Practica 2

Jose Antonio Ruiz Millan

17 de abril de 2018

Ejercicio sobre la complejidad de H y el ruido (6 puntos)

Antes de nada, vamos a definir las distintas funciones que vamos a utilizar en esta parte de la práctica.

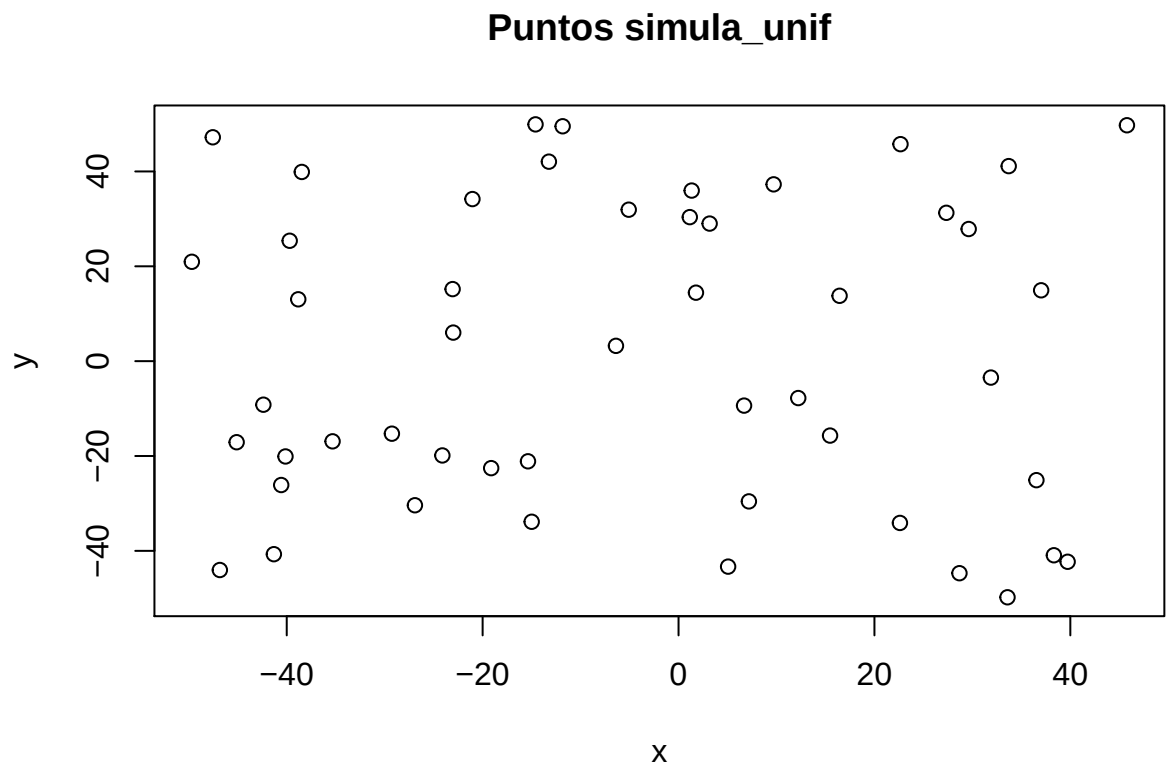
```
set.seed(331995)    # se establece la semilla

# por defecto genera 2 puntos entre [0,1] de 2 dimensiones
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 = function(N=2,dim=2,sigma){

  if (missing(sigma)) stop('Debe dar un vector de varianzas')
  # para la generación se usa sd, y no la varianza
  sigma = sqrt(sigma)
  if(dim != length(sigma)) stop ('El numero de varianzas es distinto de la dimensión')

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

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

  # se generan 2 puntos
   ptos = simula_unif(2,2,intervalo)
   # calculo de la pendiente
   a = (ptos[1,2] - ptos[2,2]) / (ptos[1,1]-ptos[2,1])
    # calculo del punto de corte
   b = ptos[1,2]-a*ptos[1,1]

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

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

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

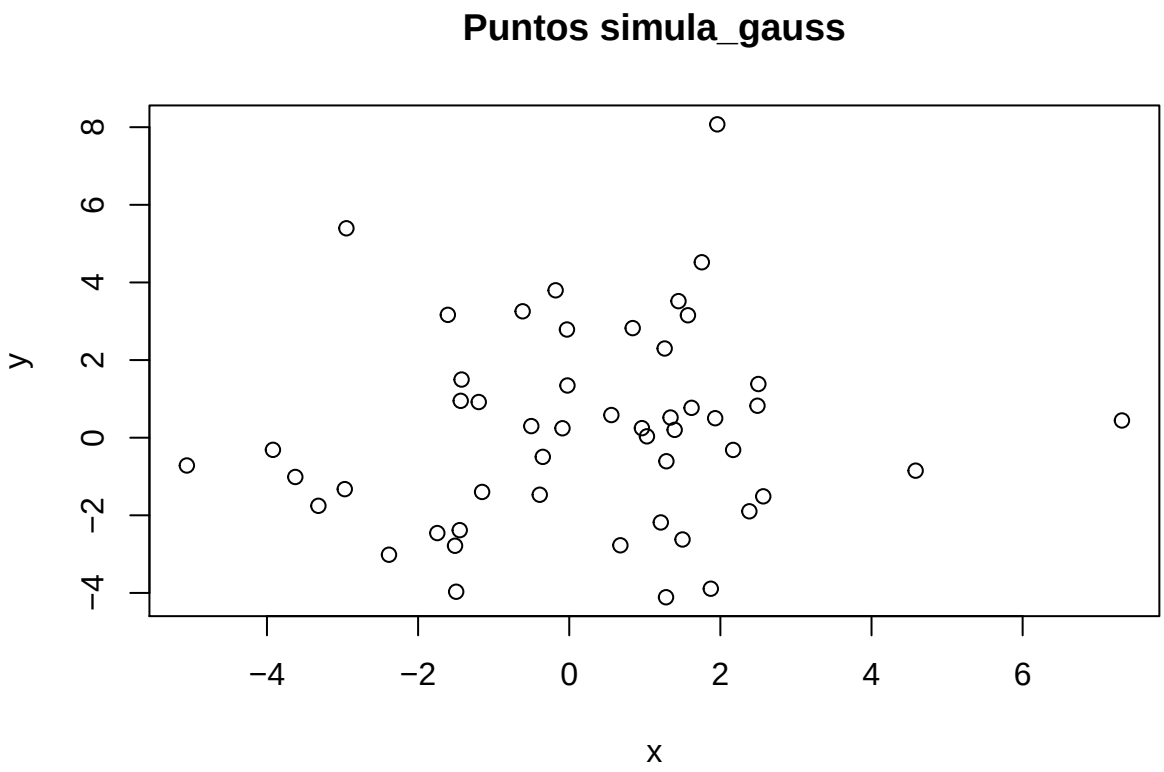
a) Considere N = 50, dim = 2, rango = [-50, +50] con simula_unif(N, dim, rango).

```
#Creamos la matriz llamando a la funcion con los parámetros que nos especifican
v_unif <- simula_unif(50,2,c(-50,50))

plot(v_unif[,1],v_unif[,2],type="p",xlab="x",ylab="y",main="Puntos simula_unif")
```



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

```
#Creamos la matriz pero ahora llamando a la funcion simula_gauss
v_gauss <- simula_gaus(50,2,c(5,7))

plot(v_gauss[,1],v_gauss[,2],type="p",xlab="x",ylab="y",main="Puntos simula_gauss")
```



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

Antes de nada, vamos a crear la funcion para facilitar el texto en los siguientes apartados.

```
funcion2 <- function(x,y,recta){
  sign(y-recta[1]*x-recta[2])
}
```

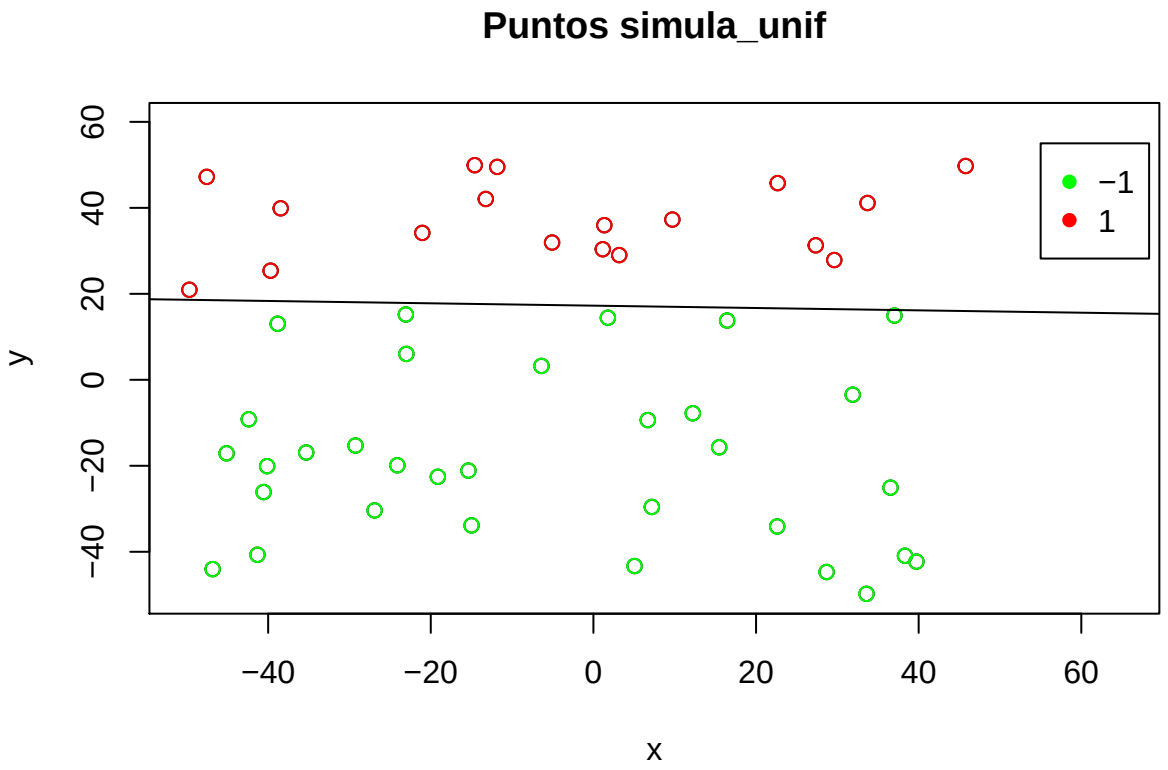
a) Dibujar una gráfica donde los puntos muestren el resultado de su etiqueta, junto con la recta usada para ello. (Observe que todos los puntos están bien clasificados respecto de la recta)

```
#Utilizaré la muestra generada en el apartado anterior.

#Creamos la recta
recta_unif <- simula_recta(c(-50,50))
recta_unif
```

```
## [1] -0.02734192 17.24771391
```

```
#Creamos las etiquetas de los puntos a traves de la recta
etiquetas_unif <- funcion2(v_unif[,1],v_unif[,2],recta_unif)

#Dibujamos el resultado
plot(v_unif[,1],v_unif[,2],type="p",xlab="x",ylab="y",main="Puntos simula_unif",xlim=c(-50,65),ylim=c(-5
points(v_unif[,1][etiquetas_unif==-1],v_unif[,2][etiquetas_unif==-1],col="green")
points(v_unif[,1][etiquetas_unif==1],v_unif[,2][etiquetas_unif==1],col="red")
abline(recta_unif[2],recta_unif[1],col="black")
```

```
legend(x= 55, y = 55, legend = c(-1,1), col = c("green","red"),pch=16)
```



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

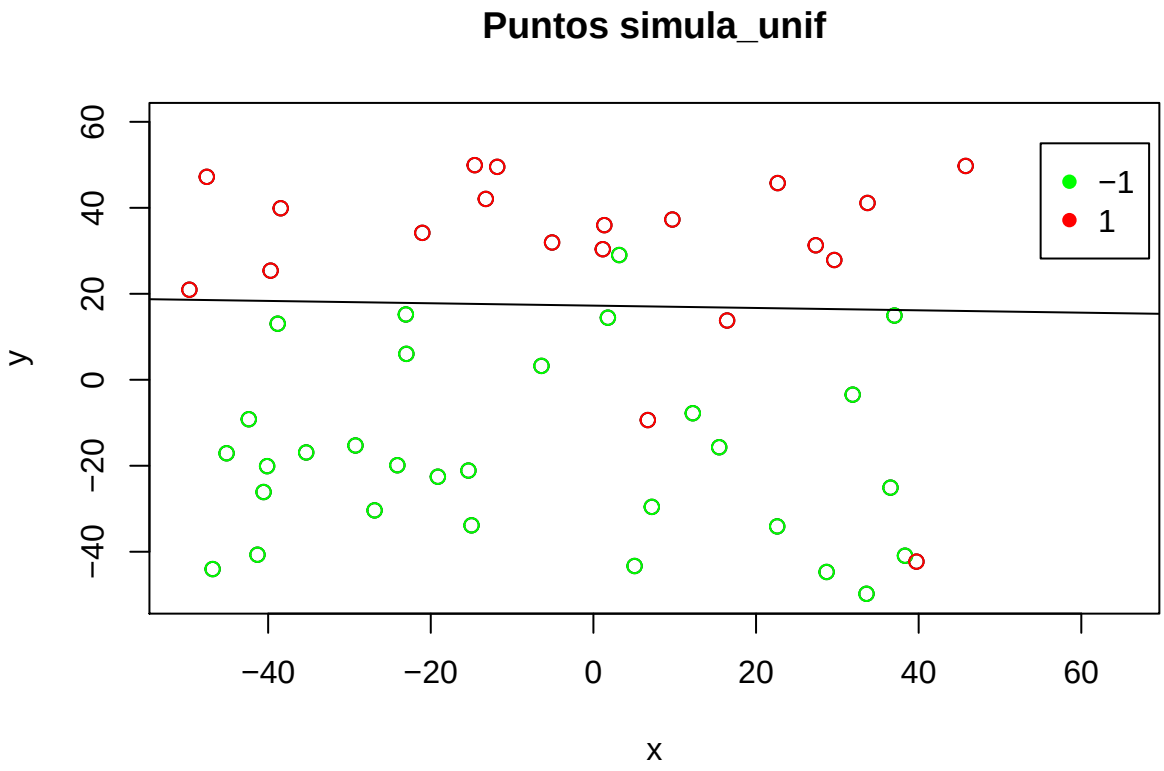
Para ello, vamos a crear primero una función que nos cambia un porcentaje determinado de etiquetas.

```
#funcion que nos cambia el % de las etiquetas
cambiarValores<- function(etiquetas, nvalores,posiciones){
  cambios <- sample(x = length(posiciones), size = nvalores, replace = FALSE)

  etiquetas[posiciones[cambios]] <- -etiquetas[posiciones[cambios]]
  etiquetas
}

#funcion que devuelve un vector con las posiciones donde se encuentran los elementos con valor "element
posiciones <- function(etiquetas,elemento){
  res <- matrix(0,nrow=1)
  elem <- 1

  for(i in 1:length(etiquetas)){
    if(etiquetas[i]==elemento){
      res[elem] = i
      elem <- elem+1
    }
  }
```

```
  res
}
```

Ahora pasamos a realizar la tarea.

```
#Cambiamos los valores de las etiquetas
pos_negativas <- posiciones(etiquetas_unif,-1)
pos_positivas <- posiciones(etiquetas_unif,1)

#Cambiamos el valor de las etiquetas
etiquetas_unif2 <- cambiarValores(etiquetas_unif,as.integer(0.1*length(pos_negativas)),pos_negativas)
etiquetas_unif2 <- cambiarValores(etiquetas_unif2,as.integer(0.1*length(pos_positivas)),pos_positivas)

#Volvemos a dibujar
plot(v_unif[,1],v_unif[,2],type="p",xlab="x",ylab="y",main="Puntos simula_unif",xlim=c(-50,65),ylim=c(-5
points(v_unif[,1][etiquetas_unif2==-1],v_unif[,2][etiquetas_unif2==-1],col="green")
points(v_unif[,1][etiquetas_unif2==1],v_unif[,2][etiquetas_unif2==1],col="red")
abline(recta_unif[2],recta_unif[1],col="black")
legend(x= 55, y = 55, legend = c(-1,1), col = c("green","red"),pch=16)
```



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

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

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

Formalizaremos ahora tanto las funciones que acabamos de indicar como la funcion de pintar la frontera.

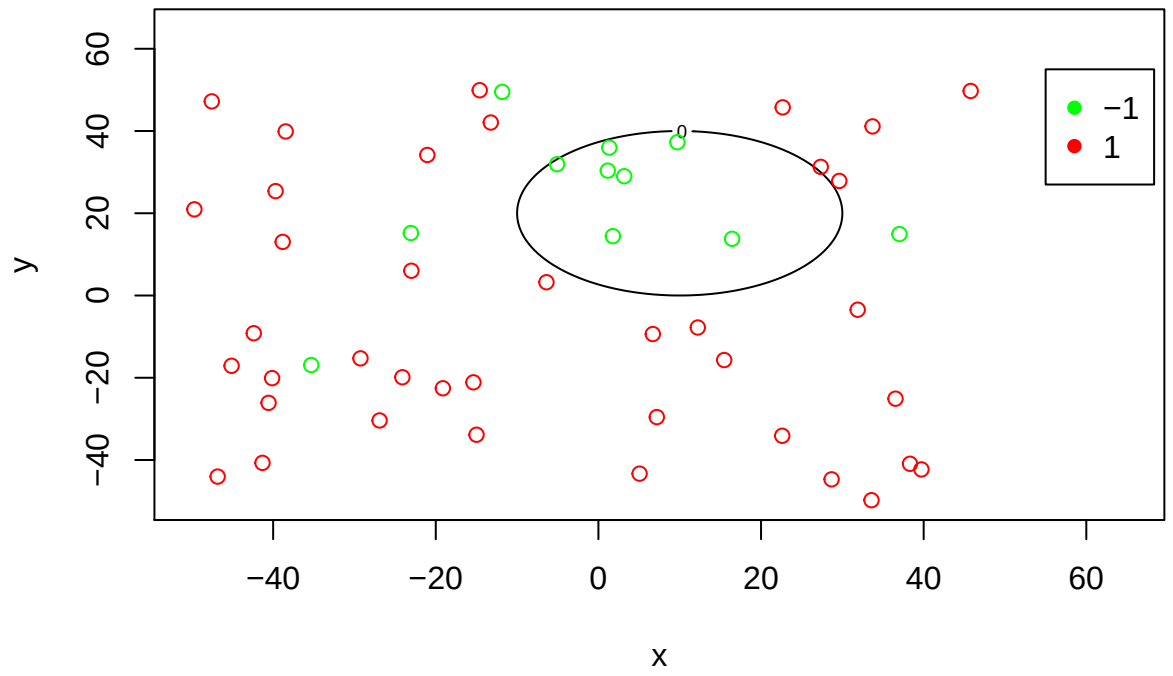
```
#Funcion que nos permite pintar la frontera de una funcion para un rango dado
pintar_frontera = function(f,puntos,etiquetas,rango=c(-50,50),legend) {
  x=y=seq(rango[1],rango[2],length.out = 500)
  z = outer(x,y,FUN=f)
  if (dev.cur()==1) # no esta abierto el dispositivo lo abre con plot
    plot(1, type="p", xlim=rango, ylim=rango)
  contour(x,y,z, levels = c(0), xlim =rango, ylim=rango, xlab = "x", ylab = "y")
  points(puntos[,1][etiquetas==-1],puntos[,2][etiquetas==-1],col="green")
  points(puntos[,1][etiquetas==1],puntos[,2][etiquetas==1],col="red")
  legend(x= legend[1], y = legend[2], legend = c(-1,1), col = c("green","red"),pch=16)
}

#Definimos todas las funciones
funcion3.1 <- function(x,y){
  (x-10)^2+(y-20)^2-400
}

funcion3.2 <- function(x,y){
  0.5*(x+10)^2+(y-20)^2-400
}

funcion3.3 <- function(x,y){
  0.5*(x-10)^2-(y+20)^2-400
}

funcion3.4 <- function(x,y){
  y-20*x^2-5*x+3
}

prueba3.1 <- function(x){
  sqrt(400-(x-10)^2)+20
}
```

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.

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

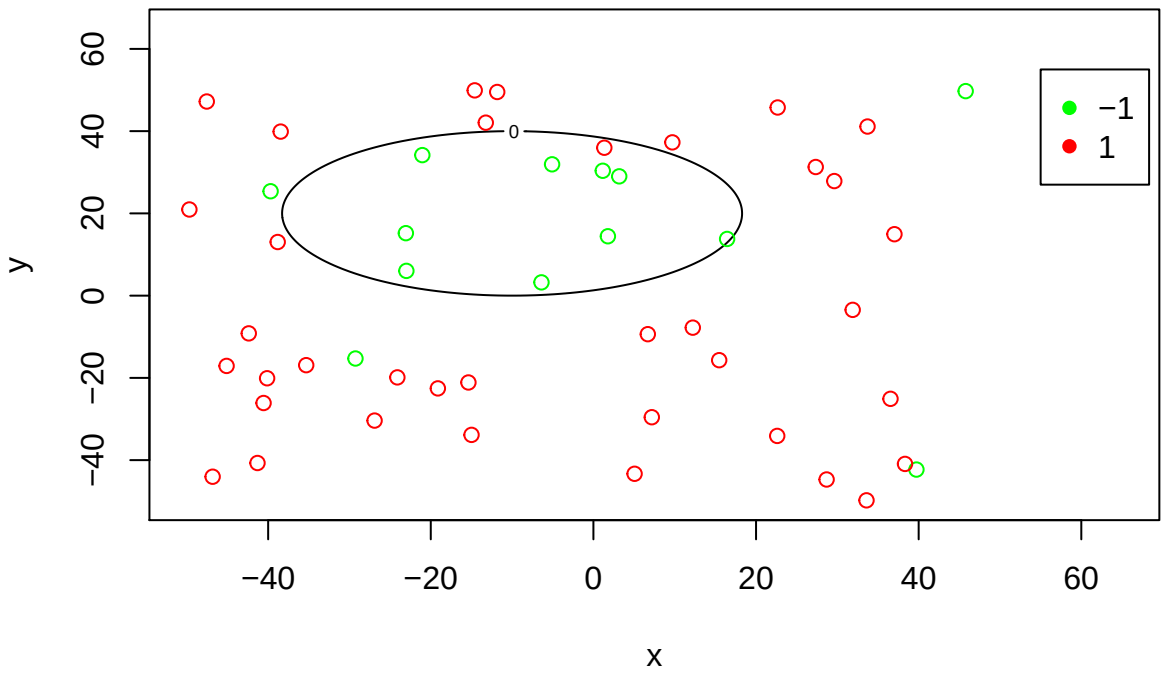
```
#Funcion 1

etiquetas_3.1 <- sign(funcion3.1(v_unif[,1],v_unif[,2]))

#Cambiamos los valores de las etiquetas
pos_negativas <- posiciones(etiquetas_3.1,-1)
pos_positivas <- posiciones(etiquetas_3.1,1)

#Cambiamos el valor de las etiquetas
etiquetas_3.1.2 <- cambiarValores(etiquetas_3.1,as.integer(0.1*length(pos_negativas)),pos_negativas)
etiquetas_3.1.2 <- cambiarValores(etiquetas_3.1.2,as.integer(0.1*length(pos_positivas)),pos_positivas)
```

```r
#Pintamos la frontera simula_unif
pintar_frontera(funcion3.1,v_unif,etiquetas_3.1.2,c(-50,65),c(55,55))
```



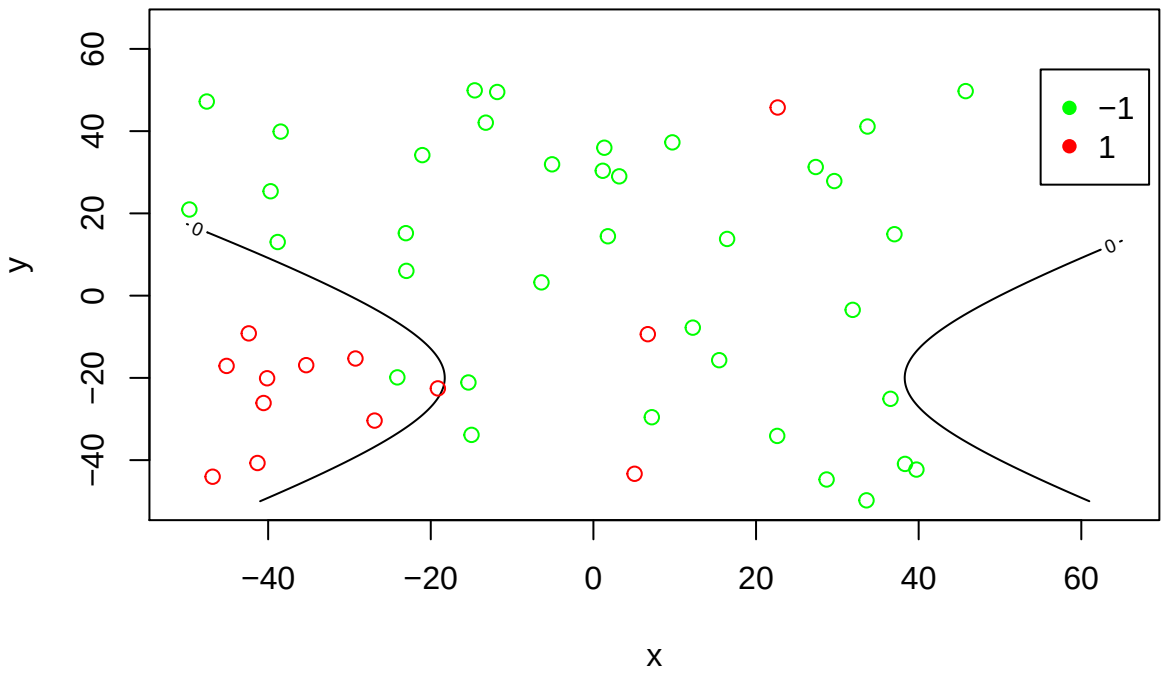
b) $f(x,y) = 0,5 \cdot (x+10)^2 + (y-20)^2 - 400$

```r
#Funcion 2

etiquetas_3.2 <- sign(funcion3.2(v_unif[,1],v_unif[,2]))

#Cambiamos los valores de las etiquetas
pos_negativas <- posiciones(etiquetas_3.2,-1)
pos_positivas <- posiciones(etiquetas_3.2,1)

#Cambiamos el valor de las etiquetas
etiquetas_3.2.2 <- cambiarValores(etiquetas_3.2,as.integer(0.1*length(pos_negativas)),pos_negativas)
etiquetas_3.2.2 <- cambiarValores(etiquetas_3.2.2,as.integer(0.1*length(pos_positivas)),pos_positivas)

#Pintamos la frontera simula_unif
pintar_frontera(funcion3.2,v_unif,etiquetas_3.2.2,c(-50,65),c(55,55))
```



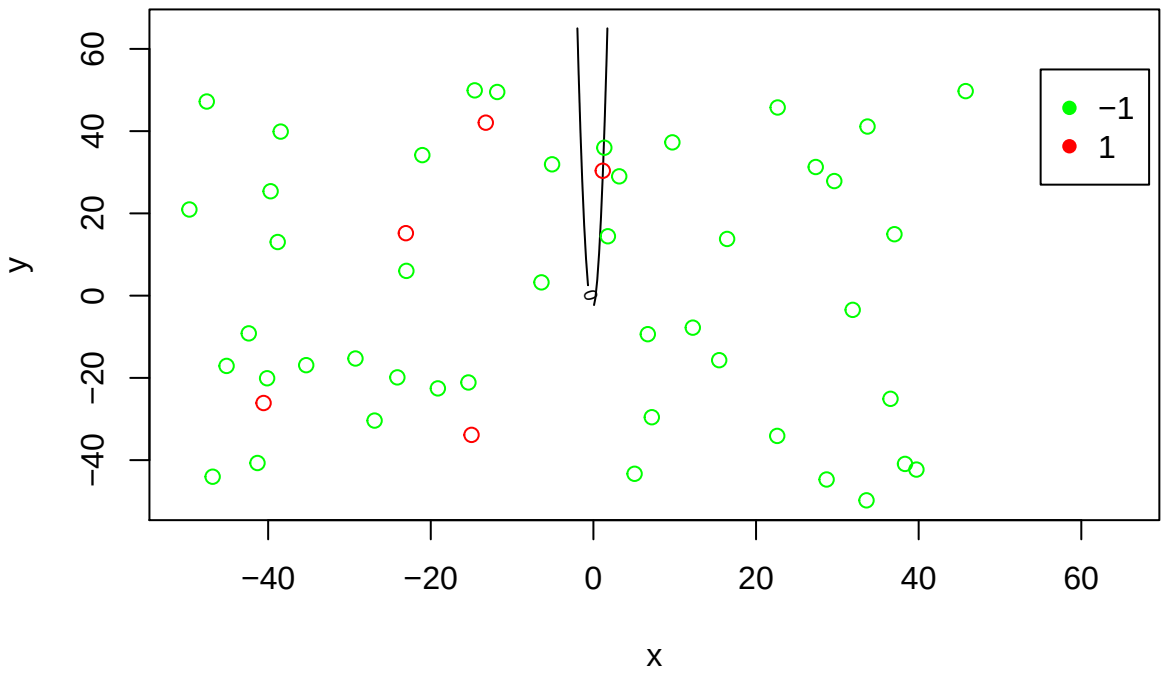
c) $0,5 \cdot (x-10)^2 - (y+20)^2 - 400$

```
#Funcion 3

etiquetas_3.3 <- sign(funcion3.3(v_unif[,1],v_unif[,2]))

#Cambiamos los valores de las etiquetas
pos_negativas <- posiciones(etiquetas_3.3,-1)
pos_positivas <- posiciones(etiquetas_3.3,1)

#Cambiamos el valor de las etiquetas
etiquetas_3.3.2 <- cambiarValores(etiquetas_3.3,as.integer(0.1*length(pos_negativas)),pos_negativas)
etiquetas_3.3.2 <- cambiarValores(etiquetas_3.3.2,as.integer(0.1*length(pos_positivas)),pos_positivas)

#Pintamos la frontera simula_unif
pintar_frontera(funcion3.3,v_unif,etiquetas_3.3.2,c(-50,65),c(55,55))
```



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

```
#Funcion 4

etiquetas_3.4 <- sign(funcion3.4(v_unif[,1],v_unif[,2]))

#Cambiamos los valores de las etiquetas
pos_negativas <- posiciones(etiquetas_3.4,-1)
pos_positivas <- posiciones(etiquetas_3.4,1)

#Cambiamos el valor de las etiquetas
etiquetas_3.4.2 <- cambiarValores(etiquetas_3.4,as.integer(0.1*length(pos_negativas)),pos_negativas)
etiquetas_3.4.2 <- cambiarValores(etiquetas_3.4.2,as.integer(0.1*length(pos_positivas)),pos_positivas)

#Pintamos la frontera simula_unif
pintar_frontera(funcion3.4,v_unif,etiquetas_3.4.2,c(-50,65),c(55,55))
```



Podemos ver en las gráficas como se han modificado los puntos respecto a cada función y aparecen algunos elementos cambiados de clase. Respecto a la calidad de clasificación con la regresion lineal, siempre dependerá de como se reparte la muestra y los datos que trabajamos pero se ve claramente que las funciones más complejas permiten expresar clasificaciones más complejas que una recta seria imposible representar ya que con una recta, si tenemos puntos de una clase por medio de puntos de otra clase no podría dividirlos perfectamente mientras que por ejemplo una elipse podría hacerlo sin problema como es el caso del apartado d), donde se puede ver perfectamente la diferencia, ya que una recta sería imposible utilizarla para reflejar la diferencia de clases. Esto no es siempre así, no siempre una función compleja va a representar mejor los datos que una lineal, como he comentado, dependerá de la propia muestra, pero si es cierto que estas funciones permiten representar una clasificación mas compleja que una recta.

Modelos lineales

1. (3 puntos) 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.

Antes de nada crearemos la función ajustar_PLA.

```
#Funcion del Perceptron
ajustar_PLA <- function(datos,label,max_iter,vini){
  wnew <- vini
```

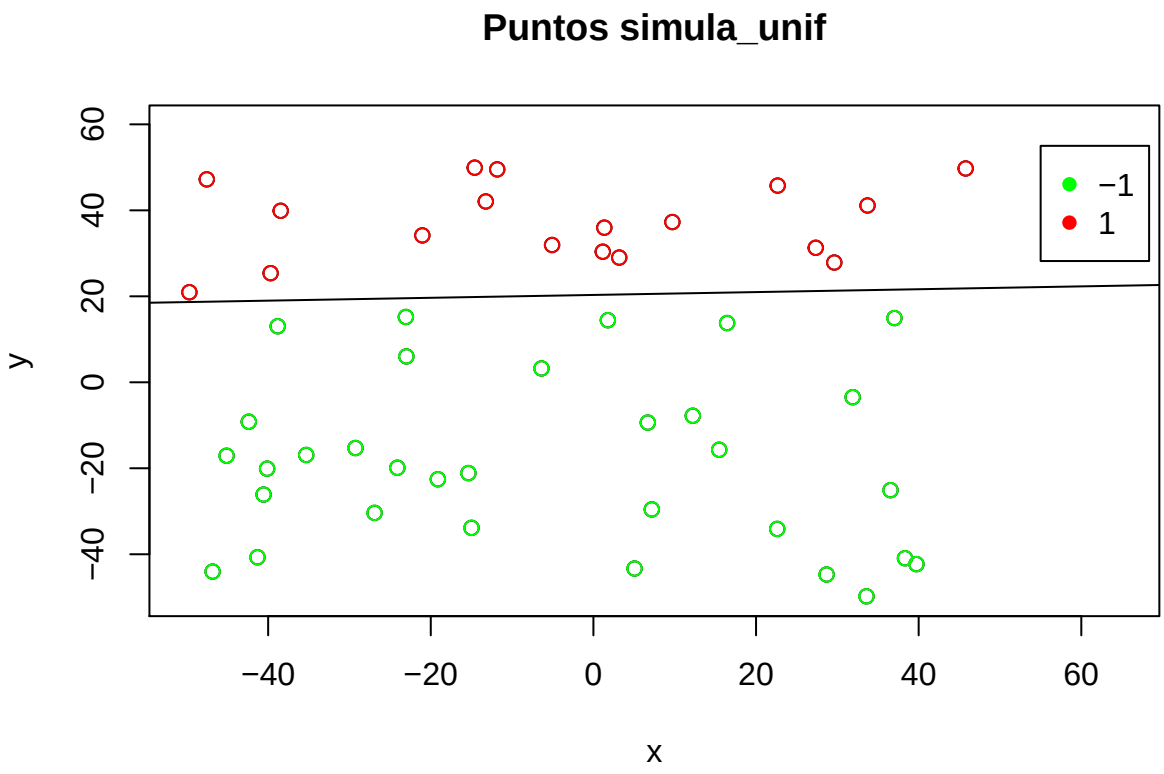
```
  datos <- cbind(datos,1.0)

  for(j in 1:max_iter){
    wold <- wnew
    for(i in 1:nrow(datos)){
      if(sign(datos[i,]%*%wnew) != label[i]){
        wnew <- wnew+label[i]*datos[i,]
      }
    }
    if(norm(wold-wnew,type = "2") < 10e-12){
      return(c(wnew,j))
    }
  }
  c(wnew,j)
}
```

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.

```
#Inicializamos los pesos iniciales
wini = matrix(0.0,nrow = 3)
#Calculamos los pesos
w <- ajustar_PLA(v_unif,etiquetas_unif,10e4,wini)
w
```

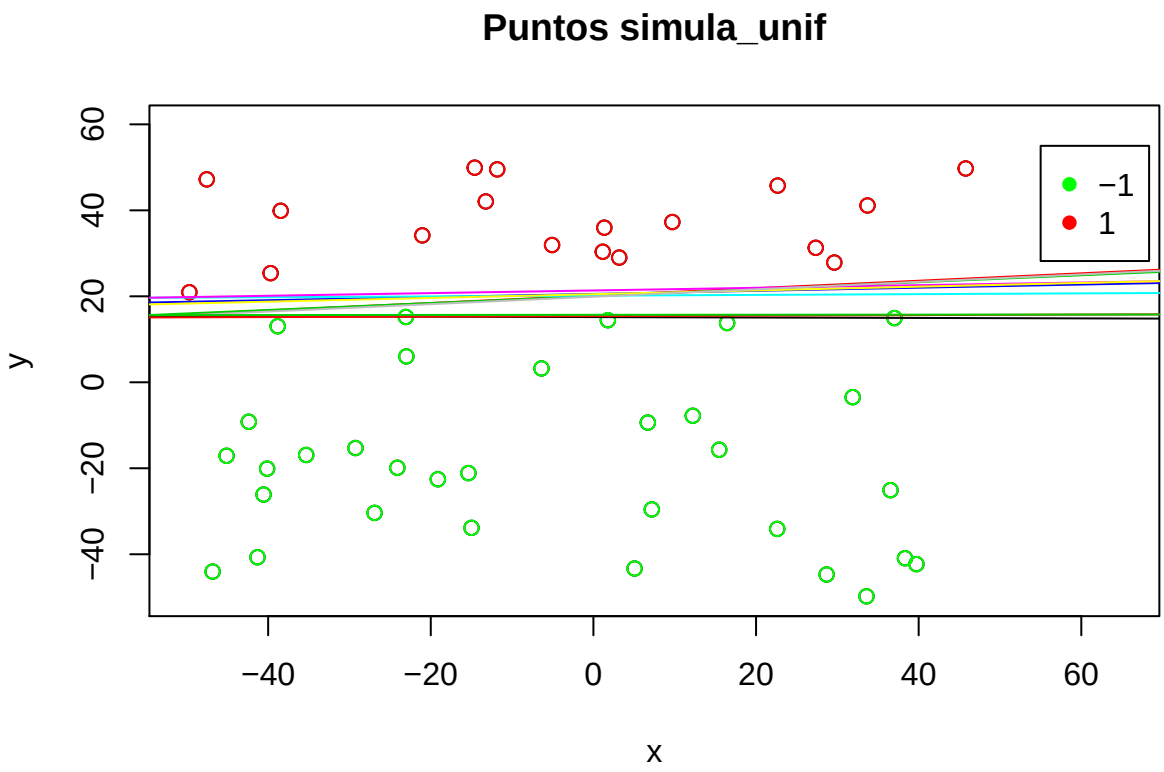
```
## [1]   -1.139467   34.306122 -697.000000  240.000000
```

```
#Almaceno el valor de la iteracion para mostrarlo mas tarde en una tabla
resultados <- matrix(c(wini,w),nrow=1)

#Dibujamos los puntos con la recta
plot(v_unif[,1],v_unif[,2],type="p",xlab="x",ylab="y",main="Puntos simula_unif",xlim=c(-50,65),ylim=c(-5
points(v_unif[,1][etiquetas_unif==-1],v_unif[,2][etiquetas_unif==-1],col="green")
points(v_unif[,1][etiquetas_unif==1],v_unif[,2][etiquetas_unif==1],col="red")
abline(-(w[3]/w[2]),-(w[1]/w[2]),col="black")
legend(x= 55, y = 55, legend = c(-1,1), col = c("green","red"),pch=16)
```



Ahora pasamos ha realizarlo con un vector de inicio aleatorio.

```
plot(v_unif[,1],v_unif[,2],type="p",xlab="x",ylab="y",main="Puntos simula_unif",xlim=c(-50,65),ylim=c(-5
points(v_unif[,1][etiquetas_unif==-1],v_unif[,2][etiquetas_unif==-1],col="green")
points(v_unif[,1][etiquetas_unif==1],v_unif[,2][etiquetas_unif==1],col="red")
legend(x= 55, y = 55, legend = c(-1,1), col = c("green","red"),pch=16)
c <- 1

for(i in 1:10){
  #Creamos los pesos iniciales aleatoriamente
  wini <- matrix(c(runif(3,0,1)),nrow=3)

  #Calculamos los valores de la recta
  w <- ajustar_PLA(v_unif,etiquetas_unif,10e4,wini)

  resultados <- rbind(resultados,c(wini,w))

  #Dibujamos
  c <- c+1
  abline(-(w[3]/w[2]),-(w[1]/w[2]),col=c)
}
```



Vamos a mostrar la tabla con los resultados y a comentarlos.

	x_ini	y_ini	v_indep_ini	x_fin	y_fin	v_indep_fin	Iteraciones
ini0-unif	0.0000000	0.0000000	0.0000000	-1.1394674	34.30612	-697.0000	240
ale1-unif	0.8679463	0.2015974	0.6051639	-3.1032911	35.96589	-725.3948	247
ale2-unif	0.6466307	0.6932997	0.6425489	-2.8849854	35.72680	-716.3575	247
ale3-unif	0.6978288	0.7117889	0.7625658	-1.2324924	33.78270	-694.2374	235
ale4-unif	0.1933802	0.7992337	0.0824301	-0.2948310	33.29667	-670.9176	221
ale5-unif	0.8319392	0.1692830	0.7821419	-1.0036881	32.55155	-695.2179	229
ale6-unif	0.5382605	0.3365344	0.0292436	-1.4910315	34.24523	-701.9708	236
ale7-unif	0.8138010	0.8611121	0.3062543	-3.1084033	35.95172	-715.6937	248
ale8-unif	0.2510711	0.6543522	0.0877284	0.2473705	49.83717	-755.9123	274
ale9-unif	0.8058788	0.7528111	0.6585271	-0.1471841	25.60909	-394.3415	93
ale10-unif	0.0597817	0.9745928	0.3014817	-0.0626637	54.16550	-849.6985	329

Tenemos como valores de media **236 iteraciones**. No obstante, vemos que aunque la mayoría de las iteraciones son muy parecidas, varían y en algunos casos tienen una diferencia notable. Lo que nos indica que efectivamente según del punto inicial tendremos que realizar más o menos iteraciones.

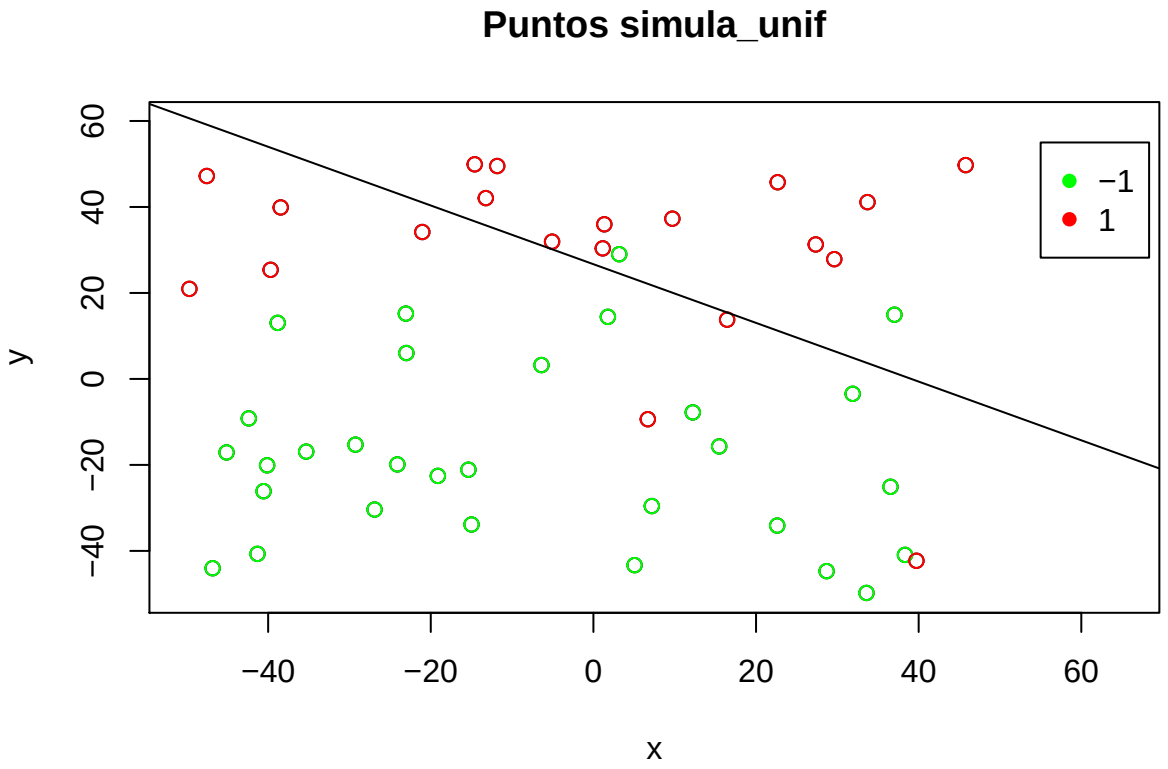
b) Hacer lo mismo que antes usando ahora los datos del apartado 2b de la sección.1. ¿Observa algún comportamiento diferente? En caso afirmativo diga cual y las razones para que ello ocurra.

```
#Inicializamos los pesos iniciales
wini = matrix(0.0,nrow = 3)
#Calculamos los pesos
w <- ajustar_PLA(v_unif,etiquetas_unif2,10e3,wini)
```

```
w
```

```
## [1]     8.953473    13.114202  -350.000000 10000.000000
```

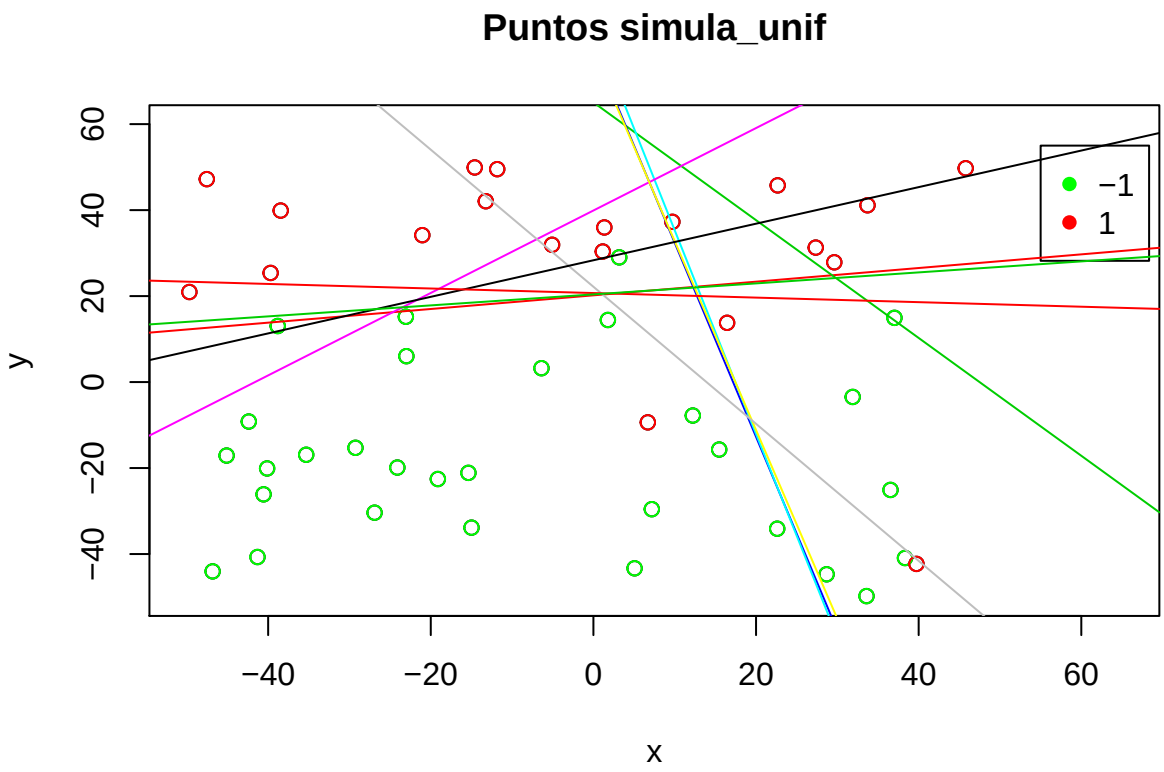
```
resultados <- matrix(c(wini,w),nrow=1)

#Dibujamos los puntos con la recta
plot(v_unif[,1],v_unif[,2],type="p",xlab="x",ylab="y",main="Puntos simula_unif",xlim=c(-50,65),ylim=c(-5
points(v_unif[,1][etiquetas_unif2==-1],v_unif[,2][etiquetas_unif2==-1],col="green")
points(v_unif[,1][etiquetas_unif2==1],v_unif[,2][etiquetas_unif2==1],col="red")
abline(-(w[3]/w[2]),-(w[1]/w[2]),col="black")
legend(x= 55, y = 55, legend = c(-1,1), col = c("green","red"),pch=16)
```



Ahora con los pesos iniciales aleatorios

```
#Dibujamos
plot(v_unif[,1],v_unif[,2],type="p",xlab="x",ylab="y",main="Puntos simula_unif",xlim=c(-50,65),ylim=c(-5
points(v_unif[,1][etiquetas_unif2==-1],v_unif[,2][etiquetas_unif2==-1],col="green")
points(v_unif[,1][etiquetas_unif2==1],v_unif[,2][etiquetas_unif2==1],col="red")
legend(x= 55, y = 55, legend = c(-1,1), col = c("green","red"),pch=16)
c <- 1

for(i in 1:10){
  #Creamos los pesos iniciales aleatoriamente
  wini <- matrix(c(runif(3,0,1)),nrow=3)

  #Calculamos los valores de la recta
  w <- ajustar_PLA(v_unif,etiquetas_unif2,10e3,wini)
```

```
  resultados <- rbind(resultados,c(wini,w))

  c <- c+1
  abline(-(w[3]/w[2]),-(w[1]/w[2]),col=c)
}
```



Vamos a mostrar la tabla con los resultados y a comentarlos.

	x_ini	y_ini	v_indep_ini	x_fin	y_fin	v_indep_fin	Iteraciones
ini0-unif	0.0000000	0.0000000	0.0000000	8.9534731	13.114202	-350.0000	10000
ale1-unif	0.9353572	0.1958483	0.4097832	-2.7845494	17.527098	-353.5902	10000
ale2-unif	0.3984936	0.3471488	0.8520748	7.4788528	5.458435	-355.1479	10000
ale3-unif	0.0035511	0.1608926	0.6255179	20.2779732	4.494044	-348.3745	10000
ale4-unif	0.5822893	0.6814801	0.5079881	20.3695051	4.301541	-355.4920	10000
ale5-unif	0.6452597	0.0289176	0.2093367	-8.3328272	8.689781	-346.7907	10000
ale6-unif	0.9985143	0.0572269	0.6793312	20.5794505	4.687062	-359.3207	10000
ale7-unif	0.6794991	0.7390569	0.4209916	24.7765678	15.540655	-344.5790	10000
ale8-unif	0.5059779	0.0582139	0.4904851	-5.3209025	12.510795	-354.5095	10000
ale9-unif	0.3931270	0.7804698	0.9208513	0.9032747	17.052781	-353.0791	10000
ale10-unif	0.0724992	0.4147924	0.3131684	-2.1798421	17.046581	-347.6868	10000

Vemos claramente diferencias respecto al anterior, tanto en el trazo de la recta en el plano como en la propia tabla. Al tener una muestra "errónea" en la que no podemos separar completamente una clase de otra, el PLA ha agotado todas las iteraciones que he puesto como máximo de búsqueda y aún así encuentra rectas un

poco defectuosas. Respecto a la relación entre punto de inicio e iteraciones, en este caso al hacer en todos los caso el máximo no podemos deducir nada.

2. (4 puntos) 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 $X = [0, 2] \times [0, 2]$ con probabilidad uniforme de elegir cada $x \in X$. 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.

a) Implementar Regresión Logística(RL) con Gradiente Descendente Estocástico (SGD) bajo las siguientes condiciones:
- Inicializar el vector de pesos con valores 0.
- 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.
- 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 $\eta = 0,01$

Lo primero que necesitamos es definir la sigmoide, que la definiremos como $\sigma = \frac{1}{1+e^{-x}}$.

Tambien definiremos la función $E_{in}(w) = \frac{1}{N} \cdot \sum_{i=1}^{N} ln(1 + e^{-y_i \cdot w^T \cdot x_i})$ y $\nabla E_{in}(w) = \frac{1}{N} \cdot \sum_{i=1}^{N} -y_i \cdot x_i \cdot \sigma(-y_i \cdot w^T \cdot x_i)$ de la regresión logística que utilizaremos con el GDE.

```
sigmoide <- function(x){
  as.double(1/(1+exp(-x)))
}

#Calcula el error de la muestra en regresion logistica
E_in.logistica <- function(datos,label,w){
  suma <- 0.0
  datos <- cbind(datos,1)

  for(i in 1:nrow(datos)){
    suma <- suma + log(1+exp(-datos[i,]%*%w*label[i]))
  }
  suma/nrow(datos)
}

#Calcula el valor de un único elemento del gradiente de la regresion logística. (para usarlo en el GDE)
derv.E_in.logistica <- function(datos,label,w,i){
  (-label[i]*datos[i,]*sigmoide(-datos[i,]%*%w*label[i]))
}
```

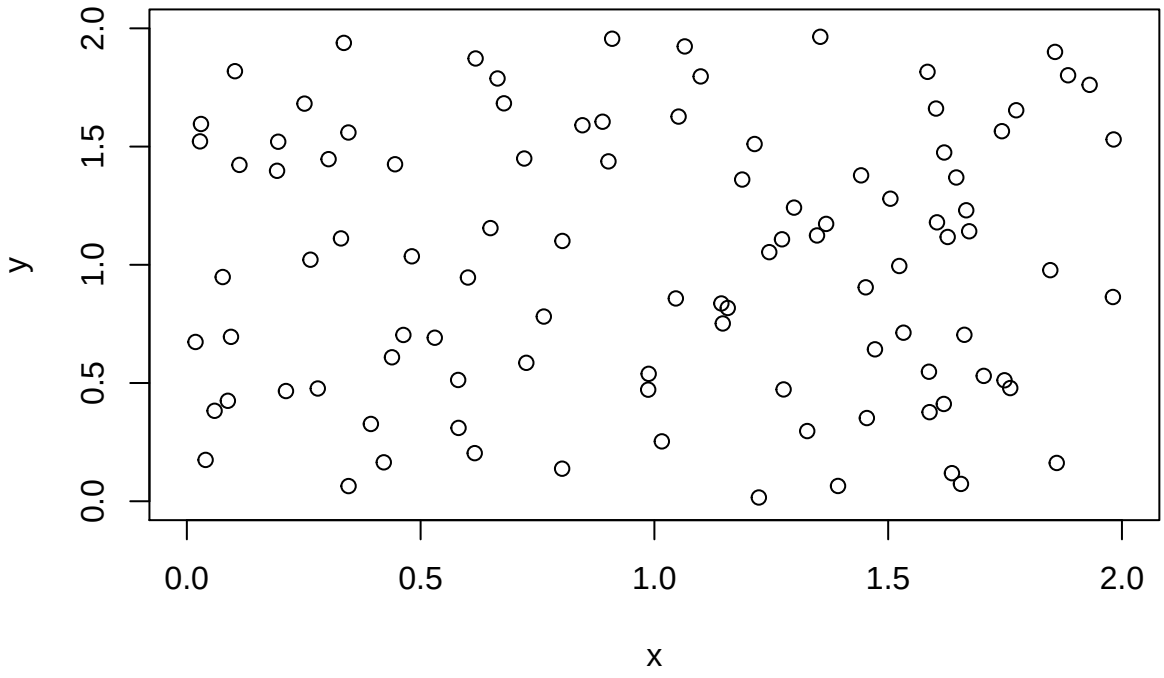
Ahora definimos el gradiente descendente estocástico para utilizarlo con regresión logística y cumpliendo las restricciones que nos piden.

```
GDE <- function(funcion, derv, datos, label, niter, paso, umbral,tam,wini){
  wnew <- wini
  i <- 1
  dif <- umbral + 1
  datos <- cbind(datos,1)
  iteraciones <- as.integer(nrow(datos)/tam)
```

```
  while(umbral < dif && i < niter){
    datos <- cbind(datos,as.matrix(label,ncol=1))
    datos <- datos[sample.int(nrow(datos)),]
    label <- c(datos[,4])
    datos <- datos[,-4]
    wold <- wnew
    k <- 1
    for(j in 1:iteraciones){
      grad <- 0
      lim <- k+tam
      while(k < lim){
        grad <- grad + derv(datos,label,wnew,k)
        k <- k+1
      }
      grad <- grad/tam
      wnew <- wnew - paso*grad
      i <- i+1
    }
    dif <- norm(wold-wnew,type ="2")
  }
c(wnew,i)
}
```

Pasamos ahora a la propia ejecucion del algoritmo.

```
#Escogemos el rango que nos dice el ejercicio
rango <- c(0,2)

#Creamos la muestra de los puntos
muestra <- simula_unif(100,2,rango)
plot(muestra[,1],muestra[,2],type="p",xlab="x",ylab="y",main="Puntos muestra",xlim=rango,ylim=rango)
```



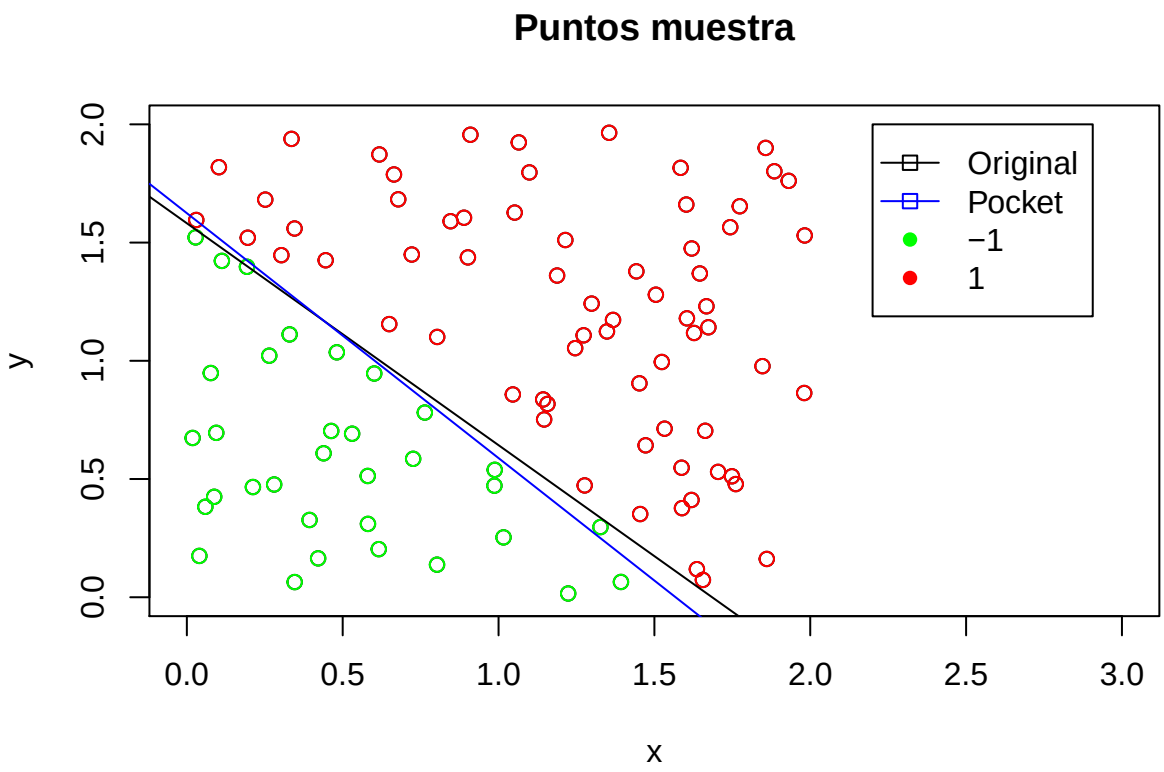

```
#Creamos la recta
recta_muestra <- simula_recta(rango)
recta_muestra
```

```
## [1] -0.9385862  1.5816946
```

```
#Clasificamos las etiquetas respecto a la recta usando la funcion que utilizamos en un ejercicio anteri
etiquetas_muestra <- funcion2(muestra[,1],muestra[,2],recta_muestra)

#Calculamos los pesos utilizando el GDE y regresión logística
wini <- matrix(0.0,nrow=3)
w <- GDE(E_in.logistica,derv.E_in.logistica,muestra,etiquetas_muestra,10e12,0.01,0.01,1,wini)
w
```

```
## [1]     5.251944     5.069944    -8.235462 46301.000000
```

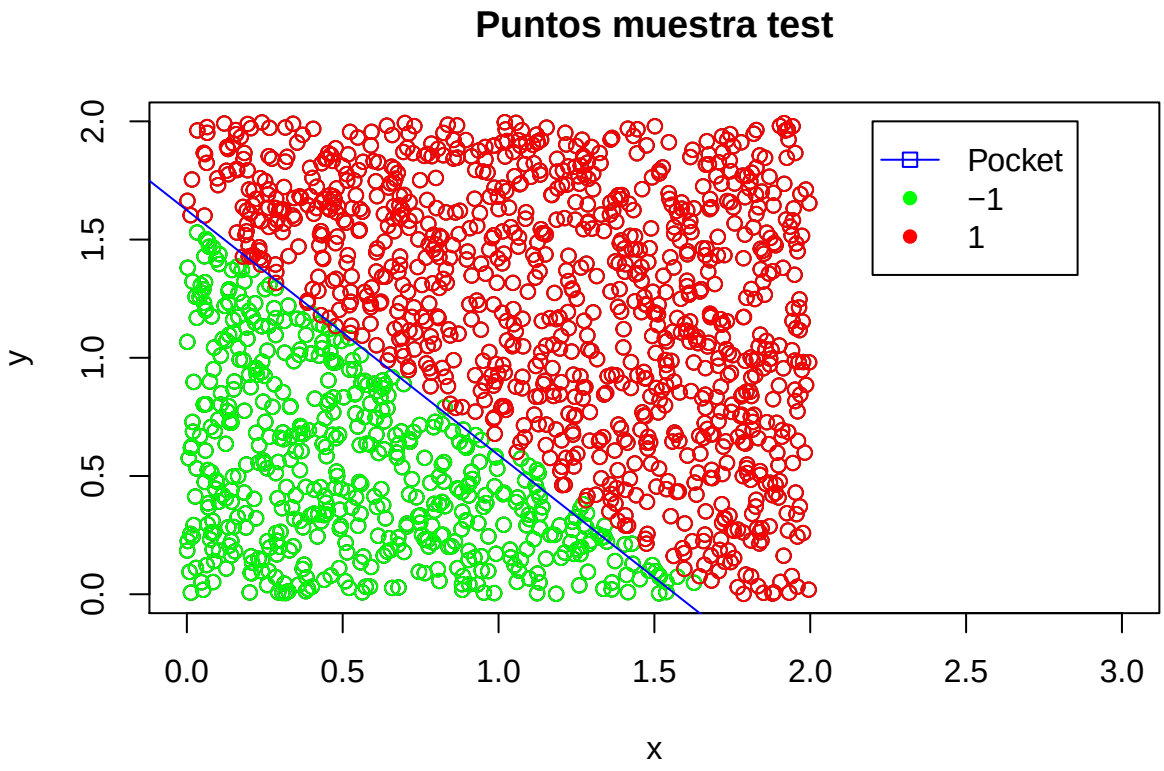
```
#Dibujamos los resultados
plot(muestra[,1],muestra[,2],type="p",xlab="x",ylab="y",main="Puntos muestra",xlim=c(0,3),ylim=rango)
points(muestra[,1][etiquetas_muestra==-1],muestra[,2][etiquetas_muestra==-1],col="green")
points(muestra[,1][etiquetas_muestra==1],muestra[,2][etiquetas_muestra==1],col="red")
abline(recta_muestra[2],recta_muestra[1],col="black")
abline(-(w[3]/w[2]),-(w[1]/w[2]),col="blue")
legend(x= 2.2, y = 2, legend = c("Original","Pocket",-1,1), col = c("black","blue","green","red"),lty=c
```



Podemos ver como el GDE se acerca al resultado esperado aunque con esta tasa de aprendizaje difícilmente conseguiremos un resultado exacto o muy bueno, dependerá de la muestra.

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).

```
#Creamos la muestra de 1500 puntos
muestra.test <- simula_unif(1500,2,rango)

#Etiquetamos los puntos
etiquetas_muestra.test <- funcion2(muestra.test[,1],muestra.test[,2],recta_muestra)

#Reestructuramos los pesos ya que en el apartado anterior también contenían las iteraciones.
w <- matrix(c(w[1],w[2],w[3]),nrow = 3)

#Dibujamos los puntos de la muestra test
plot(muestra.test[,1],muestra.test[,2],type="p",xlab="x",ylab="y",main="Puntos muestra test",xlim=c(0,3
points(muestra.test[,1][etiquetas_muestra.test==-1],muestra.test[,2][etiquetas_muestra.test==-1],col="g
points(muestra.test[,1][etiquetas_muestra.test==1],muestra.test[,2][etiquetas_muestra.test==1],col="red
abline(-(w[3]/w[2]),-(w[1]/w[2]),col="blue")
legend(x= 2.2, y = 2, legend = c("Pocket",-1,1), col = c("blue","green","red"),lty=c(1,0,0),pch = c(0,1
```




```
print(paste("E_in en la muestra para RL con GDE: ",E_in.logistica(muestra,etiquetas_muestra,w)))
```

```
## [1] "E_in en la muestra para RL con GDE:  0.128152351810062"
```

```
print(paste("E_in en el test para RL con GDE: ",E_in.logistica(muestra.test,etiquetas_muestra.test,w)))
```

```
## [1] "E_in en el test para RL con GDE:  0.132255964383773"
```

Vemos claramente que el error fuera de la muestra es algo más elevado al error dentro de la muestra, aunque no hay mucha diferencia ya que entonces hubiesemos realizado un sobreajuste.

Bonus

1. (2 puntos) Clasificación de Dígitos. Considerar el conjunto de datos de los dígitos manuscritos y seleccionar las muestras de los dígitos 4 y 8. Usar los ficheros de entrenamiento (training) y test que se proporcionan. Extraer las características de intensidad promedio y simetría en la manera que se indicó en el ejercicio 3 del trabajo 1.

1. Plantear un problema de clasificación binaria que considere el conjunto de entrenamiento como datos de entrada para aprender la función g.

```
#Funcion para calcular la simetria
fsimetria <- function(A){
  A = abs(A-A[,ncol(A):1])
  -sum(A)
}
```

```
setwd("./datos")

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

digitos48.train = digit.train[digit.train$V1==4 | digit.train$V1==8,]
digitos.train = digitos48.train[,1]    # vector de etiquetas del train
ndigitos.train = nrow(digitos48.train)  # numero de muestras del train

# se retira la clase y se monta una matriz 3D: 599*16*16
grises.train = array(unlist(subset(digitos48.train,select=-V1)),c(ndigitos.train,16,16))
rm(digit.train)
rm(digitos48.train)

intensidad.train <- apply(grises.train, 1, mean)

simetria.train <- apply(grises.train, 1, fsimetria)

rm(grises.train)
digitos.train <- replace(digitos.train, digitos.train == 4, 1)
digitos.train <- replace(digitos.train, digitos.train == 8, -1)
datosTr = as.matrix(cbind(intensidad.train,simetria.train))

##############################################################################################################

#TEST

digit.test <- read.table("zip.test",quote="\"", comment.char="", stringsAsFactors=FALSE)

digitos48.test = digit.test[digit.test$V1==4 | digit.test$V1==8,]
digitos.test = digitos48.test[,1]    # vector de etiquetas del test
ndigitos.test = nrow(digitos48.test)  # numero de muestras del test

# se retira la clase y se monta una matriz 3D: 599*16*16
grises.test = array(unlist(subset(digitos48.test,select=-V1)),c(ndigitos.test,16,16))
grises.test <- as.numeric(grises.test)
grises.test <- array(grises.test,(c(ndigitos.test,16,16)))
rm(digit.test)
rm(digitos48.test)

intensidad.test <- apply(grises.test, 1, mean)

simetria.test <- apply(grises.test, 1, fsimetria)
rm(grises.test)
digitos.test <- replace(digitos.test, digitos.test == 4, 1)
digitos.test <- replace(digitos.test, digitos.test == 8, -1)
datosTst = as.matrix(cbind(intensidad.test,simetria.test))
```

2. Usar un modelo de Regresión Lineal y aplicar PLA-Pocket como mejora. Responder a las siguientes cuestiones.

Para ello definiremos las funciones $E_{in}(w) = \frac{1}{N} \cdot \sum_{i=1}^{N} sign(w^T \cdot x_i \neq y_i)$ y $E_{out}(w) \leq E_{in} + \sqrt{\frac{1}{2 \cdot N} \cdot log(\frac{2}{\delta})}$.

También rescataremos la función de regresion lineal utilizando la pseudoinversa de la practica anterior y

crearemos PLA-Pocket.

```
E_in.PLA <- function(datos,label,w){
  suma <- 0.0
  datos <- cbind(datos,1)

  for(i in 1:nrow(datos)){
    if(sign(datos[i,]%*%w) != label[i]){
      suma <- suma+1
    }
  }
  suma/nrow(datos)
}

E_out <- function(delta,N,E_in){
  E_in+sqrt(1/(2*N)*log(2/delta))
}

Regress_Lin <- function(datos, label){
 # Añadimos al final la columna de 1 ya que nuestra variable independiente es w3
  datos <- cbind(datos,1)
  # multiplicamos la matriz traspuesta de datos por la matriz de datos
  x <- t(datos)%*%datos
  # Calculamos la inversa de esta matriz
  x.inv <- solve(x)
  # Multiplicamos la inversa calculada por la traspuesta de los datos = pseudoinversa = (X^T*X)^-1*X^T
  x.pseudo.inv <- x.inv %*% t(datos)
  # Calculamos los pesos
  w <- x.pseudo.inv %*% label
  w
}

PLA_Pocket <- function(datos,label,max_iter,wini){
  w <- wini
  best_w <- w
  best_error <- E_in.PLA(datos,label,w)

  for(i in 1:max_iter){
    w <- ajustar_PLA(datos,label,1,w)
    w <- matrix(c(w[1],w[2],w[3]),ncol=1)
    Err <- E_in.PLA(datos,label,w)
    if(Err < best_error){
      best_w <- w
      best_error <- Err
    }
  }
  best_w
}
```

a) Generar gráficos separados (en color) de los datos de entrenamiento y test junto con la función estimada.

```
#Datos de entrenamiento

wini <- matrix(0,nrow=3)
wr <- Regress_Lin(datosTr,digitos.train)
```

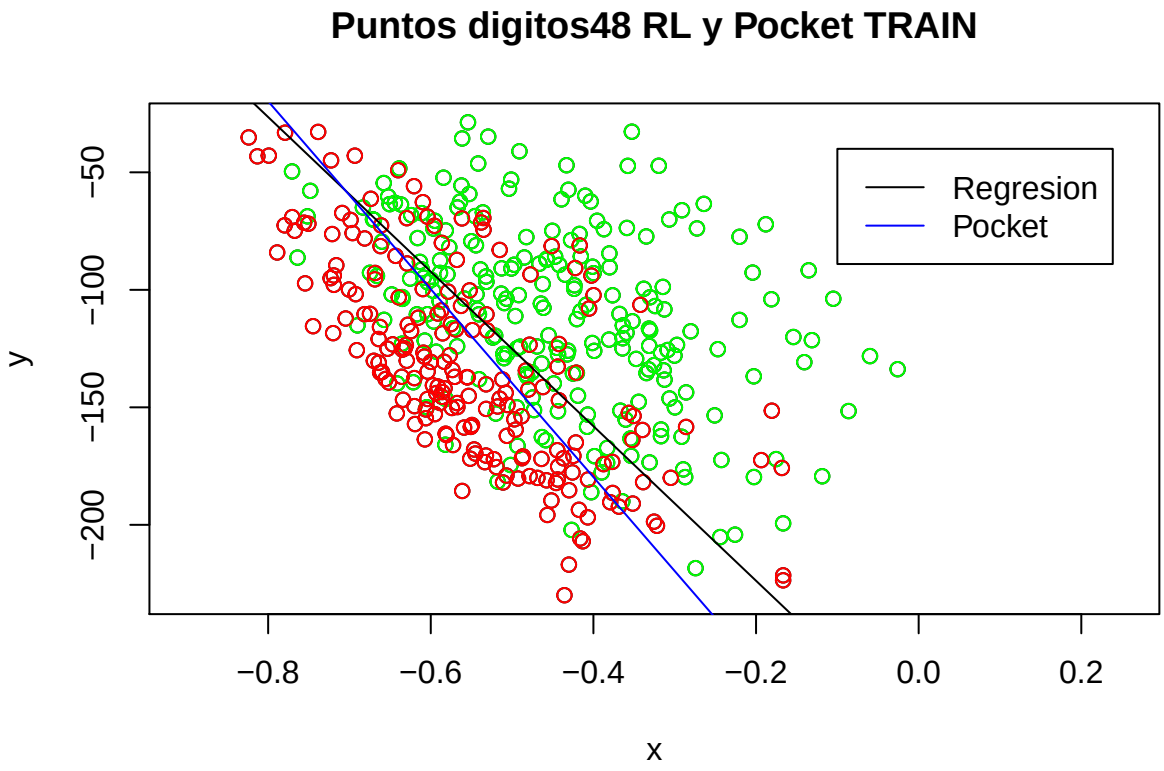
```
wr
```

```
##                          [,1]
## intensidad.train -3.60417555
## simetria.train   -0.01095946
##                  -3.17359335
```

```
plot(datosTr[,1],datosTr[,2],type="p",xlab="x",ylab="y",main="Puntos digitos48 RL y Pocket TRAIN",xlim =
points(datosTr[,1][digitos.train==-1],datosTr[,2][digitos.train==-1],col="green")
points(datosTr[,1][digitos.train==1],datosTr[,2][digitos.train==1],col="red")
abline(-(wr[3]/wr[2]),-(wr[1]/wr[2]),col="black")

wp <- PLA_Pocket(datosTr,digitos.train,1e4,wr)
wp
```

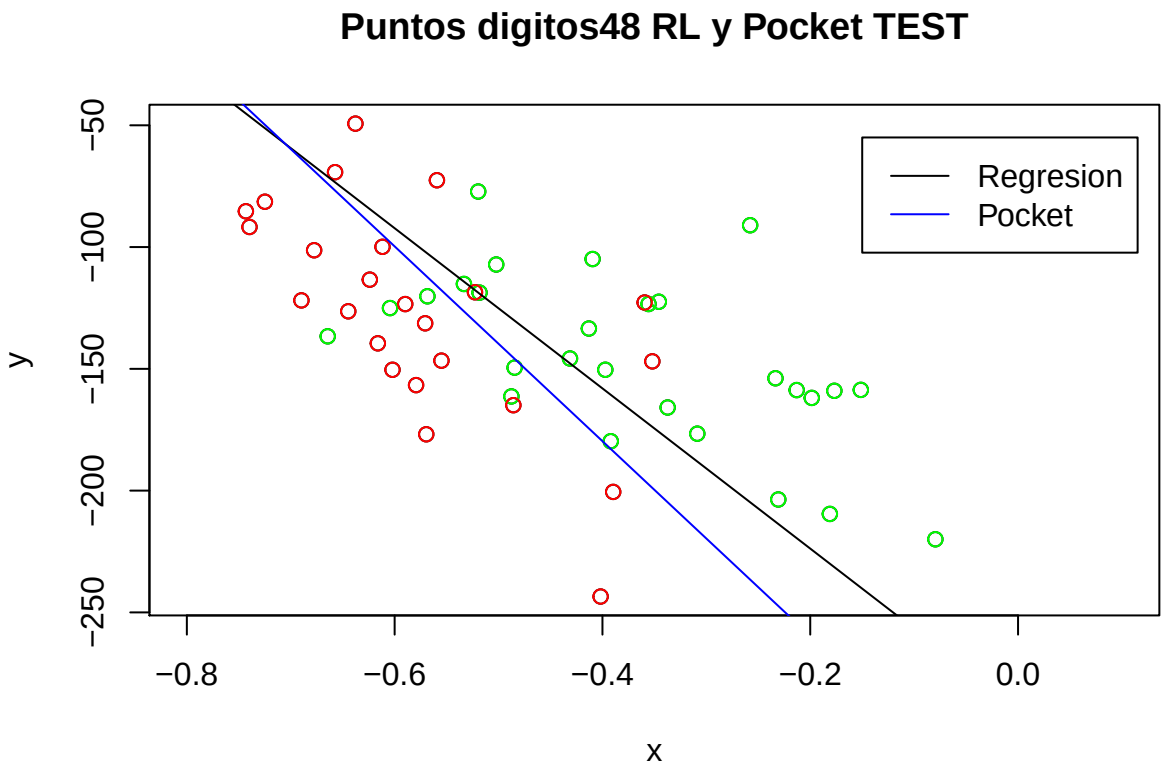
```
##              [,1]
## [1,] -24956.35155
## [2,]    -62.41696
## [3,] -21198.17359
```

```
abline(-(wp[3]/wp[2]),-(wp[1]/wp[2]),col="blue")
legend(x= -0.1, y = -40, legend = c("Regresion","Pocket"), col = c("black","blue"),lty=1)
```




```
#Datos de test

plot(datosTst[,1],datosTst[,2],type="p",xlab="x",ylab="y",main="Puntos digitos48 RL y Pocket TEST",xlim
points(datosTst[,1][digitos.test==-1],datosTst[,2][digitos.test==-1],col="green")
points(datosTst[,1][digitos.test==1],datosTst[,2][digitos.test==1],col="red")
```

```
abline(-(wr[3]/wr[2]),-(wr[1]/wr[2]),col="black")
abline(-(wp[3]/wp[2]),-(wp[1]/wp[2]),col="blue")
legend(x= -0.15, y = -55, legend = c("Regresion","Pocket"), col = c("black","blue"),lty=1)
```



b) Calcular E_{in} y E_{test} (error sobre los datos de test).

```
print(paste("Error en la muestra con Regresion: ",E_in.PLA(datosTr,digitos.train,wr)))
```

```
## [1] "Error en la muestra con Regresion:  0.247685185185185"
```

```
print(paste("Error en la muestra con PLA-Pocket: ",E_in.PLA(datosTr,digitos.train,wp)))
```

```
## [1] "Error en la muestra con PLA-Pocket:  0.224537037037037"
```

```
print(paste("Error en test con Regresion: ",E_in.PLA(datosTst,digitos.test,wr)))
```

```
## [1] "Error en test con Regresion:  0.235294117647059"
```

```
print(paste("Error en test con PLA-Pocket: ",E_in.PLA(datosTst,digitos.test,wp)))
```

```
## [1] "Error en test con PLA-Pocket:  0.215686274509804"
```

c) Obtener cotas sobre el verdadero valor de E_{out}. Pueden calcularse dos cotas una basada en E_{in} y otra basada en E_{test}. Usar una tolerancia $\delta = 0,05$. ¿Que cota es mejor?

```
delta <- 0.05

print(paste("Cota de error en la muestra con Regresion: ",E_out(delta,nrow(datosTr),E_in.PLA(datosTr,di
```

```
## [1] "Cota de error en la muestra con Regresion:  0.313026874826826"
```

```
print(paste("Cota de error en la muestra con PLA-Pocket: ",E_out(delta,nrow(datosTr),E_in.PLA(datosTr,di
```

```
## [1] "Cota de error en la muestra con PLA-Pocket:  0.289878726678677"
```

```
print(paste("Cota de error en test con Regresion: ",E_out(delta,nrow(datosTst),E_in.PLA(datosTst,digitos
```

```
## [1] "Cota de error en test con Regresion:  0.425466368105043"
```

```
print(paste("Cota de error en test con PLA-Pocket: ",E_out(delta,nrow(datosTst),E_in.PLA(datosTst,digito
```

```
## [1] "Cota de error en test con PLA-Pocket:  0.405858524967788"
```

Podemos visualizar por los resultados que obtenemos una mejor cota utilizando PLA-Pocket que regresión linean y también obtenemos mejor resultado tanto en regresion lineal como con PLA-Pocket basandonos en E_{in} que en E_{test}. Podemos ver que aunque el E_{test} nos de un valor mejor que E_{in}, al tener menos datos, al calcular E_{out} tenemos un mayor error. No obstante obtenemos alrededor de un valor de 0.3 que es un valor bastante alto de error.