Aprendizaje Automático: Memoria Práctica 2

Óscar López Arcos. 75571640

1. Ejercicio sobre la complejidad de H y el ruido

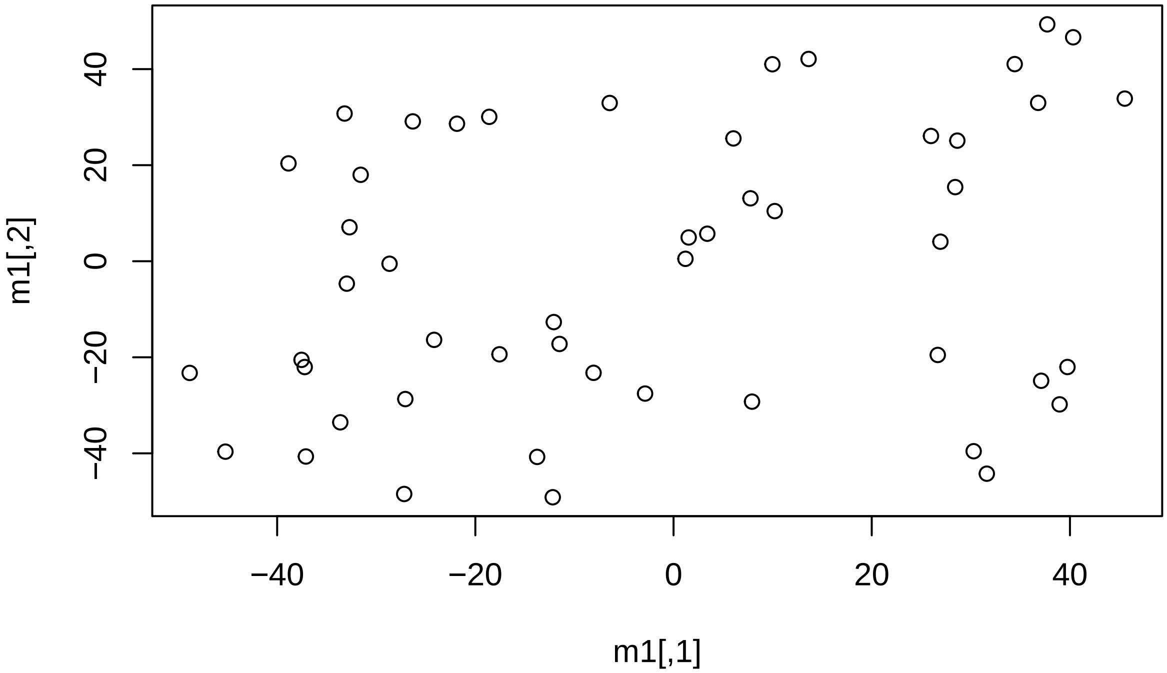
Funciones programadas:

```
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=6)  #pinta en verde los puntos
    abline(b,a,col=6)   # y la recta
  }
  c(a,b) # devuelve el par pendiente y punto de corte
}

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
                                  a la dimension")

  simula_gauss1 = function() rnorm(dim, sd = sigma) # genera 1 muestra, con
                                                    # las desviaciones especificadas
  m = t(replicate(N,simula_gauss1())) # repite N veces, simula_gauss1
                                      # y se hace la traspuesta
  m
}

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
}
```

1. Dibujar las gráficas de las siguientes nubes de puntos

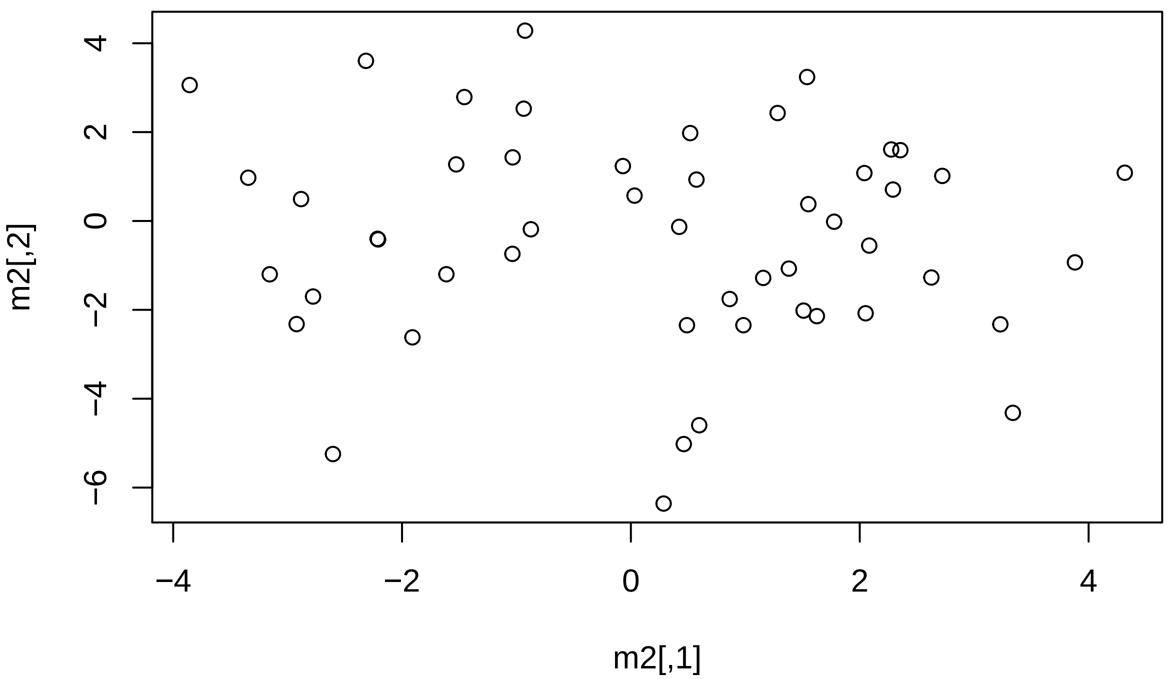
a) $N = 50, dim = 2, rango = [-50, +50]$ **con simula_unif**

```
set.seed(3)

m1 <- simula_unif(50,2,c(-50,50))
plot(m1)
```



b) $N = 50, dim = 2, rango = [5, 7]$ **con simula_gaus**

```
m2 <- simula_gaus(50,2,c(5,7))
plot(m2)
```

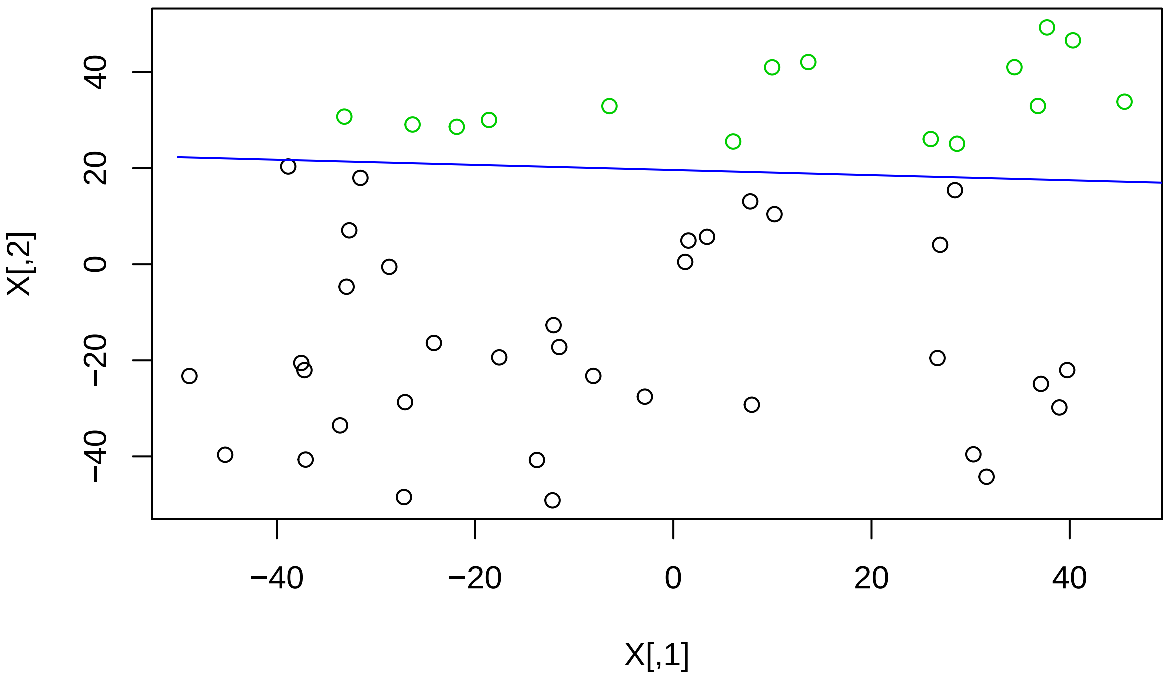


2. Generar con simula_unif una muestra 2D, etiquetada con el signo de la función $f(x,y) = y - ax - b$, es decir, el signo de la distancia de cada punto a la recta simulada con simula_recta

```
set.seed(3)
X <- simula_unif(50,2,c(-50,50))

#Creo función f
ab <- simula_recta(c(-50,50))

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

#Guardo en etiquetas el resultado de f
evaluacion <- f(X[,1],X[,2],ab)
etiquetas <- sign( evaluacion )
```

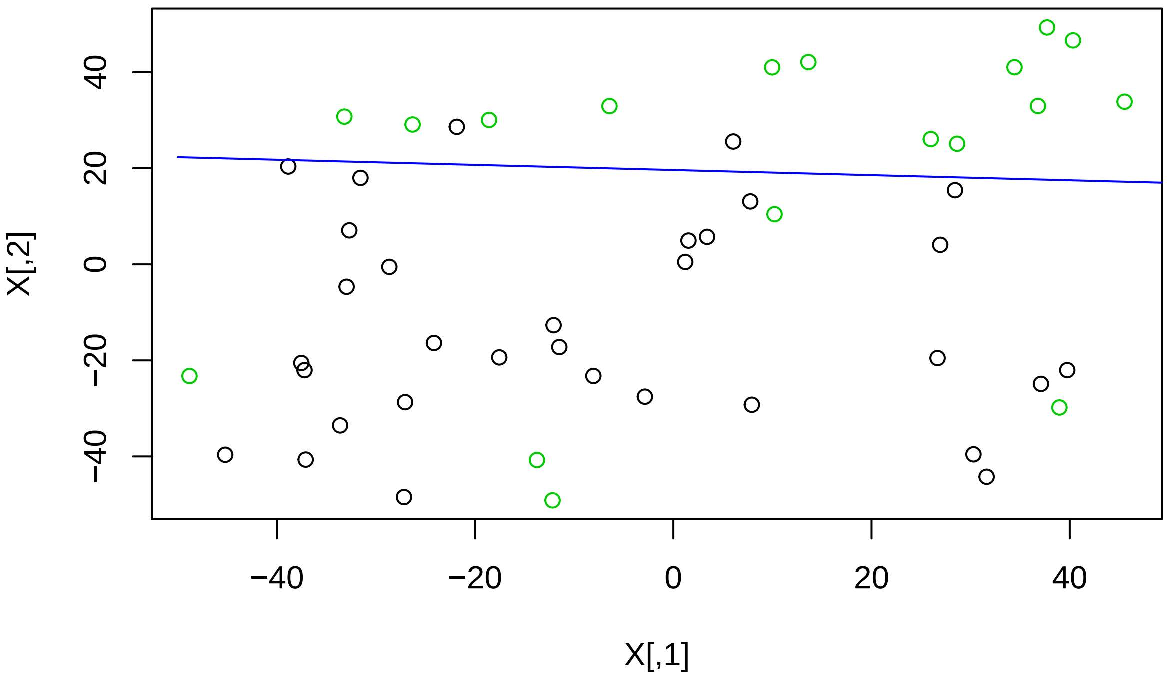
a) Dibujar una gráfica con los puntos y la recta usada para clasificarlos

```
plot(X, col=etiquetas+2)

# Devuelve la coordenada y
recta <- function(x,a,b){
  y = a*(x)+b
}

pintar_recta <- function(x1=-50,x2=50, ab, col="blue"){

  y1 = recta(x1,ab[1],ab[2])

  y2= recta(x2,ab[1],ab[2])

  lines(c(x1,x2),c(y1,y2), col=col,type="l")
}

pintar_recta(ab=ab)
```



b) Modificar un 10% de etiquetas positivas y 10% de negativas y volver a representar la gráfica anterior.

```
# ruido
noise <- function(label, p){
  result <- label*sample(c(1, -1), size=length(label), replace=TRUE, prob= c(1-p,p))
  result
}

set.seed(2)

# Alterar 10% positivas y 10% negativas
alterar_10 <- function(etiquetas, noise){
  etiquetas_aux <- etiquetas
  etiquetas_aux[etiquetas == 1] <- noise(etiquetas[etiquetas == 1], 0.1)
  etiquetas_aux[etiquetas == -1] <- noise(etiquetas[etiquetas == -1], 0.1)
  etiquetas_aux
}

etiquetas_modificadas <- alterar_10(etiquetas, noise)
plot(X, col=etiquetas_modificadas+2)
pintar_recta(ab=ab)
```



3. Visualizar el etiquetado generado en 2b con cada una de las gráficas de las siguientes funciones:

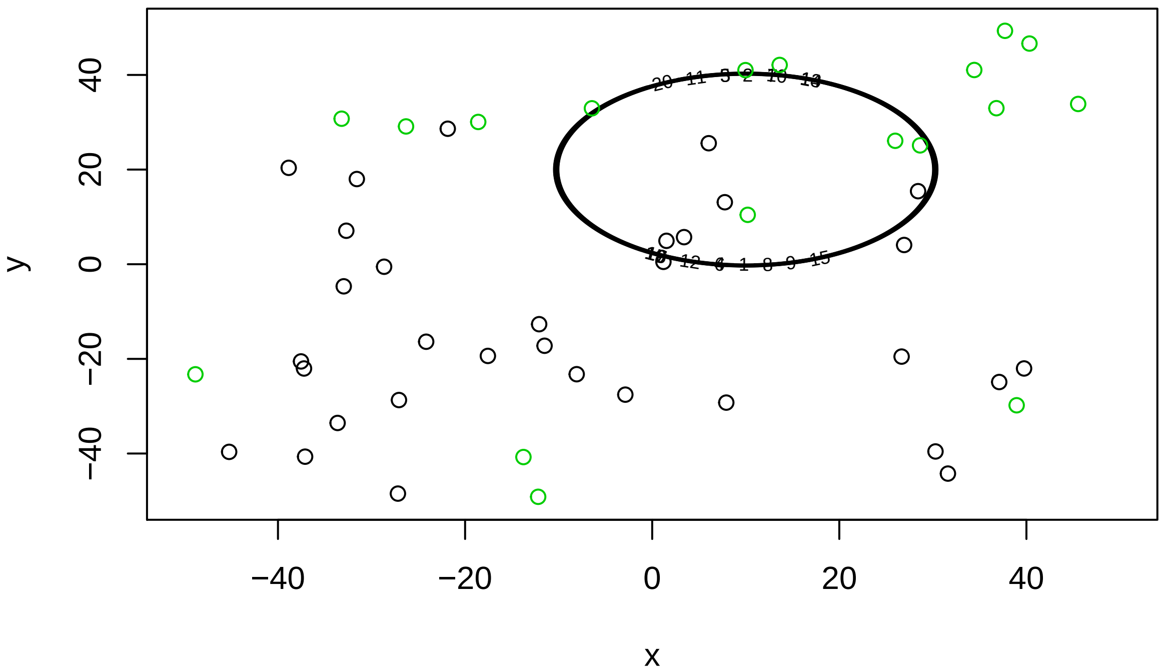
Funciones:

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

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

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

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

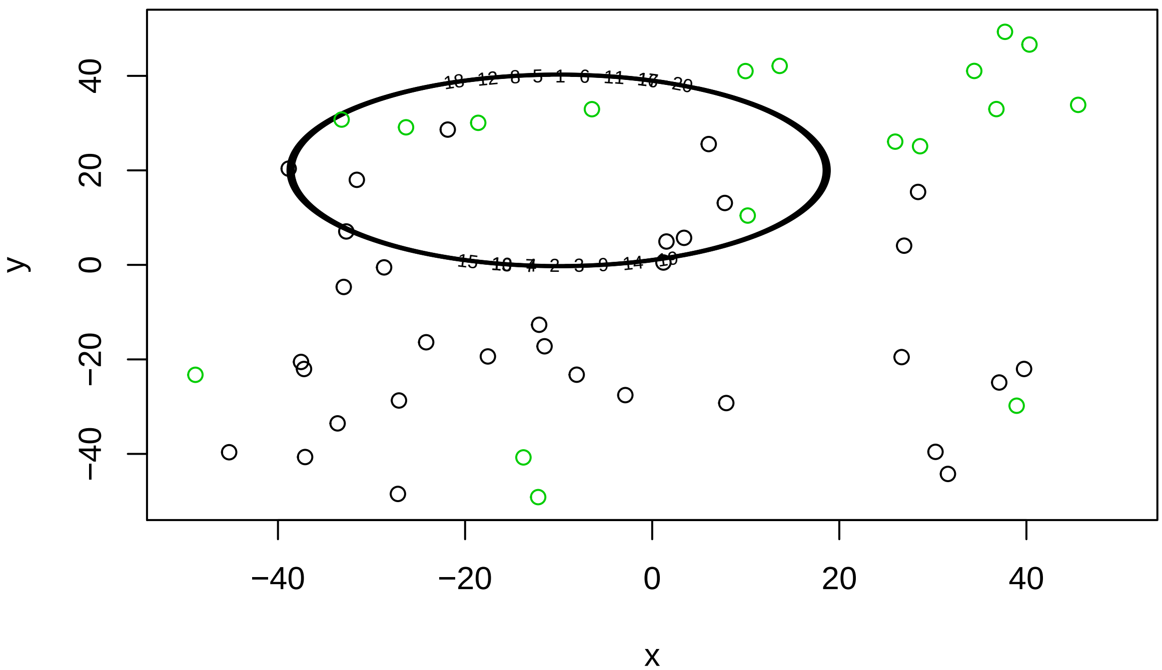
Etiquetado:

```
pintar_frontera = function(f,rango=c(-50,50)) {
  x=y=seq(rango[1],rango[2],length.out = 500)
  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 = 1:20, xlim =rango, ylim=rango, xlab = "x", ylab = "y")
}
```

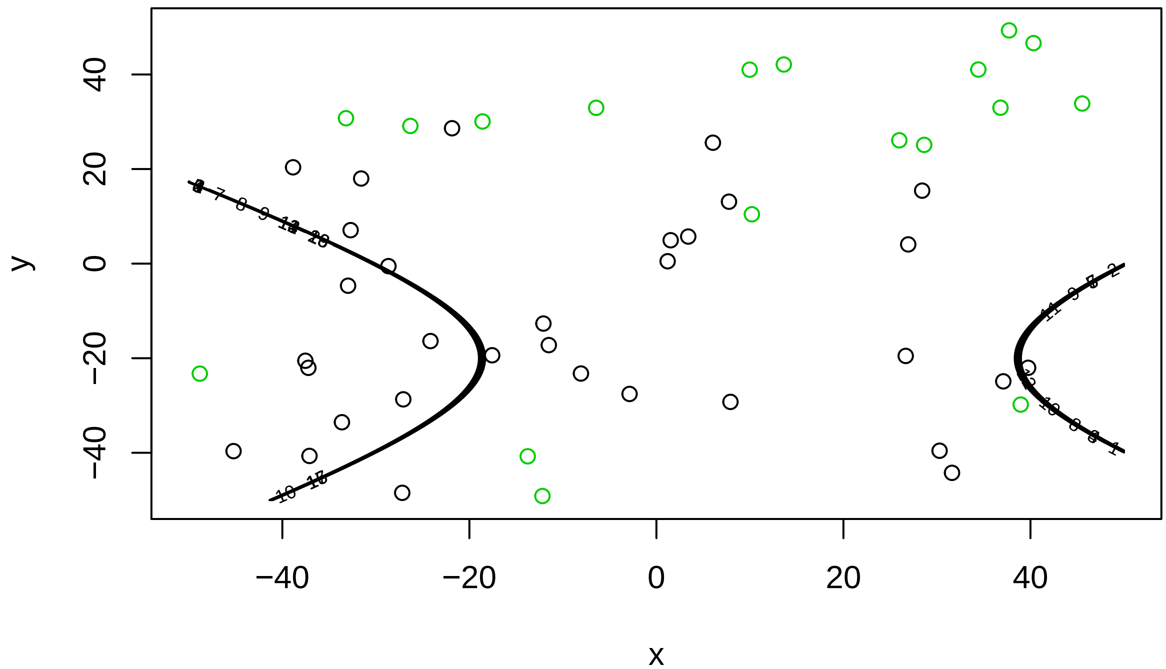
```r
pintar_frontera(f1)
points(X, col=etiquetas_modificadas+2)
```



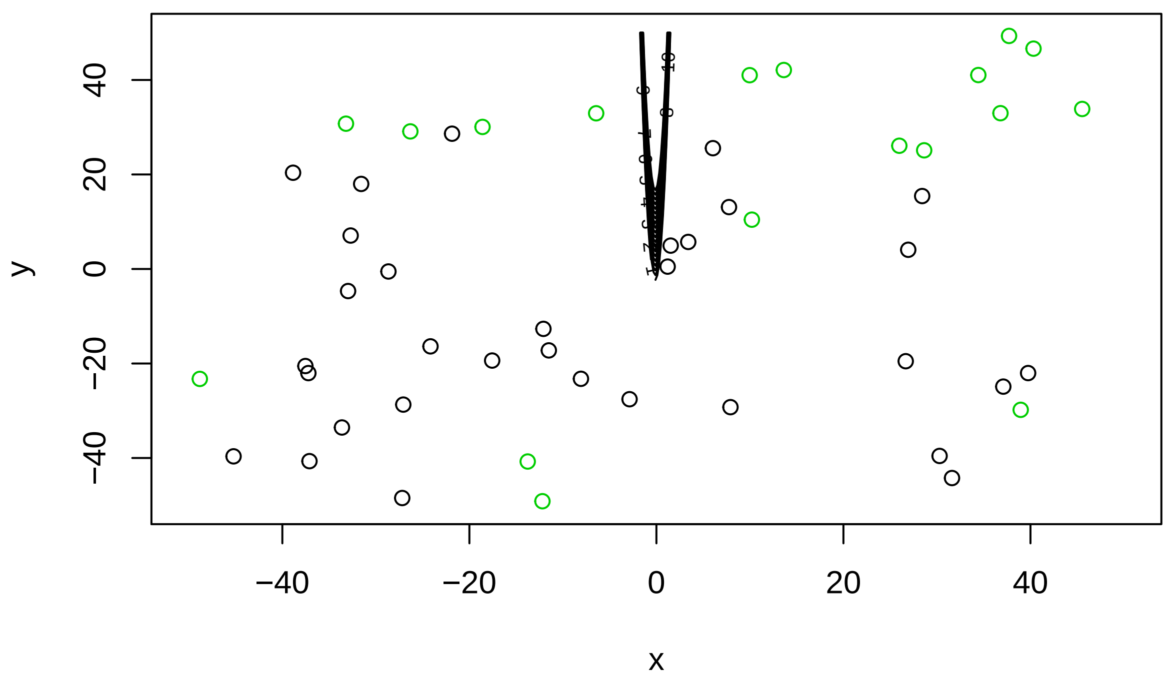
```r
pintar_frontera(f2)
points(X, col=etiquetas_modificadas+2)
```



```r
pintar_frontera(f3)
points(X, col=etiquetas_modificadas+2)
```




```r
pintar_frontera(f4)
points(X, col=etiquetas_modificadas+2)
```



Estas funciones representan diferentes secciones cónicas que, aun siendo más complejas que la recta, no necesariamente han de clasificar mejor. Esto dependerá del problema y de la distribución que los datos-etiquetas tengan en el plano. Sin embargo, sí es cierto que ganan a la función lineal en que el conjunto de datos no ha de ser separable por un hiperplano para poder clasificarse correctamente.

2. Modelos Lineales

1. Algoritmo Perceptron: Implementar la función ajusta_PLA(datos,label,max_iter,vini)

a) Ejecutar el algoritmo PLA con los datos simulados en el apartado 1-2a. Comentar el resultado.

Perceptron:

```
ajusta_PLA <- function(datos, label, max_iter, vini){
  it = 0
  change = TRUE
  datos <- cbind(datos,1)
  while(change & max_iter > it){

    change = FALSE

    for(i in 1:length(label)){

      if( sign(datos[i,] %*% vini ) != label[i] ){
        vini <- vini + label[i]%*%datos[i,]
        vini <- c(vini)
        change = TRUE
      }
    }
    it = it+1
  }
  list(w=vini,iteraciones=it)
}
```

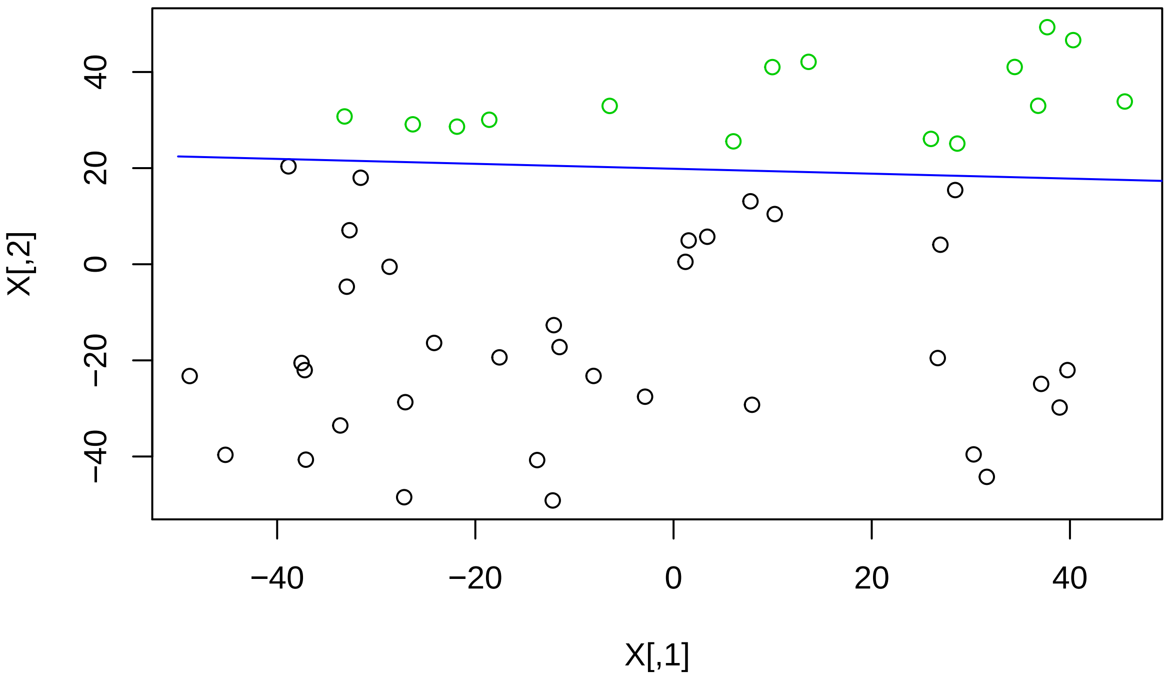
Función para pasar a recta los pesos:

```
pasoARecta= function(w){
  if(length(w)!= 3)
    stop("Solo tiene sentido con 3 pesos")
  a = -w[1]/w[2]
  b = -w[3]/w[2]
  c(a,b)
}
```

Experimentos:

w inicializado a 0

```
vini <- sample(0,ncol(X)+1,TRUE)
resultado <- ajusta_PLA(X,etiquetas,10000,vini)
plot(X, col=etiquetas+2)
ab <- pasoARecta(resultado$w)
pintar_recta(ab=ab)
```

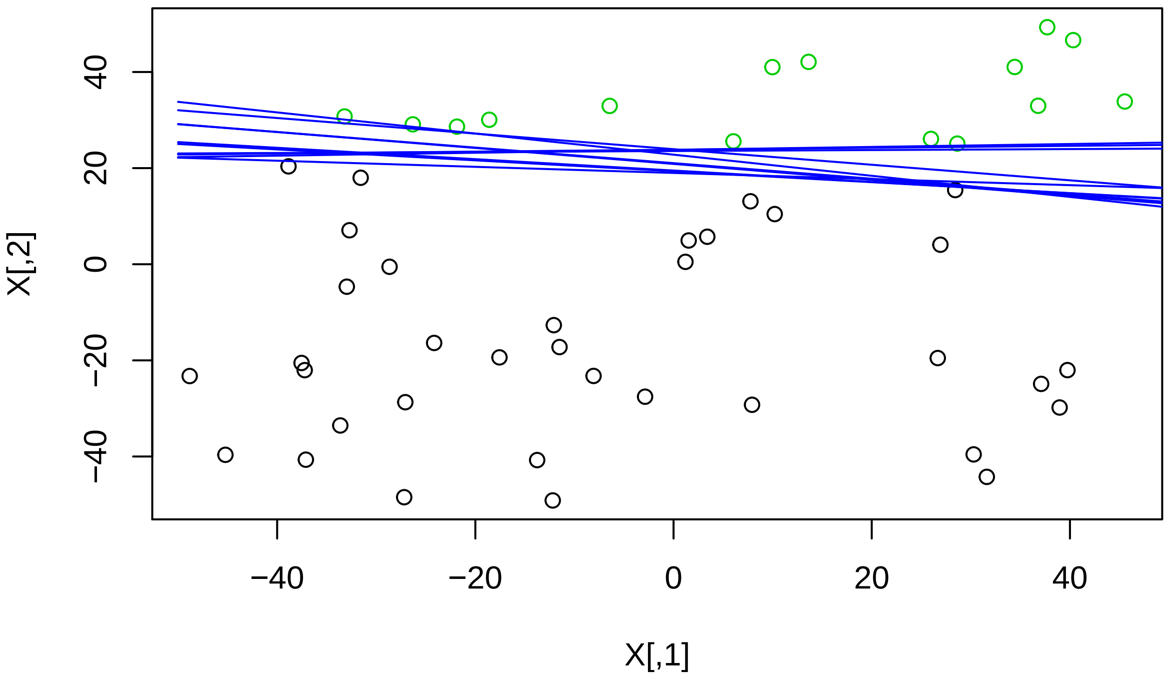



```r
sprintf("TOTAL ITERACIONES: %i", resultado$iteraciones)
```

```
## [1] "TOTAL ITERACIONES: 148"
```

w inicializado a números aleatorios en [0,1] (10 veces)

```r
set.seed(3)
plot(X, col=etiquetas+2)
iteraciones <- 1:10
for(i in 1:10){
  vini <- runif(ncol(X)+1, 0,1)
  resultado <- ajusta_PLA(X,etiquetas,10000,vini)
  ab <- pasoARecta(resultado$w)
  pintar_recta(ab=ab)
  iteraciones[i] = resultado$iteraciones
}
```




```
sprintf("MEDIA ITERACIONES: %f", sum(iteraciones)/length(iteraciones))
```

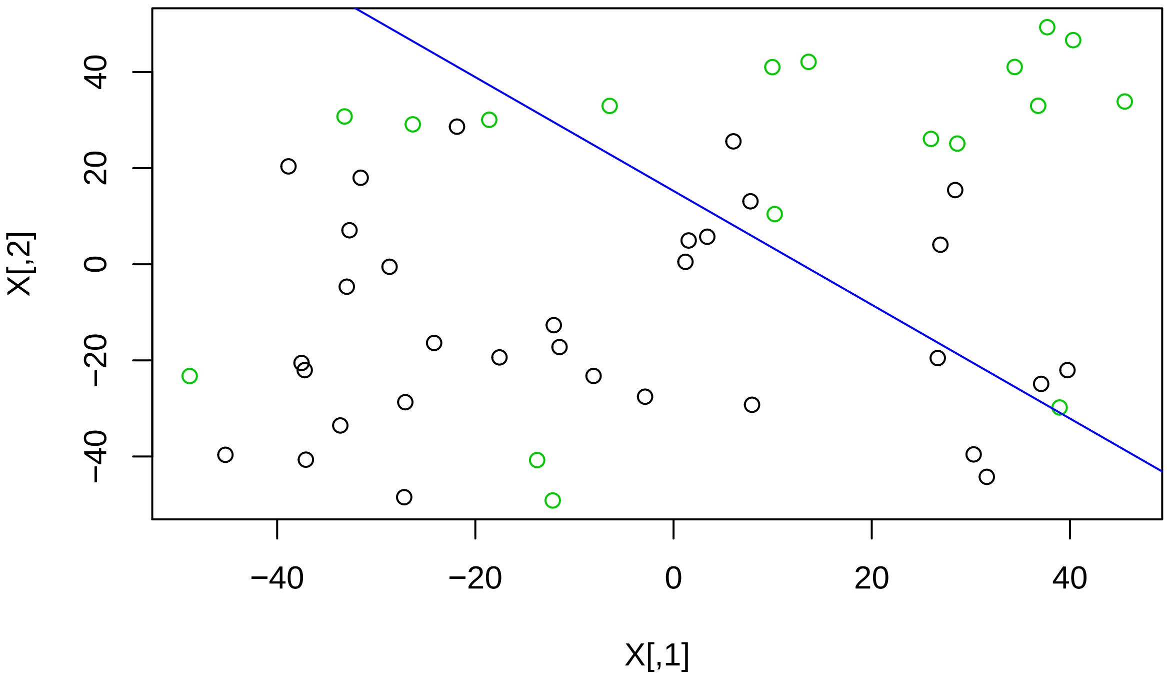
```
## [1] "MEDIA ITERACIONES: 144.400000"
```

El número de iteraciones es muy parecido en ambos casos. Para un *vini* inicializado a 0, conseguirá el óptimo en 148 iteraciones dado que los puntos son claramente separables. Para un *vini* aleatorio, en media, tardará un número de iteraciones parecido debido a la misma razón, aunque dependiendo del caso particular pueda tardar un poco más o un poco menos.

b) Hacer lo mismo usando los datos del apartado 1-2b. Comentar las diferencias

w inicializado a 0

```
vini <- sample(0,ncol(X)+1,TRUE)
resultado <- ajusta_PLA(X,etiquetas_modificadas,10000,vini)
plot(X, col=etiquetas_modificadas+2)
ab <- pasoARecta(resultado$w)
pintar_recta(ab=ab)
```

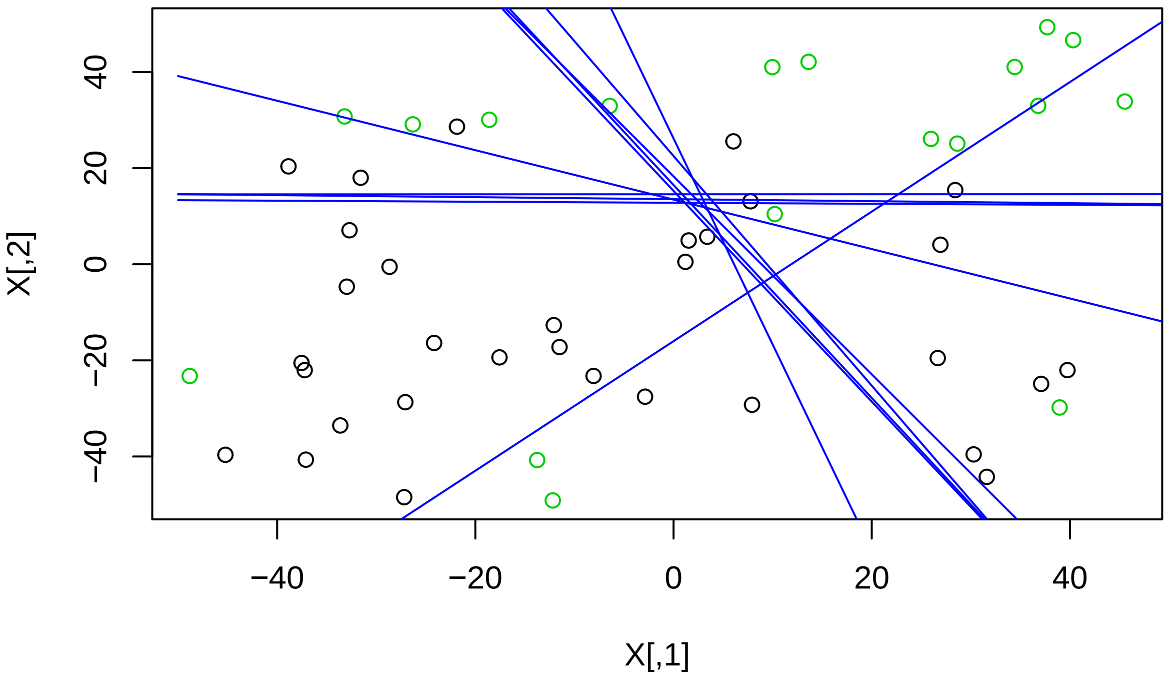



```r
sprintf("TOTAL ITERACIONES: %i", resultado$iteraciones)
```

```
## [1] "TOTAL ITERACIONES: 10000"
```

w inicializado a números aleatorios en [0,1] (10 veces)

```r
set.seed(1)
plot(X, col=etiquetas_modificadas+2)
iteraciones <- 1:10
for(i in 1:10){
  vini <- runif(ncol(X)+1, 0,1)
  resultado <- ajusta_PLA(X,etiquetas_modificadas,10000,vini)
  ab <- pasoARecta(resultado$w)
  pintar_recta(ab=ab)
  iteraciones[i] = resultado$iteraciones
}
```



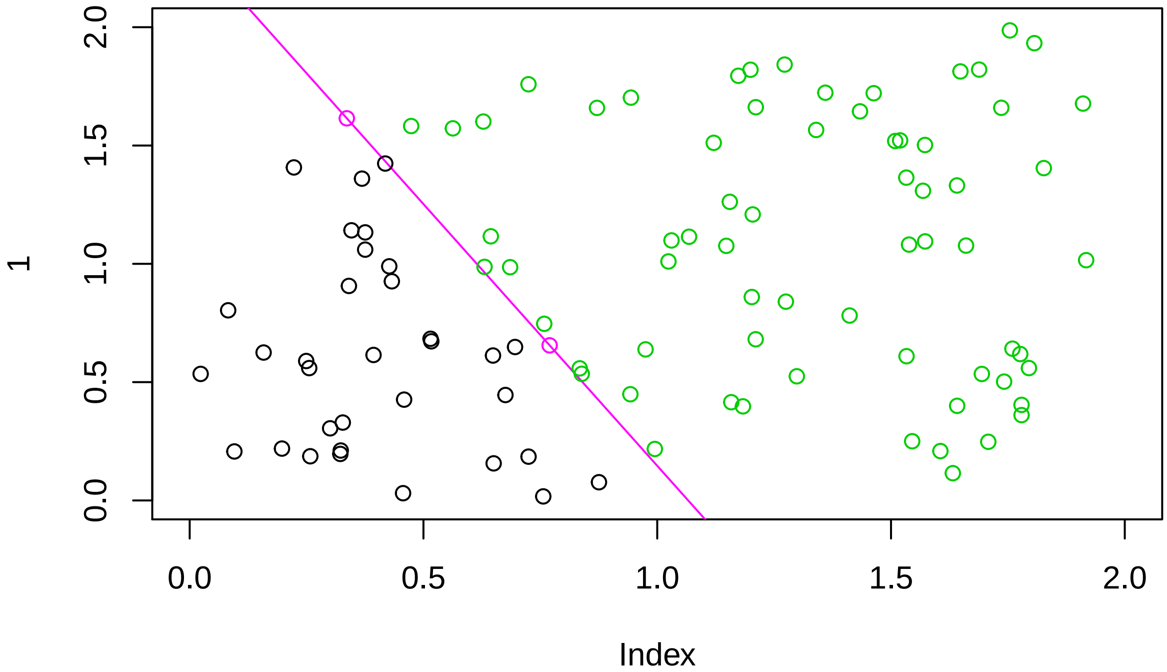

```
sprintf("MEDIA ITERACIONES: %f", sum(iteraciones)/length(iteraciones))
```

```
## [1] "MEDIA ITERACIONES: 10000.000000"
```

En este caso, al no ser clasificables por una recta, se llega al número máximo de iteraciones sin encontrar una solución.

2. Regresión Logística: Generar una recta aleatoria que cruce el espacio $X = [0, 2] \times [0, 2]$ y clasificar 100 puntos aleatorios respecto a la frontera elegida

```
set.seed(3)
plot(1, type="n", xlim=c(0,2), ylim=c(0,2))
ab <- simula_recta(c(0,2), T)

conjunto <- simula_unif(100,2,c(0,2))

# Funcion para etiquetar puntos respecto a una recta
# Por abajo de la recta -1, por encima 1
etiquetar_Recta <- function(conjunto,ab){

  etiquetas <- sample( -1,nrow(conjunto),T)
  y <- recta(conjunto[,1],ab[1],ab[2])
  etiquetas[conjunto[,2]>y] = 1
  etiquetas
}

etiquetas<-etiquetar_Recta(conjunto,ab)
points(conjunto, col=etiquetas+2)
```

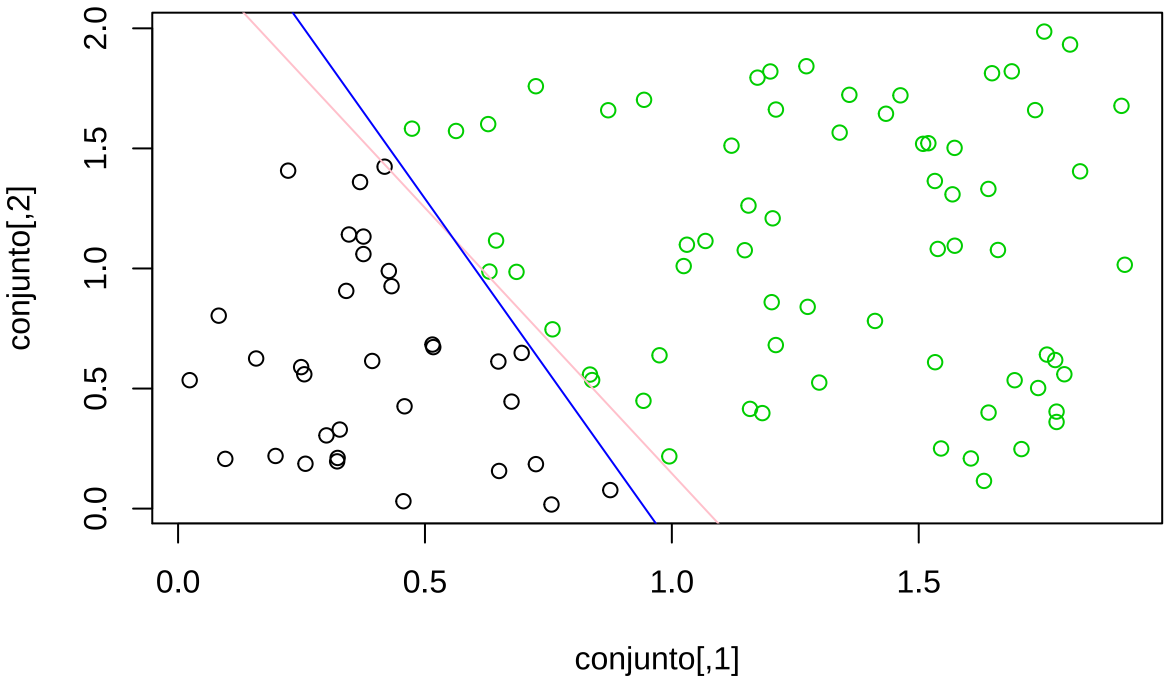


a) Implementar RLSGD

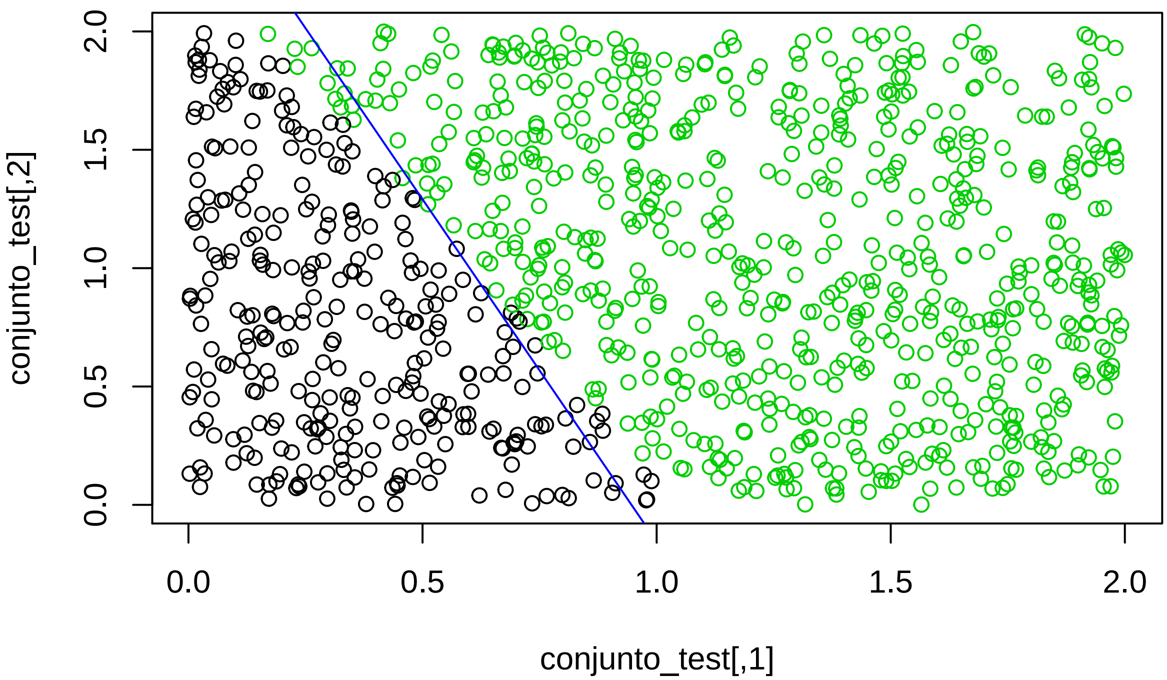
```
RLSGD <- function(X, Y, eta = 0.01, seed = 1){

  X = cbind(X,1)
  theta = sample(0,ncol(X),replace=TRUE)

  fin = FALSE
  indices = c(1:nrow(X))
  set.seed(seed)
  while(!fin){

    oldtheta = theta

    stochasticList <- sample(indices)

    for(i in stochasticList){
      theta = theta - eta*(-Y[i]*X[i,])/as.vector((1+exp(Y[i]*(theta%*%X[i,]))))
    }

    if( sqrt(sum((oldtheta - theta)^2)) < 0.01)
      fin = TRUE

  }
  set.seed(NULL)
  theta
}
```

b) Usar la muestra de datos etiquetada para encontrar una solución y estima E_{out} para una muestra >999

```
# Calculo solución
w <- RLSGD(conjunto, etiquetas)
ab2 <- pasoARecta(w)
plot(conjunto, col = etiquetas+2)
pintar_recta(ab=ab, col="pink") # Recta division etiquetas
pintar_recta(ab=ab2) # Recta gerada por RLSGD
```




```
set.seed(1)
# Clasificación de conjunto_test
conjunto_test <- simula_unif(N = 1000, dims = 2, rango = c(0,2))
etiquetas_test <- etiquetar_Recta(conjunto_test,ab)
plot(conjunto_test, col = etiquetas_test+2)
pintar_recta(ab=ab2)
```




```
# Error en conjunto_Test

sum(log(1+exp(-etiquetas_test*(t(w)%*%t(cbind(conjunto_test,1))))))/nrow(conjunto_test)
```

```
## [1] 0.1128503
```

Debido a que la muestra elegida para el entrenamiento ha resultado ser bastante representativa, el error en el conjunto_test es muy bajo. En definitiva, una buena clasificación.

BONUS

Clasificación de Dígitos. Extraer las características intensidad y simetría de las muestras proporcionadas para los dígitos 4 y 8.

1. Plantear un problema de clasificación binaria

Training:

```
# Leer datos
digit.train <- read.table("./datos/zip.train", quote="\"", comment.char="",
                          stringsAsFactors=FALSE)
```

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

```
digitos15.train = digit.train[digit.train$V1==4 | digit.train$V1==8,]
digitos = digitos15.train[,1]
ndigitos = nrow(digitos15.train)

grises = array(unlist(subset(digitos15.train,select=-V1)),c(ndigitos,16,16))

rm(digit.train)
```

```r
rm(digitos15.train)

# Definir simetría
fsimetria <- function(A){
  A = abs(A-A[,ncol(A):1])
  -sum(A)
}

# Cargar Intensidad y Simetría
intensidadtr <- apply(grises,1,mean)
simetriatr <- apply(grises,1,fsimetria)

#Guardar datos y etiquetas (X,Y)
datosTr = as.matrix(cbind(intensidadtr,simetriatr))
etiquetasTr = digitos
etiquetasTr[etiquetasTr==4]=-1 # Sustituir 4 por -1
etiquetasTr[etiquetasTr==8]=1  # Sustituir 8 por 1
etiquetasTr <- as.matrix(etiquetasTr)
```

Test:

```r
# Mismo proceso
digit.test <- read.table("./datos/zip.test", quote="\"",
                         comment.char="", stringsAsFactors=FALSE)
digitos15.test = digit.test[digit.test$V1==4 | digit.test$V1==8,]
digitos = digitos15.test[,1]    # vector de etiquetas del train
ndigitos = nrow(digitos15.test)

grises = array(unlist(subset(digitos15.test,select=-V1)),c(ndigitos,16,16))

rm(digit.test)
rm(digitos15.test)

intensidadtst <- apply(grises,1,mean)
simetriatst <- apply(grises,1,fsimetria)

datosTest = as.matrix(cbind(intensidadtst,simetriatst))
etiquetasTest = digitos
etiquetasTest[etiquetasTest==4]=-1
etiquetasTest[etiquetasTest==8]=1
etiquetasTest <- as.matrix(etiquetasTest)
```

2. Usar un modelo de Regresión Lineal y aplicar PLA-pocket

Funciones auxiliares:

```r
# Clasifica las muestras según w
Clasificar <- function(X,w){
  sign(X%*%w)
}
#Calcula el error
Err <- function(Yprima, Y, error){
  sum(error(Yprima,Y))/length(Y)
}
```

```r
#Error binario
binary_error <- function(Yprima, Y){
  error <- sample(0,length(Y),replace = T)
  error[ Yprima != Y ] = 1
  error
}
```

Algoritmos:

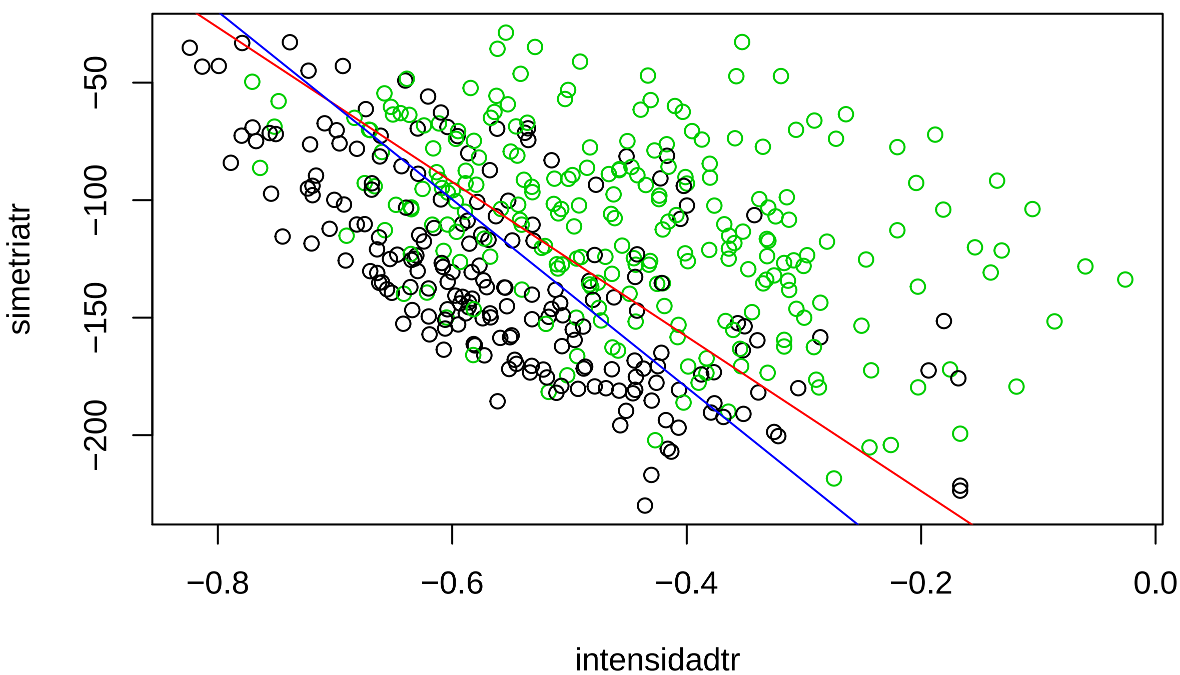
```r
# PLA Pocket

PLApocket <- function(datos, label, max_iter, vini){
  it = 0
  change = TRUE
  datos <- cbind(datos,1)

  best_w <- vini # Fijo mejor w
  Yprima <- Clasificar(datos, vini)
  best_Ein <- Err(Yprima, label, binary_error) # Fijo mejor Ein

  while(change & (max_iter > it)){

    change = FALSE

    for(i in 1:length(label)){

      if( sign(datos[i,] %*% vini ) != label[i] ){
        vini <- vini + label[i]%*%datos[i,]
        vini <- c(vini)
        change = TRUE
      }

    }
    Yprima <- Clasificar(datos, vini)
    Ein <- Err(Yprima, label, binary_error) # Calculo Ein actual

    if(Ein < best_Ein){ # Si Ein actual es mejor, sustituyo
      best_w = vini
      best_Ein = Ein
    }
    it = it+1
  }
  list(w=best_w,iteraciones=it)
}

# Pseudo Inversa

Pseudo_inversa <- function(X, Y){

  X = as.matrix(X)
  Y = as.matrix(Y)

  X <- cbind(X, 1) # cbind(X,1)
```

```
  x = t(X) %*% X
  pseudo = svd(x)
  aux = pseudo$v%*%diag(1/pseudo$d)%*%t(pseudo$v)
  pseudoinversa = aux%*%t(X)
  w = pseudoinversa%*%Y
  w
}
```

a) Generar gráficos en color de los datos y la función

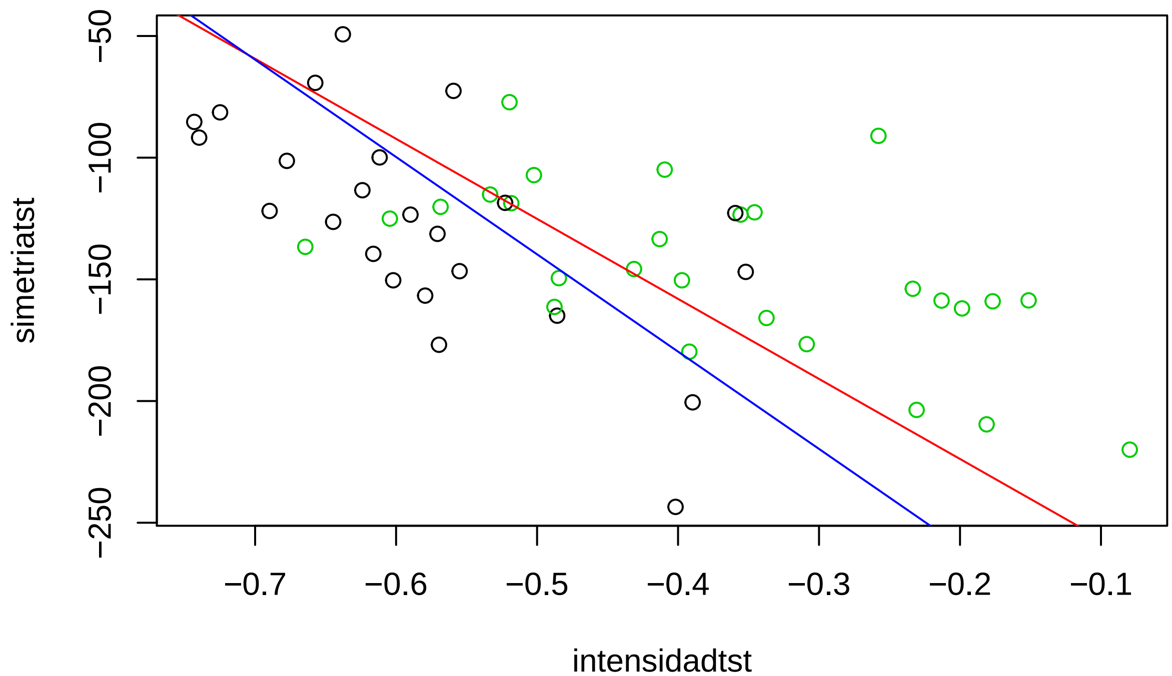
Training:

```
w_pseudo <- Pseudo_inversa(datosTr,etiquetasTr)
ab1<-pasoARecta(w_pseudo)

# Calculo PLAPocket
w <- PLApocket(datosTr,etiquetasTr,10000,c(w_pseudo))
ab2<-pasoARecta(w$w)

# Pinto soluciónes
plot(datosTr, col=etiquetasTr+2)
pintar_recta(x1=-50, x2=50,ab=ab1, col="red") # Rojo = PseudoInversa
pintar_recta(x1=-50, x2=50,ab=ab2) # Azul = PLA Pocket
```



Test:

```
plot(datosTest, col=etiquetasTest+2)
pintar_recta(x1=-50, x2=50,ab=ab1, col="red") # Rojo = PseudoInversa
pintar_recta(x1=-50, x2=50,ab=ab2) # Azul = PLA Pocket
```



b) Calcular E_{in} y E_{out}

E_{in} **para Pseudo Inversa y PLA Pocket**

```
X = cbind(datosTr,1)

Yprima <- Clasificar(X,w_pseudo)
Ein_p <- Err(Yprima,etiquetasTr,binary_error)
Ein_p # PseudoInversa
```

```
## [1] 0.2476852
```

```
Yprima <- Clasificar(X,w$w)
Ein <- Err(Yprima,etiquetasTr,binary_error)
Ein # PLA Pocket
```

```
## [1] 0.224537
```

E_{test} **para Pseudo Inversa y PLA Pocket**

```
X = cbind(datosTest,1)

Yprima <- Clasificar(X,w_pseudo)
Eout_p <- Err(Yprima,etiquetasTest,binary_error)
Eout_p # PseudoInversa
```

```
## [1] 0.2352941
```

```
Yprima <- Clasificar(X,w$w)
Eout <- Err(Yprima,etiquetasTest,binary_error)
Eout # PLA Pocket
```

```
## [1] 0.2156863
```

Después 10000 iteraciones de PLA Pocket observamos que la solución consigue mejorar un poco, aunque la relación mejora/iteraciones es bastante mala.

c) Calcular cotas sobre el verdadero valor de E_{out}

d_{vc} es igual al máximo de puntos que pueden clasificarse correctamente usando una sola recta si estos se encuentran en cualquier posición.

En el caso de etiquetas positivas y negativas, el máximo de puntos clasificables si estos se encuentran distribuidos de cualquier forma es igual a 3.

```r
# Cota para E_out(h)
bound <- function(Ein, d, N, dvc){
  Ein + sqrt( 8/N * log( 4*((2*N)^dvc +1)/d))
}

cota = bound(Ein, 0.05, nrow(datosTr), dvc=3)

cota
```

```
## [1] 0.9004006
```