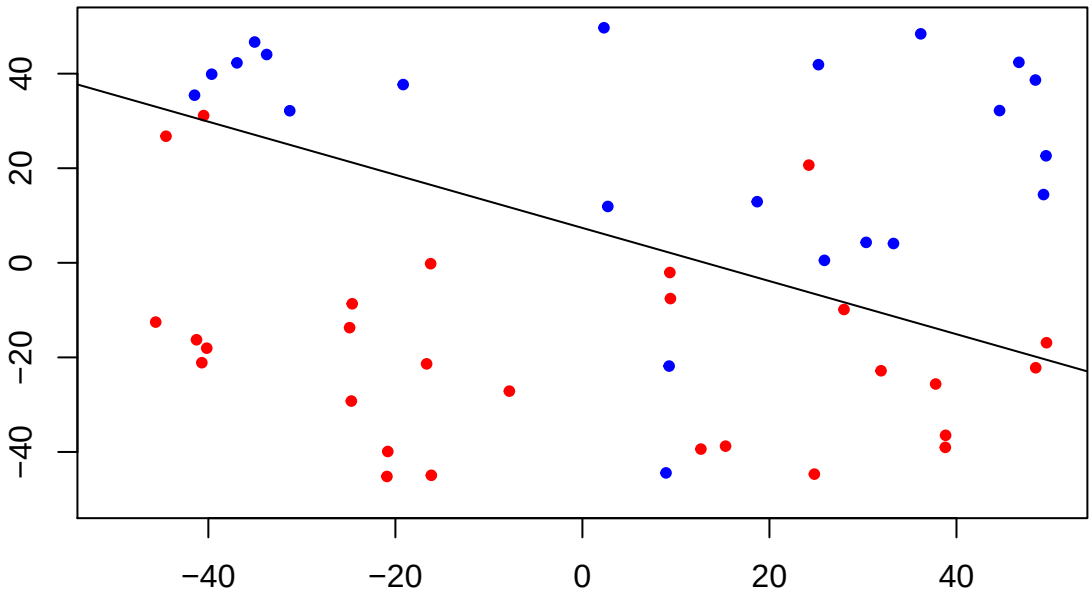
```
perceptron2_a <- ajusta_PLA(muestra2D, etiquetasRuido1, 1000, c(0,0,0))
plot(muestra2D, main = "PLA ruido para w = 0 y 1000 iters", col = etiquetasRuido1+3,
     xlim = c(-50,50), ylim = c(-50,50), xlab = "", ylab = "",pch=20)
abline(perceptron2_a[1],perceptron2_a[2])
```




```
cat("\nCon: ", 1000, " ->Num iters necesarias: ", perceptron2_a[3])
```

```
##
## Con:  1000  ->Num iters necesarias:  1001
```

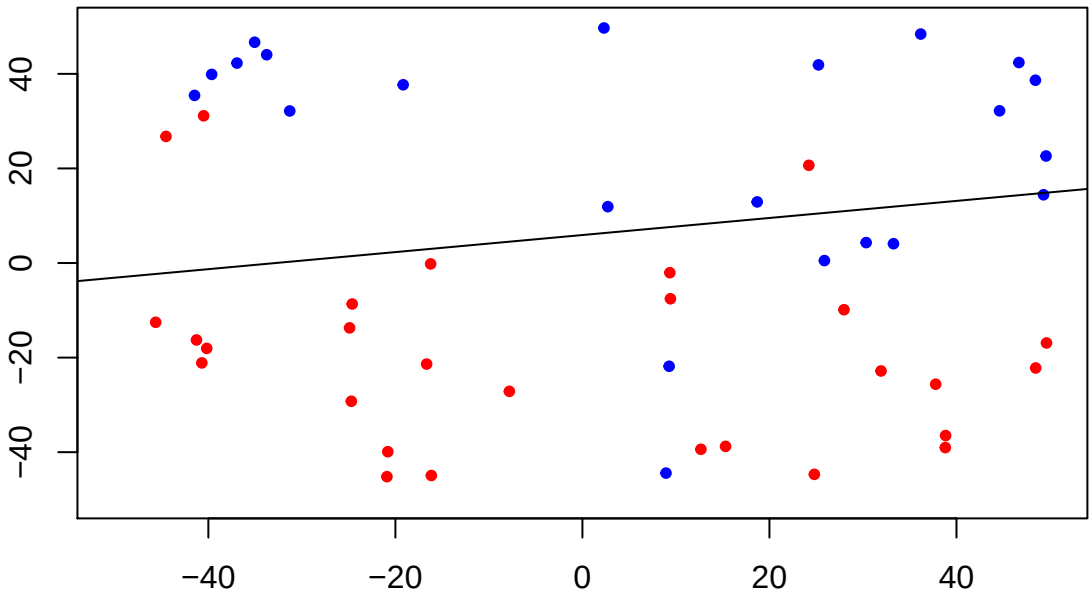
```
perceptron2_a <- ajusta_PLA(muestra2D, etiquetasRuido1, 10000, c(0,0,0))
plot(muestra2D, main = "PLA ruido para w = 0 y 10000 iters", col = etiquetasRuido1+3,
     xlim = c(-50,50), ylim = c(-50,50), xlab = "", ylab = "",pch=20)
abline(perceptron2_a[1],perceptron2_a[2])
```




```
cat("\nCon: ", 10000, " ->Num iters necesarias: ", perceptron2_a[3])
```

```
##
## Con:  10000  ->Num iters necesarias:  10001
```

```
perceptron2_a <- ajusta_PLA(muestra2D, etiquetasRuido1, 20000, c(0,0,0))
plot(muestra2D, main = "PLA ruido para w = 0 y 20000 iters", col = etiquetasRuido1+3, xlim = c(-50,50),
abline(perceptron2_a[1],perceptron2_a[2])
```

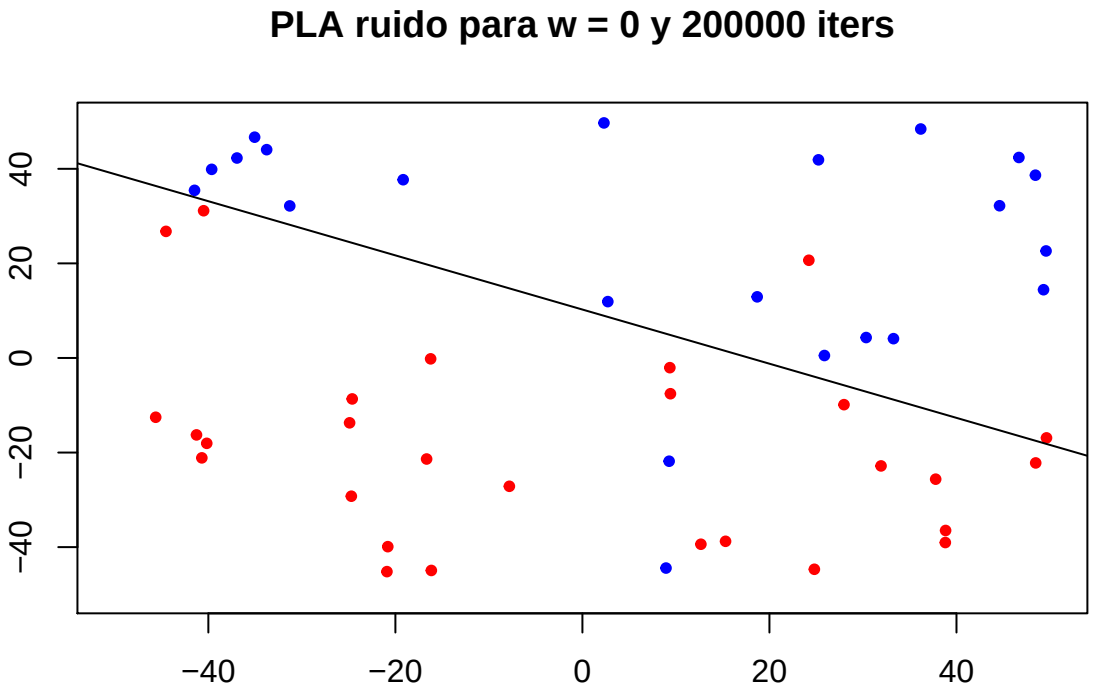



```
cat("\nCon: ", 20000, " ->Num iters necesarias: ", perceptron2_a[3])
```

```
##
```

```
## Con:  20000  ->Num iters necesarias:  20001
```

```
perceptron2_a <- ajusta_PLA(muestra2D, etiquetasRuido1, 200000, c(0,0,0))
plot(muestra2D, main = "PLA ruido para w = 0 y 200000 iters", col = etiquetasRuido1+3, xlim = c(-50,50)
abline(perceptron2_a[1],perceptron2_a[2])
```

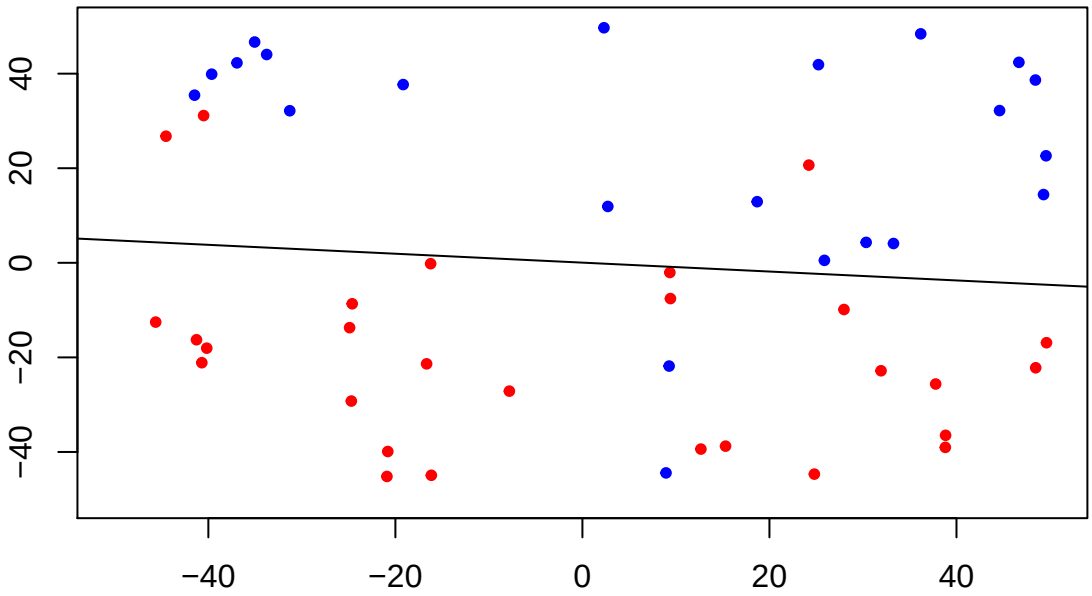



```
cat("\nCon: ", 200000, " ->Num iters necesarias: ", perceptron2_a[3])
```

```
##
## Con:  2e+05  ->Num iters necesarias:  200001
```

b) El vector w inicializado a valores aleatorios

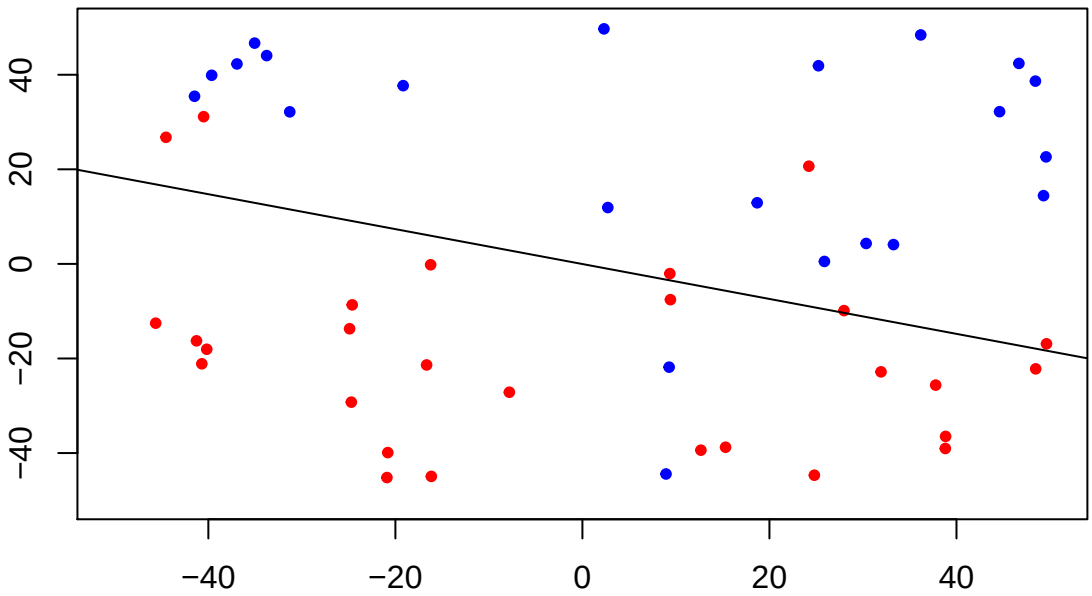
```
perceptron2_b <- ajusta_PLA(muestra2D, etiquetasRuido1, 10, runif(3,0,1))
plot(muestra2D, main = "PLA ruido para w Random y 10 iters", col = etiquetasRuido1+3,
     xlim = c(-50,50), ylim = c(-50,50), xlab = "", ylab = "",pch=20)
abline(perceptron2_b[1],perceptron2_b[2])
```




```
cat("\nCon: ", 10, " ->Num iters necesarias: ", perceptron2_b[3])
```

```
##
## Con:  10  ->Num iters necesarias:  51
```

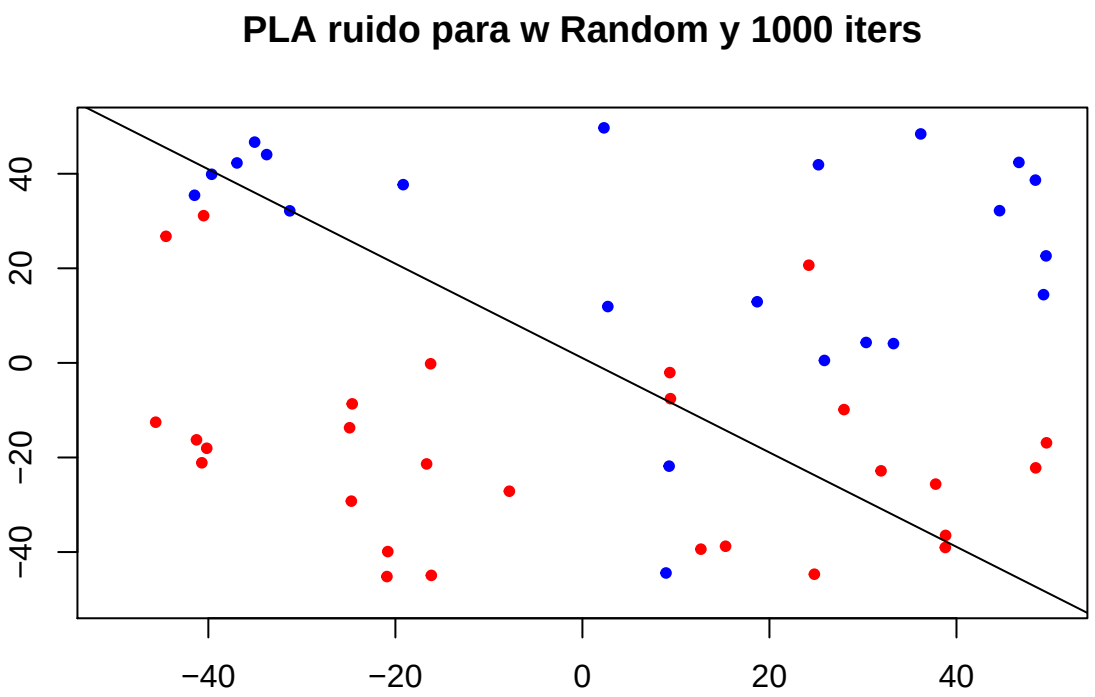
```
perceptron2_b <- ajusta_PLA(muestra2D, etiquetasRuido1, 100, runif(3,0,1))
plot(muestra2D, main = "PLA ruido para w Random y 100 iters", col = etiquetasRuido1+3,
     xlim = c(-50,50), ylim = c(-50,50), xlab = "", ylab = "",pch=20)
abline(perceptron2_b[1],perceptron2_b[2])
```



```
cat("\nCon: ", 100, " ->Num iters necesarias: ", perceptron2_b[3])
```

```
##
## Con:  100  ->Num iters necesarias:  101
```

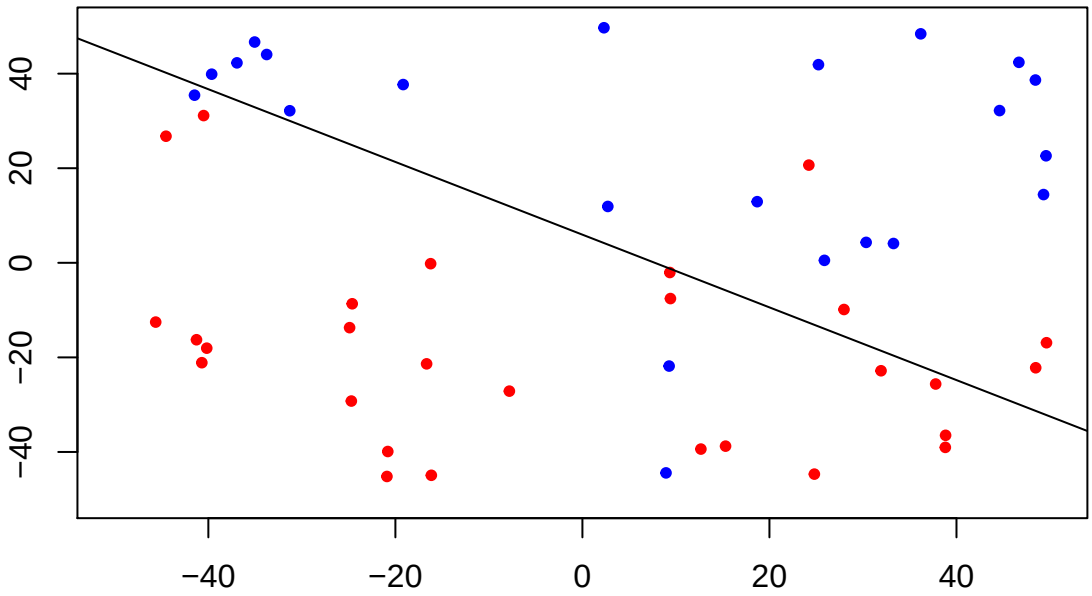
```
perceptron2_b <- ajusta_PLA(muestra2D, etiquetasRuido1, 1000, runif(3,0,1))
plot(muestra2D, main = "PLA ruido para w Random y 1000 iters", col = etiquetasRuido1+3,
     xlim = c(-50,50), ylim = c(-50,50), xlab = "", ylab = "",pch=20)
abline(perceptron2_b[1],perceptron2_b[2])
```




```
cat("\nCon: ", 1000, " ->Num iters necesarias: ", perceptron2_b[3])
```

```
##
## Con:  1000  ->Num iters necesarias:  1001
```

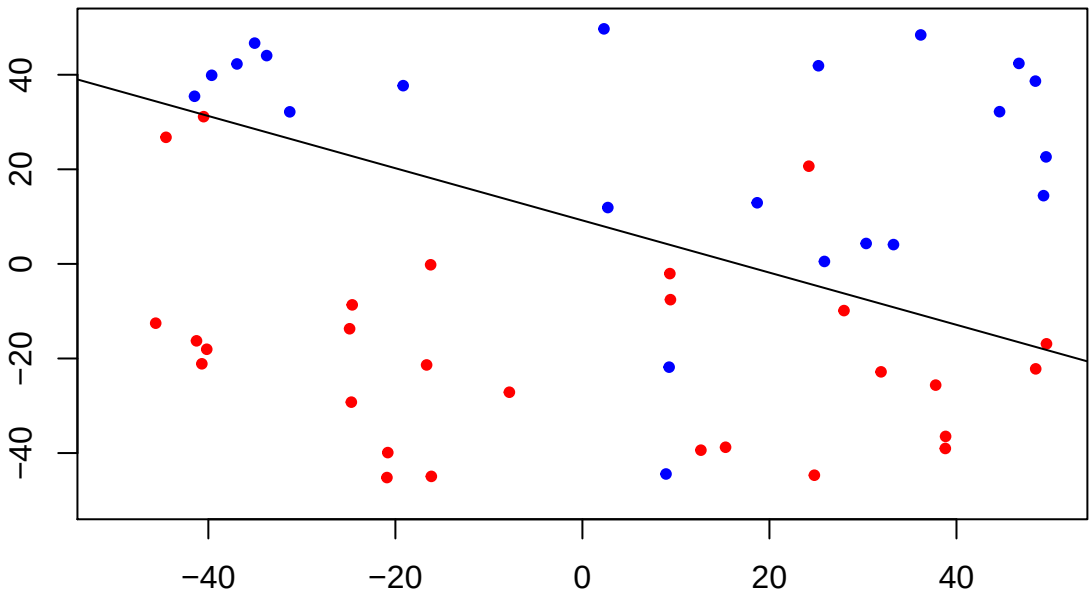
```
perceptron2_b <- ajusta_PLA(muestra2D, etiquetasRuido1, 10000, runif(3,0,1))
plot(muestra2D, main = "PLA ruido para w Random y 10000 iters", col = etiquetasRuido1+3,
     xlim = c(-50,50), ylim = c(-50,50), xlab = "", ylab = "",pch=20)
abline(perceptron2_b[1],perceptron2_b[2])
```




```
cat("\nCon: ", 10000, " ->Num iters necesarias: ", perceptron2_b[3])
```

```
##
## Con:  10000  ->Num iters necesarias:  10001
```

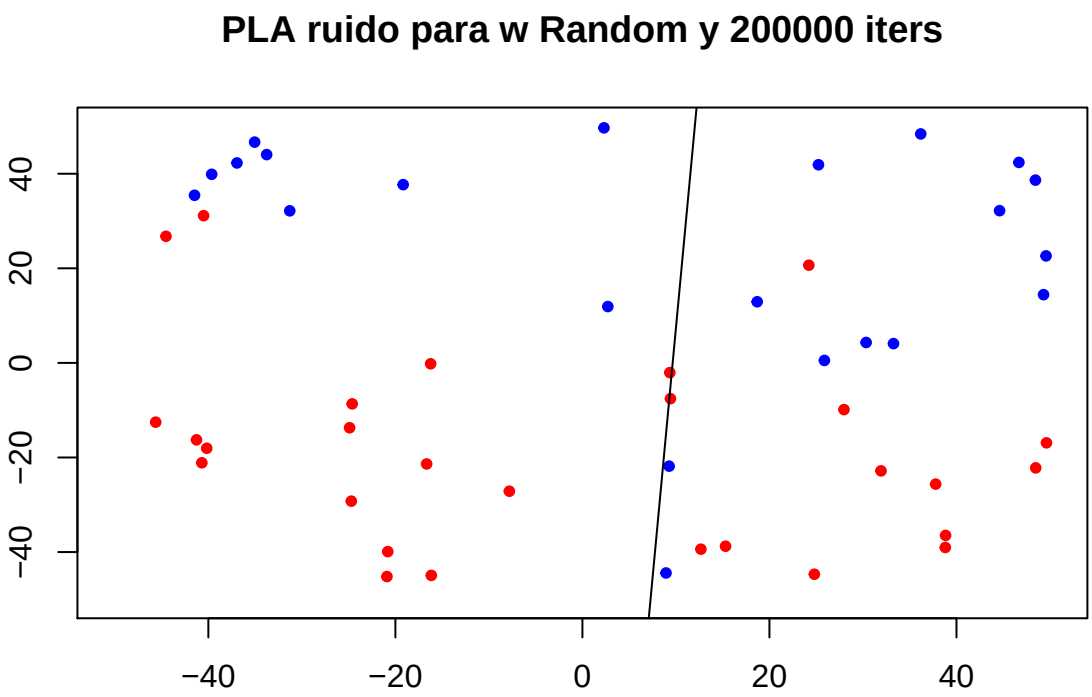
```
perceptron2_b <- ajusta_PLA(muestra2D, etiquetasRuido1, 20000, runif(3,0,1))
plot(muestra2D, main = "PLA ruido para w Random y 20000 iters", col = etiquetasRuido1+3,
     xlim = c(-50,50), ylim = c(-50,50), xlab = "", ylab = "",pch=20)
abline(perceptron2_b[1],perceptron2_b[2])
```



```
cat("\nCon: ", 20000, " ->Num iters necesarias: ", perceptron2_b[3])
```

```
##
## Con:  20000  ->Num iters necesarias:  20001
```

```
perceptron2_b <- ajusta_PLA(muestra2D, etiquetasRuido1, 200000, runif(3,0,1))
plot(muestra2D, main = "PLA ruido para w Random y 200000 iters", col = etiquetasRuido1+3,
     xlim = c(-50,50), ylim = c(-50,50), xlab = "", ylab = "",pch=20)
abline(perceptron2_b[1],perceptron2_b[2])
```




```
cat("\nCon: ", 200000, " ->Num iters necesarias: ", perceptron2_b[3])
```

```
##
## Con:  2e+05  ->Num iters necesarias:  200001
```

Para este apartado los resultados me parecen bastante malos, el algoritmo del perceptron le cuesta clasificar los datos con ruido y es entendible, no existe una recta que separe los datos en dos conjuntos linealmente separables por lo que siempre llegaremos a tener que iterar hasta el límite de iteracciones.

2.2 Regresión Logística: En este ejercicio crearemos nuestra propia función objetivo f (una probabilidad en este caso) y nuestro conjunto de datos D para ver cómo funciona regresión logística. Supondremos por simplicidad que f es una probabilidad con valores 0/1 y por tanto que la etiqueta y es una función determinista de x. Consideremos d = 2 para que los datos sean visualizables, y sea X = [0,2]×[0,2] con probabilidad uniforme de elegir cada x pertenece a X. Elegir una línea en el plano que pase por X como la frontera entre f(x)=1 (donde y toma valores +1) y f(x)=0 (donde y toma valores -1), para ello seleccionar dos puntos aleatorios del plano y calcular la línea que pasa por ambos. Seleccionar N=100 puntos aleatorios {x_n} de X y evaluar las respuestas {y_n} de todos ellos respecto de la frontera elegida.

Antes de implementar la Regresión Logística recuperamos la función norma que usamos en la práctica anterior para el SGD. Tras esto creamos los datos y de entrada como se piden, primero para dos puntos y luego para N puntos aleatorios.

```
# Función norma que usaremos para la RL
norma <- function(w_old, w_new){
  sqrt(sum((w_old - w_new)^2))
}

# Para clasificar dos puntos
dos_puntos = simula_unif(2,2,rango = c(0,2))
recta_2puntos = simula_recta(intervalo = c(0,2))
etiquetas_2puntos = sign(dos_puntos[,2]-recta_2puntos[1]*
                           dos_puntos[,1] -recta_2puntos[2])

# Para dos puntos
n_puntos = simula_unif(100,2,c(0,2))
recta_Npuntos = simula_recta(intervalo = c(0,2))
etiquetas_Npuntos = sign(n_puntos[,2]-recta_Npuntos[1]*
                           n_puntos[,1] -recta_Npuntos[2])
```

a) Implementar Regresión Logística(RL) con Gradiente Descendente Estocástico (SGD) bajo las siguientes condiciones: 1) Inicializar el vector de pesos con valores 0. 2) Parar el algoritmo cuando ||w^(t-1) - w^(t) || < 0,01, donde w (t) denota el vector de pesos al final de la época t. Una época es un pase completo a través de los N datos. 3) Aplicar una permutación aleatoria, 1,2,...,N, en el orden de los datos antes de usarlos en cada época del algoritmo. Usar una tasa de aprendizaje de mu = 0,01

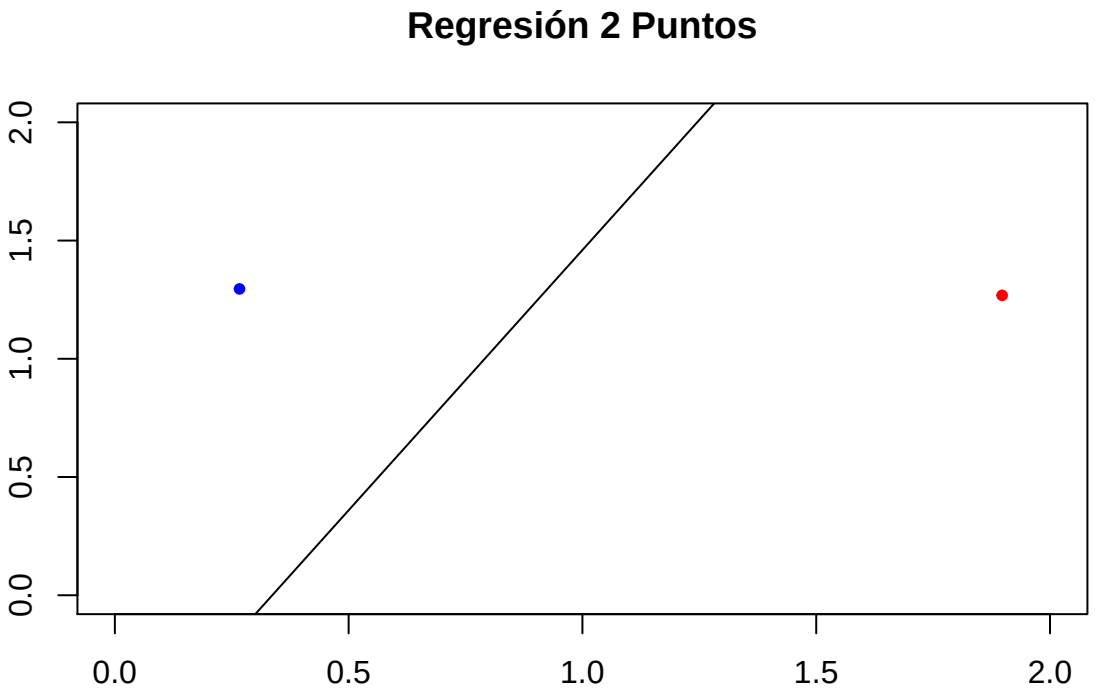
Implementamos la RL con los parámetros nombrados del apartado:

```
# Implementación de RL
RL <- function(datos,etiquetas,vini = c(0,0,0), mu = 0.01, max_iter = 500){
  # Inicializamos los pesos.
  w_nuevo <- vini
  w_viejo <- c(0,0,0)
  # Añadimos una columna para el calculo  y el cnt de iters
  datos <- cbind(1,datos)
  iters <- 1
  seguir = TRUE
  # Mientras no se llegue al limite de iters
```

```
  while (iters <= max_iter | seguir) {
    # Actualizamos el peso y permutamos las etiquetas
    w_viejo <- w_nuevo
    permutacion <- sample(1:length(etiquetas))
    # Por cada itetiqueta calculamos el valor que tendrá el SGD
    # y actualizamos el peso con el nuevo valor
    for (i in permutacion) {
      SGD <- (-etiquetas[i]*datos[i,]) /
        (1 + exp(etiquetas[i]*w_nuevo %*% datos[i,]))
      w_nuevo <- w_nuevo - mu * SGD
    }
    iters = iters +1

    # Si se mejora maramos
    if(norma(w_viejo,w_nuevo) < mu){
       seguir <- FALSE
    }
  }
  # Devolvemos el hiperplano
  c(-w_nuevo[1]/w_nuevo[3], -w_nuevo[2] / w_nuevo[3])
}
```

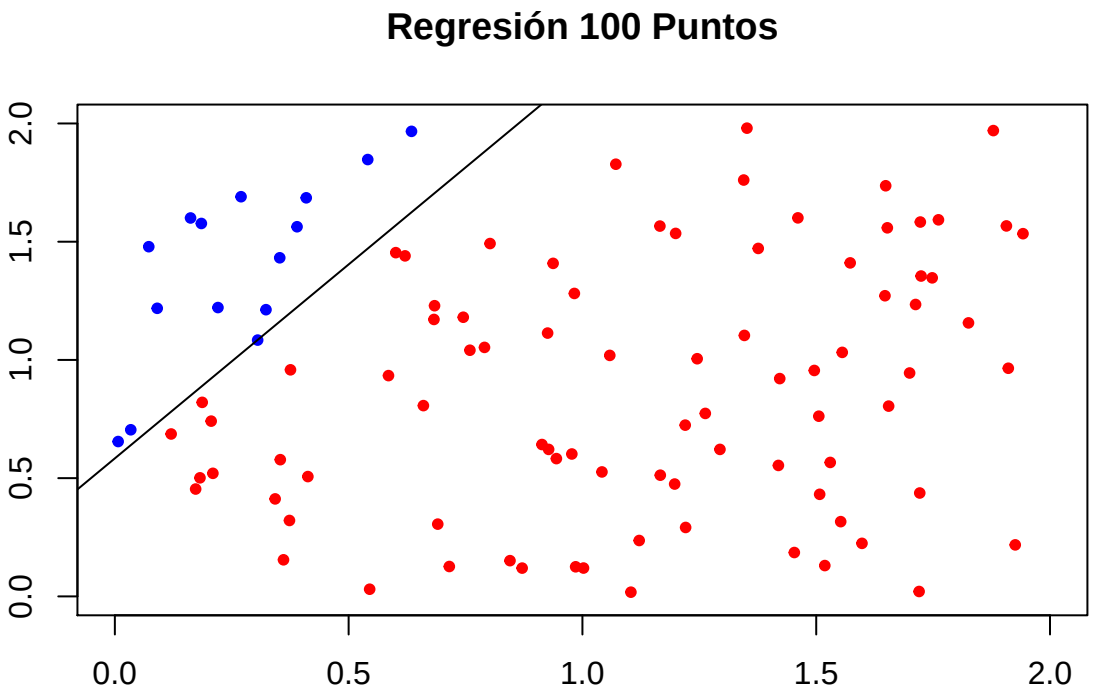
Una vez implementada la RL podemos hacer uso de ella con los datos anteriormente creados, primero para los dos puntos:

```
recta_RL <- RL(dos_puntos,etiquetas_2puntos,c(0,0,0), 0.01, 500)

plot(dos_puntos, main = "Regresión 2 Puntos", pch=20, col = etiquetas_2puntos+3,
     xlab = "", ylab = "",xlim = c(0,2), ylim = c(0,2))

abline(recta_RL[1],recta_RL[2])
```



Ahora lo hacemos para N = 100:

```
recta_RL2 <- RL(n_puntos,etiquetas_Npuntos,c(0,0,0), 0.01, 500)

plot(n_puntos, main = "Regresión 100 Puntos", pch=20, col = etiquetas_Npuntos+3,
     xlab = "", ylab = "",xlim = c(0,2), ylim = c(0,2))

abline(recta_RL2[1],recta_RL2[2])
```



b) Usar la muestra de datos etiquetada para encontrar nuestra solución g y estimar E_out usando para ello un número suficientemente grande de nuevas muestras (>999).

Lo primero que hacemos es generar una serie de datos para hacer este ejercicio en concreto con un N = 1000, usando las funciones ya vistas. Despues lo que hacemos es apreder con RL sobre un conjunto de 1000 datos. Posteriormente creamos otro conjunto de datos de prueba y probamos la recta generada con RL y obtendremos el E_out.

```
datos3 <- simula_unif(1000,2,c(0,2))
recta3 <- simula_recta(c(0,2))
etiquetas3 <- sign(datos3[,2]-recta3[1]*datos3[,1]-recta3[1] -recta3[2])

regresion <- RL(datos3,etiquetas3)

etiquetas_regresion <- sign(datos3[,2]-regresion[1]*datos3[,1]
                            -regresion[2])



E_in <- 0
E_in <- sum(etiquetas3 != etiquetas_regresion) / length(etiquetas3)
E_in
```

```
## [1] 0.018
```

```
datos3.test <- simula_unif(1000,2,c(0,2))
recta3.test <- simula_recta(c(0,2))
etiquetas3.test <- sign(datos3.test[,2]-recta3.test[1]*
                    datos3.test[,1]-recta3.test[1] -recta3[2])

etiquetas_regresion.test <- sign(datos3.test[,2]-regresion[1]
                           *datos3.test[,1]-regresion[2])

E_out <- 0
E_out <- sum(etiquetas3.test != etiquetas_regresion.test) / length(etiquetas3)
E_out
```

```
## [1] 0.616
```