APRENDIZAJE AUTOMÁTICO

****

TRABAJO 1. PROGRAMACIÓN.

****

Objetivos:

- **Usando R.**
- **Algoritmo Perceptron**
- **Regresión lineal**

Autora: Cristina Zuheros Montes.

- Correo: zuhe18@gmail.com
- Github: https://github.com/cristinazuhe

Fecha: 28 Marzo 2016

********** SECCIÓN 2 **********

EJERCICIO DE GENERACIÓN Y VISUALIZACIÓN DE DATOS.

1. Construir una función lista = simula_unif(N, dim, rango) que calcule una lista de longitud N de vectores de dimensión dim conteniendo números aleatorios uniformes en el intervalo rango.

```
simula_unif <- function(N=3, dim=5, rang=0:9) {
  if(N<0){
    print("La lista no puede tener un tamaño negativo.")
    return(NA)
  }else if(dim<0){
    print("La dimension del vector no puede ser negativa.")
    return(NA)
  }
  ini_vector = runif(dim, rang[1], rang[length(rang)])
  mi_lista = list(ini_vector)
  if(N>1){
     for( i in 2:N){
       ini_vector = runif(dim, rang[1],rang[length(rang)])
       segunda = list(ini_vector)
       mi_lista = c(mi_lista, segunda)
     }
  }
  return(mi_lista)
}
```

Primero analizamos los casos en los que no tiene sentido definir la lista de vectores, que serían los casos en los que la longitud de la lista o la dimensión de los vectores sean negativos.

Si estamos en buenas condiciones de trabajo, creamos un vector aleatorio ini_vector que contendrá valores aleatorios de una distribución uniforme comprendidos en el rango de valores que indicamos por tercer parámetro (por defecto hemos decidido poner 0:9). Ahora añadimos el vector a nuestra lista y ya tendremos una lista con el vector.

Si nos piden que la lista tenga más de un vector, entraremos en el siguiente if donde vamos creando tantos vectores aleatorios de la distribución como se indique por primer parámetro a la función e iremos añadiéndolos a nuestra lista anterior. Esta será la lista que devolveremos.

Resultados:

```
> source('~/GitHub/MachineLearningR/Practica1/R/prac1.R')
[1] "Salima simula_unif por defecto:"
[[1]]
[1] 6.5036129 0.3710352 1.8988782 8.5917460 4.8360193

[[2]]
[1] 0.3527602 2.7137674 2.7093815 8.2064005 5.6588120

[[3]]
[1] 4.142056 4.900929 3.264778 7.401844 7.178341
```

Obtenemos la lista de vectores. Se observa que los valores son uniformes en el intervalo 0:9, posteriormente los representaremos con más valores y se verá más claro.

2. Construir una función lista = simula_gaus(N; dim; sigma) que calcule una lista de longitud N de vectores de dimensión dim conteniendo números aleatorios gaussianos de media 0 y varianzas dadas por el vector sigma.

```
simula_gaus <- function(N=3, dim=5, sigma=1:9){
  if(N<0){
    print("La lista no puede tener un tamaño negativo.")
    return(NA)
  }else if(dim<0){
    print("La dimension del vector no puede ser negativa.")
    return(NA)
  }else if(sigma[1]<0 || sigma[length(sigma)]<0){
    print("Sigma no puede ser negativo.")
    return(NA)
  }
  ini_vector = rnorm(dim, mean=0, sd=sample(sigma,1))
  mi_lista = list(ini_vector)
  if(N>1){
    for( i in 2:N){
      ini_vector = rnorm(dim, mean=0, sd=sample(sigma,1))
      segunda = list(ini_vector)
      mi_lista = c(mi_lista, segunda)
    }
  }
  return(mi_lista)
}
```

El procedimiento es similar al ejercicio anterior con la diferencia de que ahora usamos una distribución normal para obtener los valores aleatorios de los vectores de la lista. Para ello usamos el comando rnorm, al que le pasamos el número de valores que queremos (dim), la media, que nos piden que sea 0, y la varianza que ha de ser uno de los valores del intervalo de sigmas del tercer parámetro que se le pasa a la función: elegimos uno aleatoriamente.

Resultados:

```
[1] "Salida simula_gaus por defecto:"
[[1]]
[1]  6.025770  1.831739 -1.967894  9.178151 14.943582

[[2]]
[1]  6.085415 -2.205275 -5.462531 -3.003422 -2.043863

[[3]]
[1]  0.6566182 13.9100974  0.6347175 -5.3772803 -1.1469769
```

Con tan pocos datos es complicado confirmar que se trata de la distribución gaussiana, pero en ejercicios siguientes veremos la representación de más datos y se podrá ver que, efectivamente, los resultados son buenos.

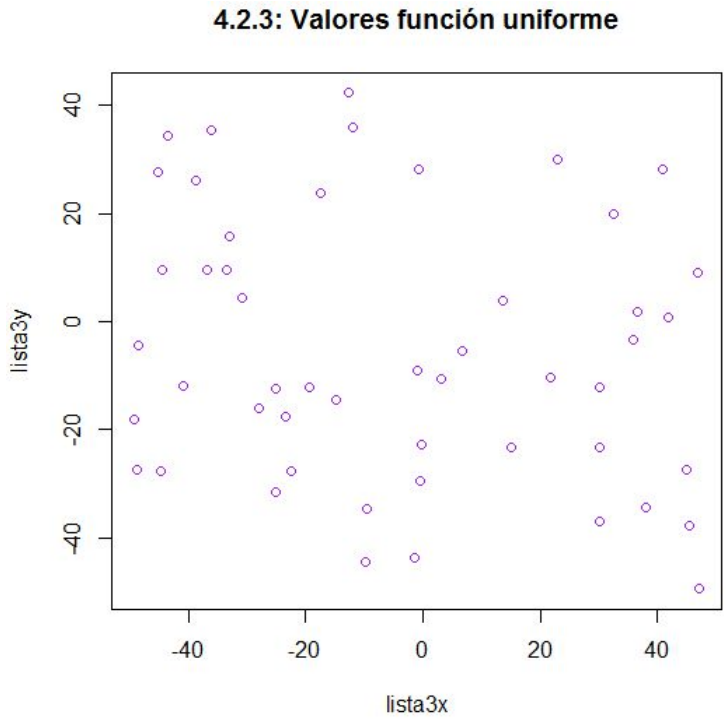
3. Suponer N = 50, dim = 2, rango = [□-50; +50] en cada dimensión. Dibujar una gráfica de la salida de la función correspondiente.

```
N3=50
dim3=2
intervalo3 = -50:50
lista3 = simula_unif(N3,dim3,intervalo3)
lista3x=NULL
lista3y=NULL

for(k in 1:N3){
  lista3x = c(lista3x,lista3[[k]][1])
  lista3y = c(lista3y,lista3[[k]][2])
}

#Represento los datos
plot(lista3x, lista3y,
     main = "4.2.3: Valores función uniforme",
     col="purple")
```

Generamos la lista de vectores con valores aleatorios uniformes (lista3).
Creamos una lista3x y una lista3y donde vamos almacenando los valores impares y pares respectivamente. Dibujamos la gráfica mediante el comando plot.

Resultados:

FIG 1:



Efectivamente se observa que los valores aleatorios se distribuyen aleatoriamente en el cuadrado rango*rango

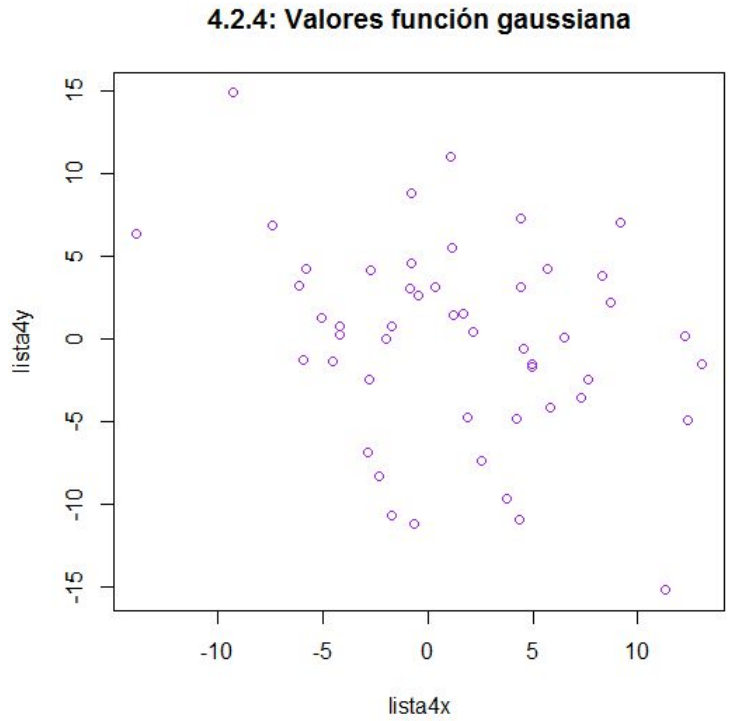
4. Suponer N = 50, dim = 2 y sigma = [5; 7] dibujar una gráfica de la salida de la función correspondiente.

```
N4=50
dim4=2
intervalo4 = 5:7
lista4 = simula_gaus(N4,dim4,intervalo4)
lista4x = NULL
lista4y= NULL

for(k in 1:N4){
    lista4x = c(lista4x,lista4[[k]][1])
    lista4y = c(lista4y,lista4[[k]][2])
}

plot(lista4x,lista4y,
     main = "4.2.4: Valores función gaussiana",
     col="purple")
```

Procedemos del mismo modo que en el ejercicio anterior, con la diferencia de que ahora los datos los obtenemos de la función gaussiana que hemos creado en el ejercicio 2 de esta sección.

Resultados:

FIG 2:



Vemos que ahora los valores se distribuyen de forma distinta, en concreto, conforme a funciones gaussianas con media 0 y sigmas entre 5 y 7.

5. Construir la función v=simula_recta(intervalo) que calcula los parámetros, v = (a; b) de una recta aleatoria, y = ax + b, que corte al cuadrado [□-50; 50] [□-50; 50] (Ayuda: Para calcular la recta simular las coordenadas de dos puntos dentro del cuadrado y calcular la recta que pasa por ellos)

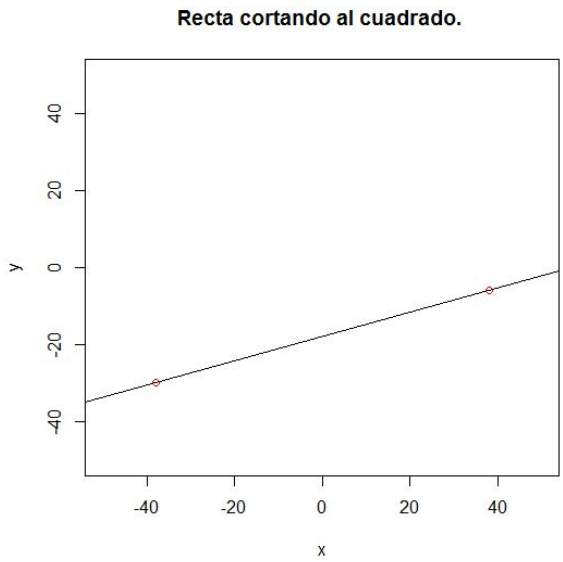
```
simula_recta <- function(intervalo=-50:50){
  #Tomo dos puntos aleatorios del cuadrado con un aleatorio
  primerx = sample(intervalo,1)
  primery = sample(intervalo,1)
  segundox = sample(intervalo, 1)
  while(segundox == primerx){ #Evito la división por 0
    segundox=sample(intervalo,1)
  }
  segundoy = sample(intervalo, 1)

  #Calculo a, b que definen la recta que pasa por los dos puntos
  a= ((segundoy - primery)/(segundox - primerx))
  b= (primery - (a*primerx))
  val=c(a,b)

  return(val) #devuelvo los coeficientes a y b
}
```

Tomamos dos puntos en el cuadrado, para ello consideramos números aleatorios comprendidos en el intervalo [-50,50] de modo que las coordenadas x de ambos puntos sean distintas para evitar la posterior división por 0.

A partir de los puntos, calculamos los coeficientes a y b que determinan la recta que pasa por ellos. Finalmente, devolvemos los coeficientes a y b de la recta.

Resultados:

FIG 3:



Esta gráfica la hemos generado para ver que realmente los coeficientes que obtenemos de la recta cortan al intervalo. No dejamos la representación en el código porque no nos la piden, y posteriormente la generamos junto a un conjunto de datos.

6. Generar una muestra 2D de puntos usando simula_unif() y etiquetar la muestra usando el signo de la función f(x; y) = y □- ax □- b de cada punto a una recta simulada con simula_recta(). Mostrar una gráfica con el resultado de la muestra etiquetada junto con la recta usada para ello.

```
val = simula_recta(-50:50)
a = val[1]
b = val[2]

#Genero los datos aleatorios uniformes
N6=50
dim6=2
intervalo6 = -50:50
lista6 = simula_unif(N6,dim6,intervalo6)
lista61x=NULL
lista61y=NULL

for(k in 1:N6){
  lista61x = c(lista61x,lista6[[k]][1])
  lista61y = c(lista61y,lista6[[k]][2])
}
```

En primer lugar simulamos una recta con la función anterior y obtenemos los coeficientes de la recta a y b.

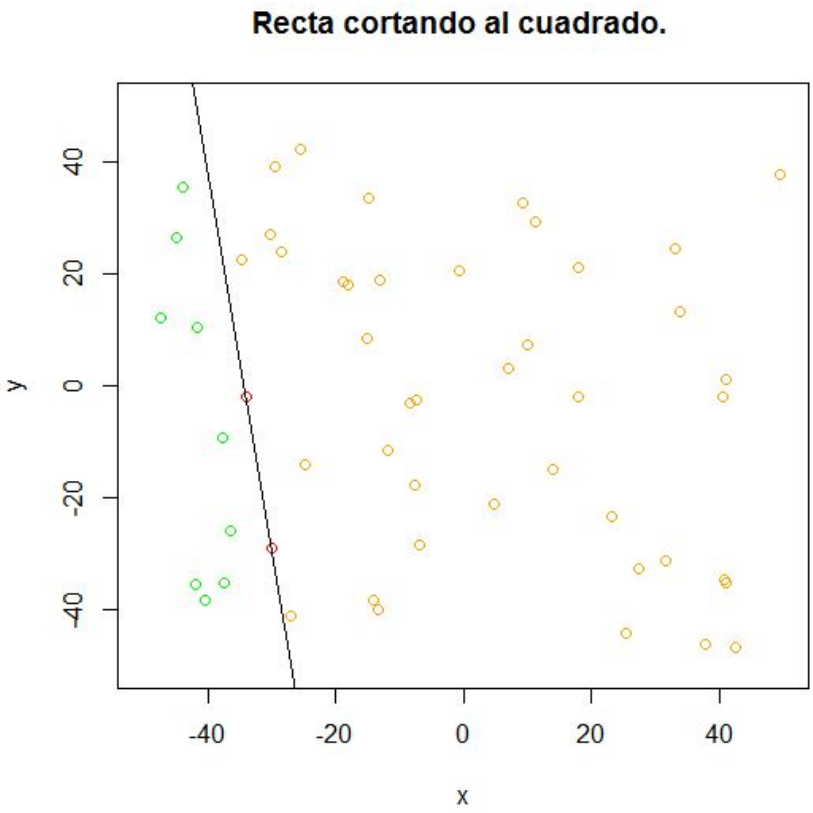
A continuación generamos los datos aleatorios uniformes haciendo uso de la función vista en el ejercicio 1 de esta misma sección. Almacenamos los valores en lista6 y, al igual que hacíamos antes, guardamos en lista61x y en lista61y los valores impares y pares respectivamente. Ya tenemos la recta y los valores a etiquetar.

```
plot(NULL,NULL, main="4.2.6:Simula recta. ",
     xlim = c(intervalo6[1], intervalo6[length(intervalo6)]),
     ylim = c(intervalo6[1], intervalo6[length(intervalo6)]))
abline(b,a)                                                    # R

#Etiqueto los datos mediante la función y las almaceno en eti
etiquetas6 = NULL                                              #Se
for(k in 1:length(lista61x)){
  num = lista61y[k] -a*lista61x[k] -b
  if(num>0){                                                   #va
    points(lista61x[k], lista61y[k],col= "orange")             #Lo
    etiquetas6 = c(etiquetas6, 1)
  }
  else{                                                        #va
    points(lista61x[k], lista61y[k], col="green")              #Lo
    etiquetas6 = c(etiquetas6, -1)
  }
}
```

Representamos la recta definida por los coeficientes a y b.
A continuación, guardamos las etiquetas de los datos en el vector etiquetas6: aplicamos la función a las listas lista61x y lista61y que contienen los datos. Si el resultado es positivo, ponemos la etiqueta de dicho dato a 1 y lo dibujamos en naranja. Si el resultado es negativo, ponemos la etiqueta a -1 y lo dibujamos en verde.

Resultados:

FIG 4:



Simplemente comentar que los resultados son tal cual se esperaban: la distribución de los datos es uniforme y la recta corta el cuadrado, dejando los puntos con etiquetas positivas y negativas bien separados.

7. Usar la muestra generada en el apartado anterior y etiquetarla con +1,-1 usando el signo de cada una de las siguientes funciones

- **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 -□ 20x² - 5x + 3**

Visualizar el resultado del etiquetado de cada función junto con su gráfica y comparar el resultado con el caso lineal ¿Qué consecuencias extrae sobre la forma de las regiones positiva y negativa?

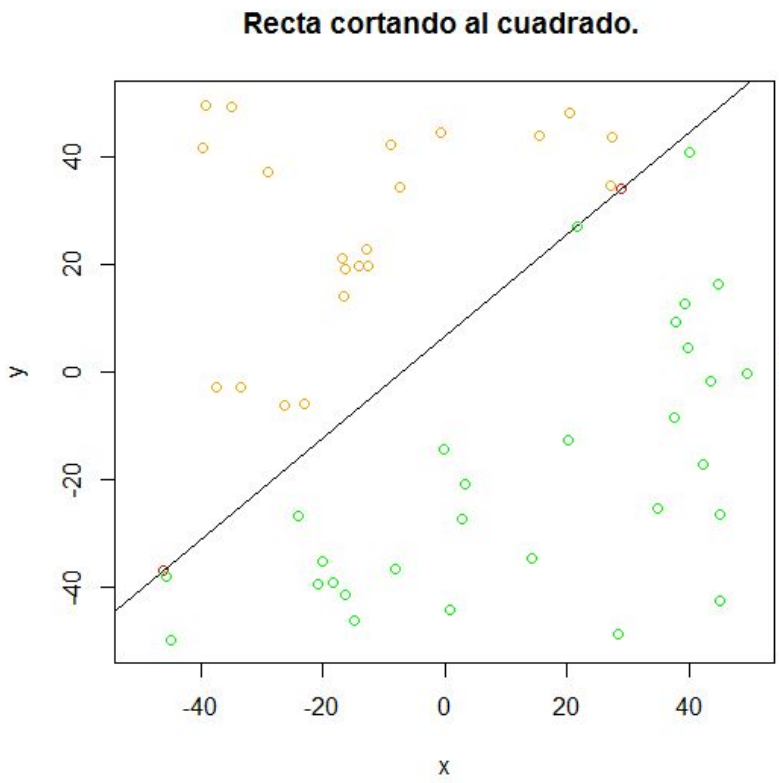
```
x7=intervalo6[1]
contador=0
while(contador< 1000){
  x7 = c(x7, x7[length(x7)] + 0.10)
  contador=contador+1;
}

#PRIMERA FUNCION.
#Valores en eje y de la primera funcion
y71a = 20 - sqrt(400-(x7-10)^2)
y71b = 20 + sqrt(400-(x7-10)^2)
#Dibujo la primera funcion en una nueva gráfica
plot(x7,y71a, col = "purple",
     xlim = c(intervalo6[1], intervalo6[length(intervalo6)]),
     ylim = c(intervalo6[1], intervalo6[length(intervalo6)]),
     main = "4.2.7:Primera funcion.", type="l")
points(x7, y71b, col="purple", type="l")

#Dibujo los puntos del apartado 6 etiquetándolos de acuerdo a
for(k in 1:length(lista61x)){
  num1 = (lista61x[k] - 10)^2 + (lista61y[k] - 20)^2 -400
  if(num1>0){
    points(lista61x[k], lista61y[k],col= "orange")
    etiquetas71 = c(etiquetas71, 1)
  }
  if(num1<0){
    points(lista61x[k], lista61y[k], col="green")
    etiquetas71 = c(etiquetas71, -1)
  }
}
```

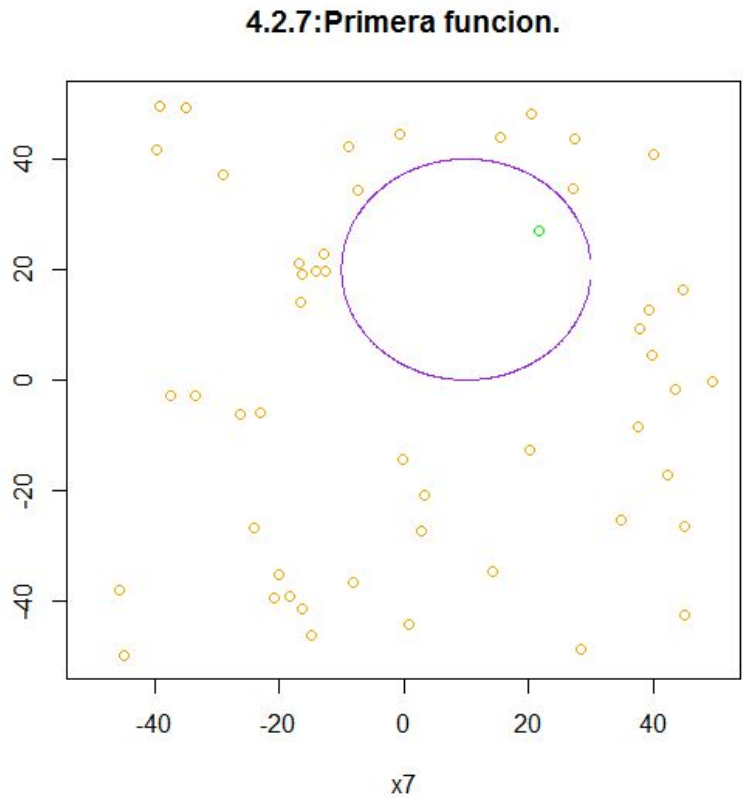
Trabajamos con la primera función:
En primer lugar vamos a dibujar la función, para ello creamos un vector x7 que contendrá valores en x comprendidos en el intervalo de trabajo. Creamos bastantes para que quede mejor. Este vector x7 será el mismo para todas las funciones.
Creamos los vectores y71a e y71b que contendrán los valores "y" generados por la primera función. Ahora ya podemos dibujar la primera función.

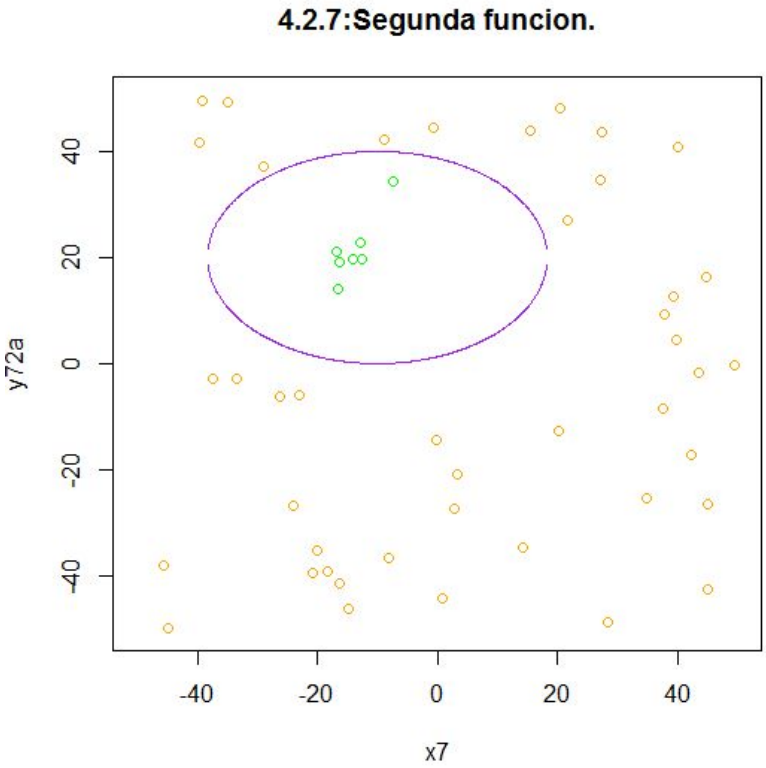
En el bucle for vamos etiquetado la muestra de datos de acuerdo a los valores que nos proporciona la función. Si el valor es positivo, lo etiquetamos a 1 y lo pintamos en naranja, igual que antes. Si el valor es negativo, lo etiquetamos a -1 y lo dibujamos en verde.

Para las demás funciones procedemos de forma análoga, cambiando la función y creando nuevos vectores de etiquetado.

Resultados:







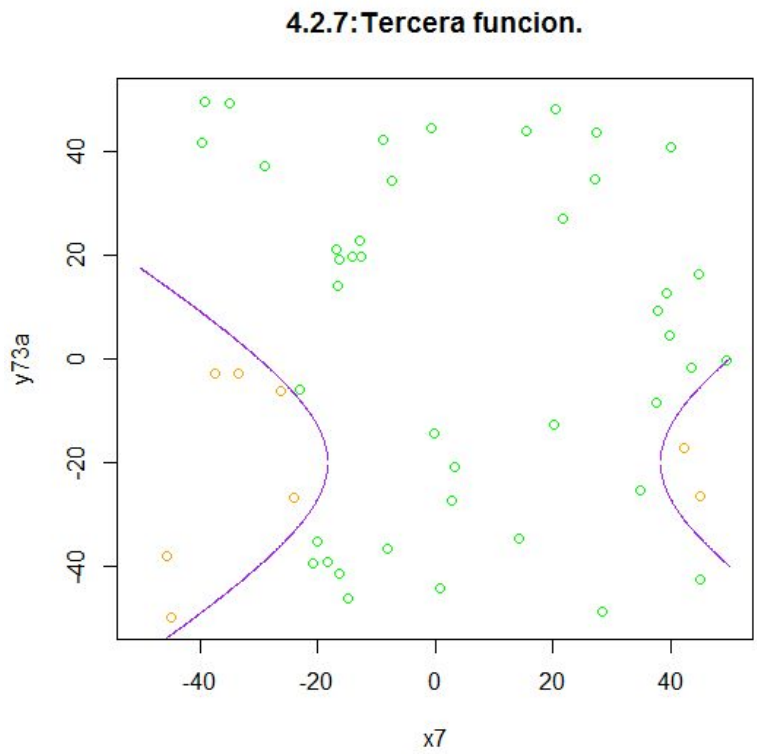
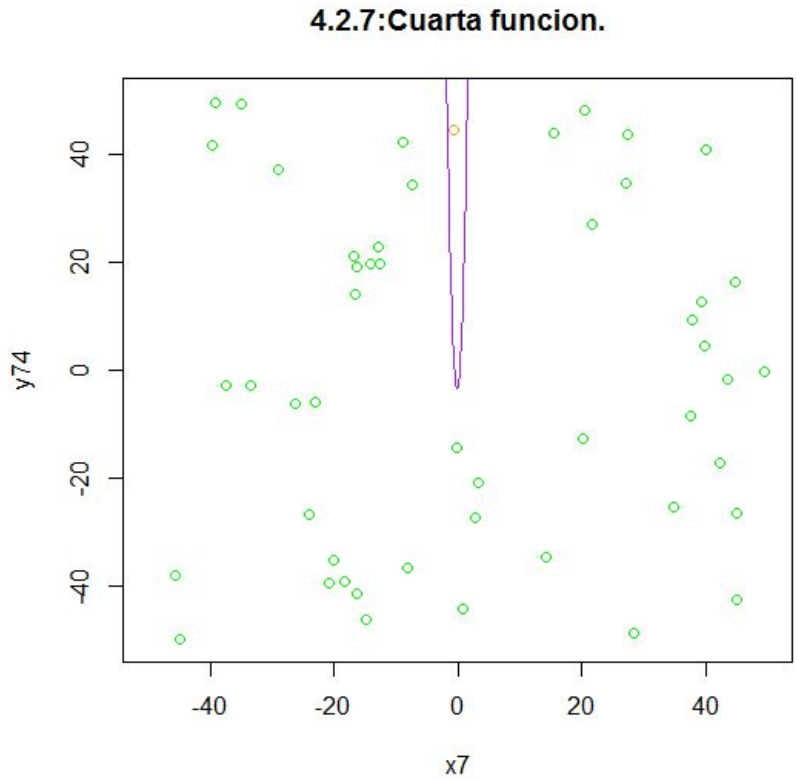


FIG 7

FIG 8

FIG 9



Mediante estas gráficas vemos que la elección de la función nos va restringiendo una mayor o menor región de muestras con etiquetas positivas/negativas.

Nos pueden servir para verificar que una misma muestra con leves cambios en las etiquetas, tendrán funciones bastante distintas que han hecho el etiquetado de las muestras. Es decir, una leve modificación en el etiquetado de una muestra puede llevarnos a buscar funciones muy diferentes.

8. Considerar de nuevo la muestra etiquetada en el apartado 6. Modifique las etiquetas de un 10% aleatorio de muestras positivas y otro 10% aleatorio de negativas.

A) **Visualice los puntos con las nuevas etiquetas y la recta del apartado6.**
B) **En una gráfica aparte visualice de nuevo los mismos puntos pero junto con las funciones del apartado7.**

Observe las gráficas y diga qué consecuencias extrae del proceso de modificación de etiquetas en el proceso de aprendizaje.

A)

```
etiquetas8 = etiquetas6
#Obtengo el número de muestras positivas y negativas a cambiar
#Un 10% de todas las que hay de positivas y negativas.
numeropositivos8 =length(which(etiquetas6 ==1))
numeronegativos8 = length(etiquetas6) - numeropositivos8

#positivosacamiar8 muestras han de pasar de etiquetado 1 a -1
positivosacambiar8 = numeropositivos8%/%10
#negativosacambiar8 muestras han de pasar de etiquetado -1 a 1
negativosacambiar8 = numeronegativos8%/%10

#Cambio positivos aleatorios a negativos.
aleatorio6 = NULL
for(j in 1:positivosacambiar8){
  aleatorio6 = sample(1:length(etiquetas8),1)
   while( etiquetas8[aleatorio6] != 1){
     aleatorio6 = sample(1:length(etiquetas8),1)
   }
  etiquetas8[aleatorio6] = -1;
}

#Cambio negativos aleatorios a positivos.
aleatorio6b = NULL
for(j in 1:negativosacambiar8){
  aleatorio6b = sample(1:length(etiquetas8),1)
  while( etiquetas8[aleatorio6b] != -1 &&
         etiquetas6[aleatorio6b] == etiquetas8[aleatorio6b] ){
    aleatorio6b = sample(1:length(etiquetas8),1)
  }
  etiquetas8[aleatorio6b] = 1;
}
```

En primer lugar obtenemos el número de etiquetas positivas y etiquetas negativas de la muestra etiquetada del ejercicio 6 de esta sección. El 10% de estos valores será el número de elementos que modificaremos, los almacenamos en positivosacambiar8 y negativosacambiar8.

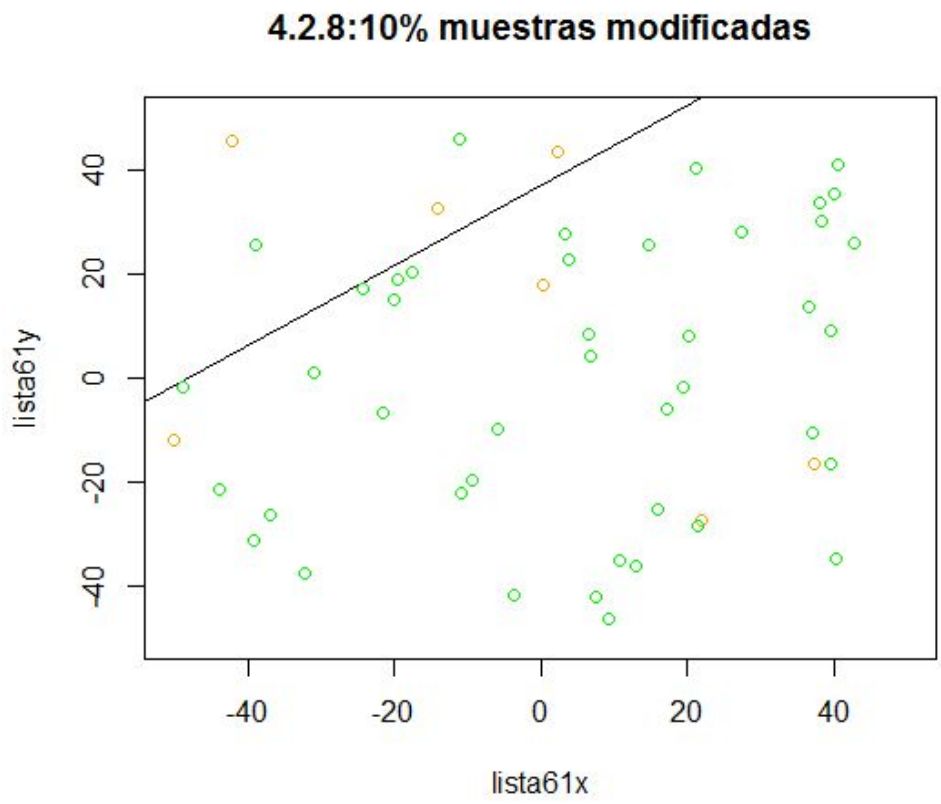
En etiquetas8 tenemos una copia de las muestras etiquetas del ejercicio 6. Tomamos un elemento aleatorio de esta lista. Mientras que este elemento sea distinto de 1, vamos generando otro elemento aleatorio de esta lista. Cuando ya tenemos un elemento de la lista etiquetado como 1, lo pasamos a -1. Hacemos este proceso positivosacambiar8 veces.

Realizamos un proceso análogo para pasar de etiquetas negativas a positivas.
Con esto ya tenemos en etiquetas8 las etiquetas de las muestras modificadas tal y como se nos pide.
A continuación, visualizamos las muestras modificadas junto con la recta anterior:

```
plot(lista61x, lista61y,
     xlim = c(intervalo6[1], intervalo6[length(intervalo6)]),
     ylim = c(intervalo6[1], intervalo6[length(intervalo6)]),
     col = (-etiquetas8 +5)/2, main="4.2.8:10% muestras modificadas")
abline(b,a) # Recta ax+b (pendiente a)(corte b)
```

Resultados:

FIG 10:



B)

```
plot(x7,y71a, col = "purple",
     xlim = c(intervalo6[1], intervalo6[length(intervalo6)]),
     ylim = c(intervalo6[1], intervalo6[length(intervalo6)]),
     main = "4.2.8(B). Primera funcion", type="l")
points(x7, y71b, col="purple", type="l")
points(lista61x, lista61y,
     col = (-etiquetas8 +5)/2, main="4.2.8(B):Primera funcion")
```

Veamos el código para la primera función:
En primer lugar dibujamos la función como hacíamos en el ejercicio 7 de esta misma sección. A continuación dibujamos las muestras etiquetas de acuerdo al vector etiquetas8 que hemos creado en el apartado anterior.

Procedemos del mismo modo para el resto de las funciones.

Resultados:

FIG 11:

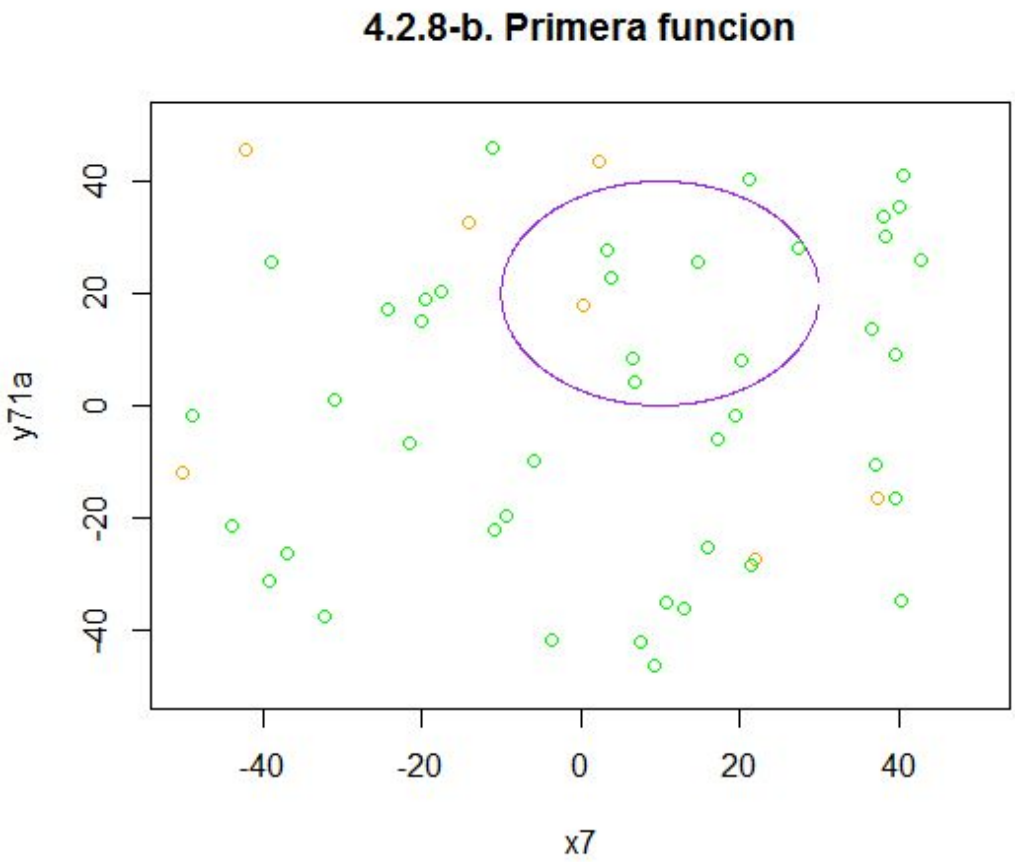


FIG 12:

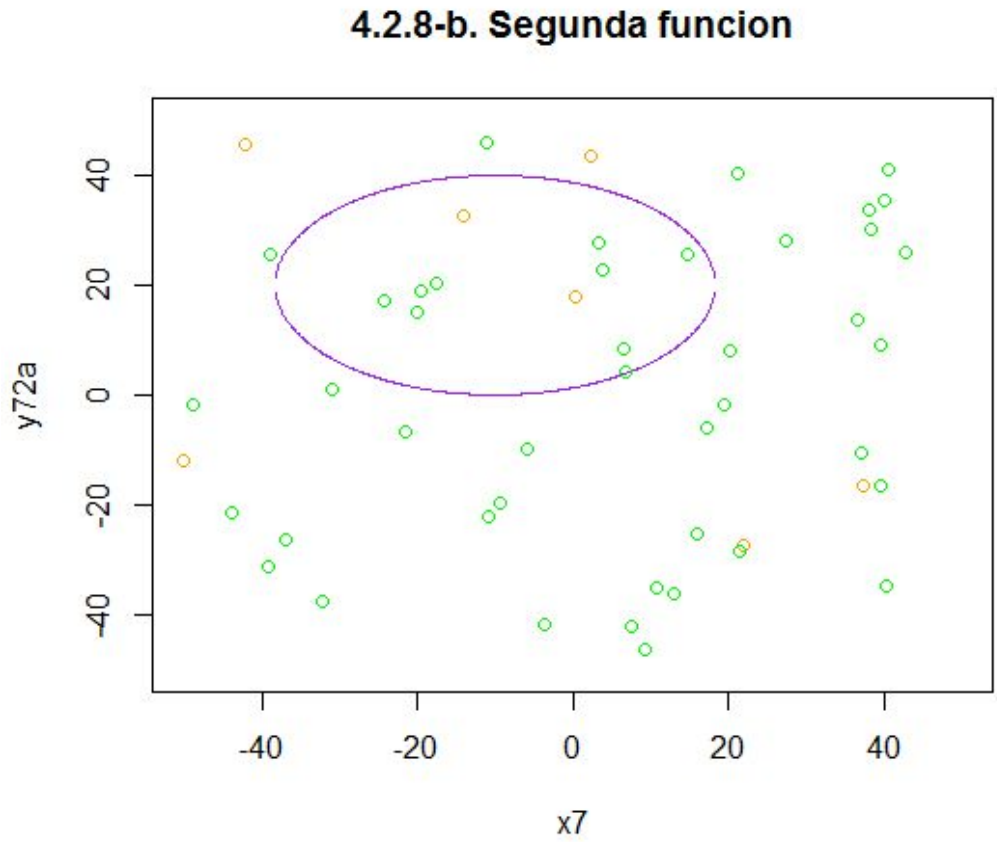

4.2.8-b. Segunda funcion
y72a
x7
40
20
0
-20
-40
-40
-20
0
20
40

FIG 13:

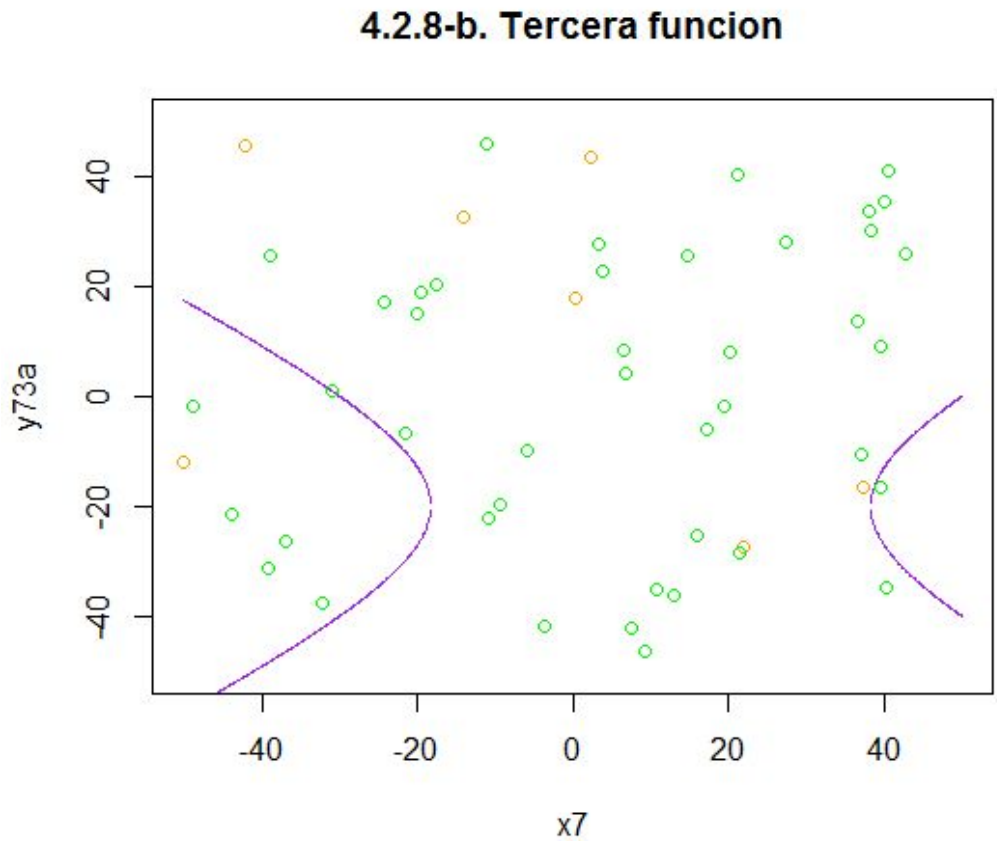
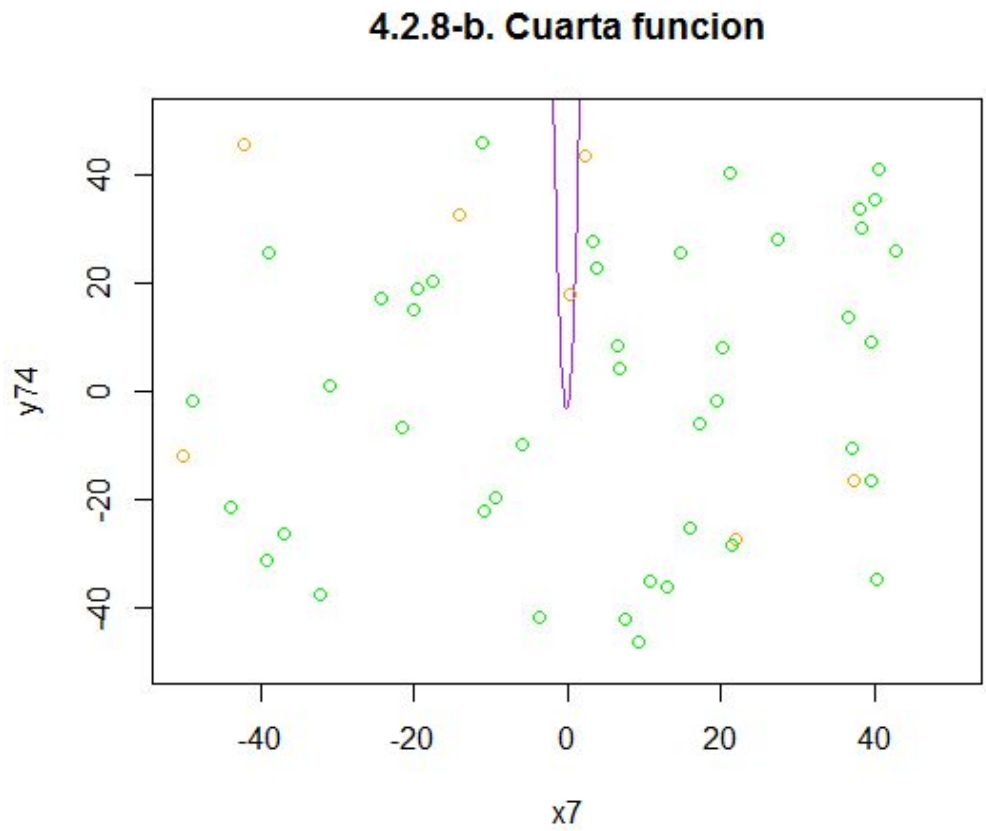

4.2.8-b. Tercera funcion
y73a
x7
40
20
0
-20
-40
-40
-20
0
20
40

FIG 14:



La muestra de datos con las etiquetas originales se encontraba perfectamente dividida por la recta que vemos en la FIG 10. Al modificar las etiquetas de dicha muestra de datos, su representación cambia y ya la misma recta no ajusta tan bien la separación de las etiquetas (ver FIG 10).

En las siguientes gráficas, FIG11-FIG14, tratamos de ver otra serie de funciones que ajusten la división de la muestra. Pero no son tampoco demasiado buenas, aunque con la función que vemos en la gráfica de la FIG14 encontramos unos resultados tan buenos como en la FIG10.

Esto nos lleva a verificar que si consideramos una función que ajuste muy bien una muestra etiquetada, no tiene porqué ajustar bien a otra muestra etiqueta que nos llegue posteriormente. Si no que podemos tener funciones que adapten no tan bien la muestra etiquetada primera de trabajo, pero adapten con gran precisión otra muestra etiquetada que en un principio desconocemos.

********** SECCIÓN 3 **********

EJERCICIO DE AJUSTE DEL ALGORITMO PERCEPTRON

1. Implementar la función sol = 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 salida sol devuelve los coeficientes del hiperplano.

```
ajusta_PLA <- function(datos, label, max_iter=10, vini){
  vini = c(vini, 1)
  d = dim(datos)
  datos = cbind(datos, rep(1,d[1]))
  contador = 0
  while(contador<max_iter){
    H = sign(datos%*%vini)
    indexa = which(H!=label) #Guarda los indices de los qu
    if(length(indexa)==0){
      print("Se ha encontrado un buen ajuste.")
      break
    }else{
      i = sample(indexa, 1)
      vini = vini + datos[i,]*label[i]
    }
    contador= contador+1
  }
  if(contador==max_iter)
    print("No se ha encontrado un buen ajuste.")
  print("Numero de iteracción en la que para: ")
  print(contador)
  return(vini)
}
```

En primer lugar añadimos al vector inicial otra coordenada final con el valor a 1, así como a la matriz de los datos. Establecemos un contador de iteraciones.

Mientras que dicho contador sea menor al número de iteraciones máximas que indicamos por parámetro, vamos a realizar el siguiente proceso:
Obtenemos las muestras que están mal etiquetas con respecto al vector de hiperplano (vini). Si no hay ninguna mal etiquetada, hemos acabado el algoritmo. Si hay alguna mal etiqueta, tomamos una muestra de las mal etiquetadas y hacemos que el hiperplano cambie para que se encuentre bien etiquetada.

Finalmente mostramos el número de iteraciones que han sido necesarias para el ajuste.

Resultados:
Veremos la ejecución del algoritmo en el siguiente ejercicio.

2. Ejecutar el algoritmo PLA con los valores simulados en apartado.6 del ejercicio.4.2, inicializando el algoritmo con el vector cero y con vectores de números aleatorios en [0; 1] (10 veces). Anotar el número medio de iteraciones necesarias en ambos para converger. Valorar el resultado

```
matriz_datoss2e2 = cbind(lista61x, lista61y) #creo la matriz de datos
mi_labels2e2 = etiquetas6
max_itera100 = 100
d = dim(matriz_datoss2e2)

vector_inicial0 = rep(0,d[2])
writeLines("\n***Ajuste PLA con vector inicial 0:***")
print(ajusta_PLA(matriz_datoss2e2,mi_labels2e2, max_itera100, vector_inicial0))

for(i in 1:10){
writeLines("\n***Ajuste PLA con vector aleatorio:***")
    vector_inicial_random = runif(d[2],0,1)
    print(vector_inicial_random)
    ajusta_PLA(matriz_datoss2e2,mi_labels2e2, max_itera100, vector_inicial_random)
}
```

Simplemente consideramos una matriz con las muestras del ejercicio 6 de la sección anterior y creamos vectores iniciales del hiperplano (vector cero y 10 vectores aleatorios).

Resultados.

```
***Ajuste PLA con vector inicial 0:***
[1] "Se ha encontrado un buen ajuste."
[1] "Numero de iteracción en la que para: "
[1] 7
lista61x lista61y
57.45086 60.41278  2.00000

***Ajuste PLA con vector aleatorio:***
[1] 0.3843500 0.8811348
[1] "Se ha encontrado un buen ajuste."
[1] "Numero de iteracción en la que para: "
[1] 14

***Ajuste PLA con vector aleatorio:***
[1] 0.1255381 0.8421044
[1] "Se ha encontrado un buen ajuste."
[1] "Numero de iteracción en la que para: "
[1] 6

***Ajuste PLA con vector aleatorio:***
[1] 0.9058284 0.9158915
[1] "Se ha encontrado un buen ajuste."
[1] "Numero de iteracción en la que para: "
[1] 0

***Ajuste PLA con vector aleatorio:***
[1] 0.62986461 0.01032778
[1] "Se ha encontrado un buen ajuste."
[1] "Numero de iteracción en la que para: "
[1] 11

***Ajuste PLA con vector aleatorio:***
[1] 0.6245865 0.4592553
[1] "Se ha encontrado un buen ajuste."
[1] "Numero de iteracción en la que para: "
[1] 6

***Ajuste PLA con vector aleatorio:***
[1] 0.5197796 0.8225719
[1] "Se ha encontrado un buen ajuste."
[1] "Numero de iteracción en la que para: "
[1] 8

***Ajuste PLA con vector aleatorio:***
[1] 0.60071574 0.02603909
[1] "Se ha encontrado un buen ajuste."
[1] "Numero de iteracción en la que para: "
[1] 2

***Ajuste PLA con vector aleatorio:***
[1] 0.08804635 0.07374026
[1] "Se ha encontrado un buen ajuste."
[1] "Numero de iteracción en la que para: "
[1] 7

***Ajuste PLA con vector aleatorio:***
[1] 0.9771887 0.1474996
[1] "Se ha encontrado un buen ajuste."
[1] "Numero de iteracción en la que para: "
[1] 13

***Ajuste PLA con vector aleatorio:***
[1] 0.5667559 0.5270397
[1] "Se ha encontrado un buen ajuste."
[1] "Numero de iteracción en la que para: "
[1] 0
```

En las capturas anteriores podemos ver una ejecución completa del algoritmo para el vector inicial 0 y vectores iniciales aleatorios. En media los resultados son para:
Vector nulo: 7
Vector aleatorio: 6.7

Probando más ejecuciones vemos los siguientes resultados:
Segunda ejecución:
    Vector nulo:100
    Vector aleatorio:100
Tercera ejecución:
    Vector nulo: 5
    Vector aleatorio: 14.7

En la segunda ejecución para los dos vectores llegamos a realizar 100 iteraciones. Este es el límite que hemos impuesto, luego nos quiere decir que no llegamos a encontrar el hiperplano que separa a la muestra.
En cuanto a la diferencia entre usar un vector inicial nulo y usar un vector inicial aleatorio vemos que, parece ser mejor, un vector inicial nulo. De modo que para futuros ejercicios, si no nos piden lo contrario, usaremos el vector inicial como nulo.

3. Ejecutar el algoritmo PLA con los datos generados en el apartado.8 del ejercicio.4.2, usando valores de 10, 100 y 1000 para max_iter. Etiquetar los datos de la muestra usando la función solución encontrada y contar el número de errores respecto de las etiquetas originales. Valorar el resultado.

```
max_iter10 = 10
max_iter1000 = 1000
matriz_datoss2e3 = cbind(lista61x, lista61y) #creo la matriz de datos
etiquetasorigs2e3 = etiquetas8
intervalors2e3 = intervalo6
d = dim(matriz_datoss2e3)

#Representamos las muestras con las etiquetas originales
plot(matriz_datoss2e3[,1], matriz_datoss2e3[,2],main="4.3.3.Etiquetas antes
       xlim = c(intervalors2e3[1], intervalors2e3[length(intervalors2e3)]),
       ylim = c(intervalors2e3[1], intervalors2e3[length(intervalors2e3)]),
       col = (-etiquetasorigs2e3 +5)/2)

#Realizo PLA
vector_inicial0 = rep(0,d[2])
writeLines("\n***Ajuste PLA con vector inicial 0:***")
hiperplanos3e3i10 = ajusta_PLA(matriz_datoss2e3,etiquetasorigs2e3, max_iter1
coefas2e3 = -hiperplanos3e3i10[3]/hiperplanos3e3i10[2]
coefbs2e3 = -hiperplanos3e3i10[1]/hiperplanos3e3i10[2]

#Etiqueto las muestras en funcion del hiperplano
etiquetass2e3 = NULL                                    #Será un vector con va
for(k in 1:length(matriz_datoss2e3[,1])){
  numi = matriz_datoss2e3[k,2] -coefas2e3*matriz_datoss2e3[k,1] -coefbs2e3
  if(numi>0)                                          #valores positivos de la f
     etiquetass2e3 = c(etiquetass2e3, 1)
  else                                          #valores negativos de la funcior
    etiquetass2e3 = c(etiquetass2e3, -1)
}
```

En primer lugar obtenemos la muestra del ejercicio 6 con las etiquetas que producimos en el ejercicio 8: tenemos una muestra que no es linealmente separable. Posteriormente, representamos dicha muestra etiquetada para después poder ver bien el cambio que se produce al aplicar PLA.

Sobre la muestra no linealmente separable, aplicamos el algoritmo PLA visto en el ejercicio anterior (aquí usamos 10 iteraciones). Una vez tenemos los coeficientes del hiperplano, creamos un nuevo etiquetado de las muestras en base a él.

```
plot(matriz_datoss2e3[,1], matriz_datoss2e3[,2],main="4.3.3.Etiquetas tras
     xlim = c(intervalors2e3[1], intervalors2e3[length(intervalors2e3)]),
     ylim = c(intervalors2e3[1], intervalors2e3[length(intervalors2e3)]),
     col = (-etiquetass2e3 +5)/2)
abline(coefbs2e3,coefas2e3) # Recta ax+b (pendiente a)(corte b)

#Obtengo etiquetas cambiadas por PLA
etiquetas_cambiadas = which(etiquetasorigs2e3!=etiquetass2e3)
num_etiquetas_cambiadas = length(etiquetas_cambiadas)
print("Número de etiquetas cambiadas: ")
print(num_etiquetas_cambiadas)
```

Una vez tenemos la muestra con las nuevas etiquetas generadas por el hiperplano, las visualizamos.
Finalmente obtenemos el número de etiquetas que han sido cambiadas.

Resultados:

Para 10 iteraciones (Número de etiquetas cambiadas):

Primera ejecución:28
Segunda ejecución:19
Tercera ejecución:26

Para 100 iteraciones (Número de etiquetas cambiadas):

Primera ejecución:27
Segunda ejecución:21
Tercera ejecución:18

Para 1000 iteraciones (Número de etiquetas cambiadas):

Primera ejecución:16
Segunda ejecución:31
Tercera ejecución:25

Tal y como esperábamos, los resultados que obtenemos son malos, pues las muestras no pueden ser separadas por un hiperplano. Por más iteraciones que realicemos no vamos a conseguir mejores resultados. Veamos también un par de gráficas para ver que el ajuste no tiene ningún valor.

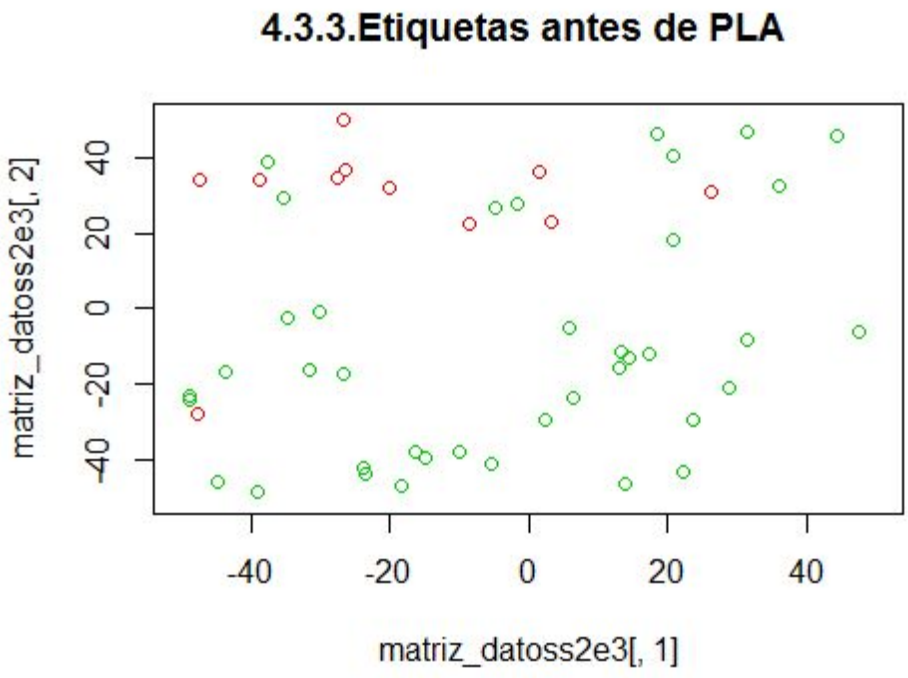


FIG 15

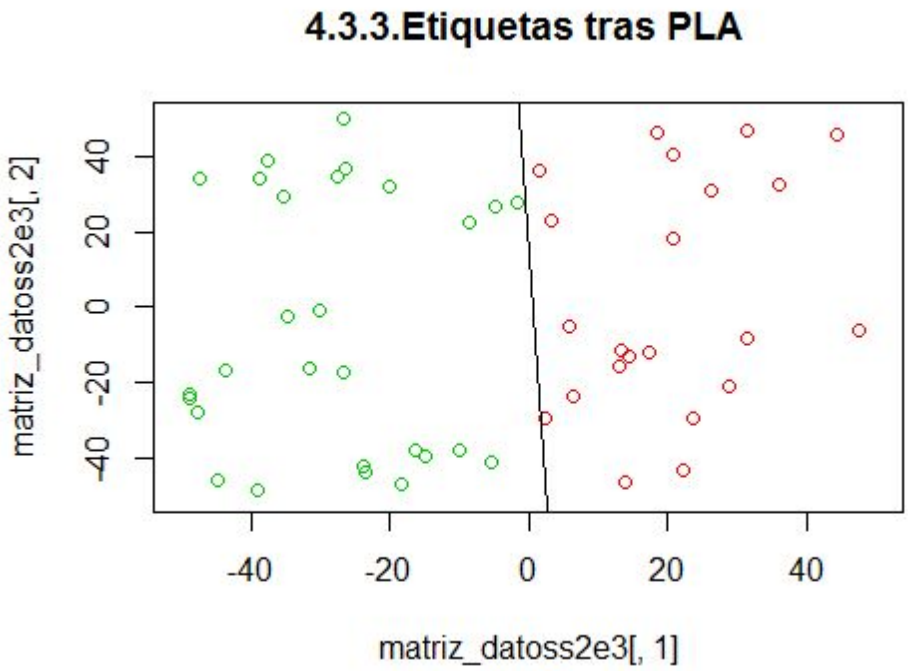

4.3.3.Etiquetas tras PLA
matriz_datoss2e3[, 2]
40
20
0
-20
-40
-40
-20
0
20
40
matriz_datoss2e3[, 1]

FIG 16

4. Repetir el análisis del punto anterior usando la primera función del apartado.7 del ejercicio.4.2

Vamos a omitir el código pues es análogo al ejercicio anterior pero usando etiquetas71 (etiquetas del ejercicio 7, usando la primera función que nos daban)

Resultados:

Para 10 iteraciones (Número de etiquetas cambiadas):

Primera ejecución:31

Segunda ejecución:33

Tercera ejecución:27

Para 100 iteraciones (Número de etiquetas cambiadas):

Primera ejecución:24

Segunda ejecución:31

Tercera ejecución:38

Para 1000 iteraciones (Número de etiquetas cambiadas):

Primera ejecución:28

Segunda ejecución:22

Tercera ejecución:35

Vemos algunas representaciones:

FIG 17

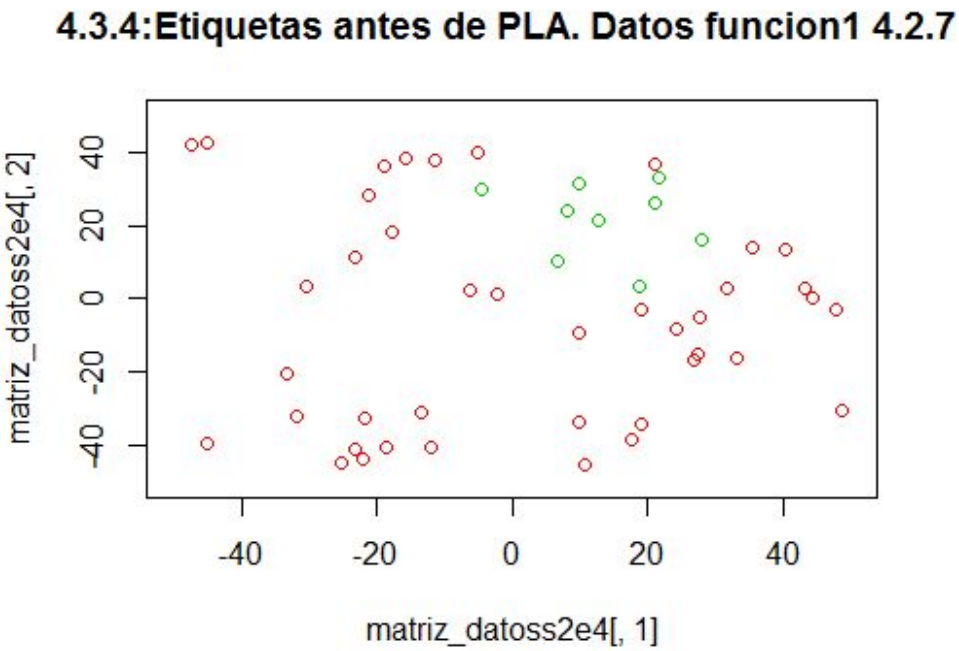
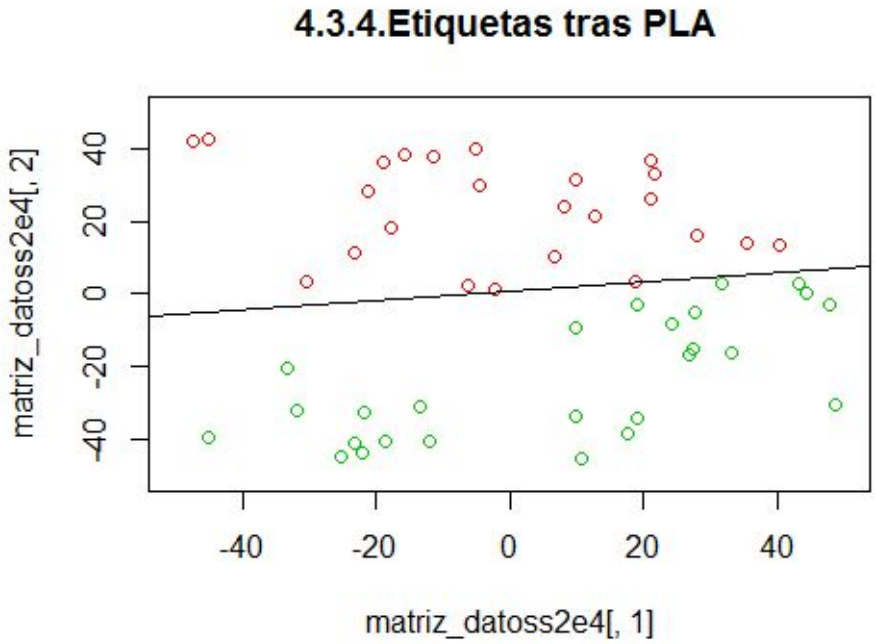


FIG 18



Al igual que en el ejercicio anterior, los resultados son pésimos pues no tiene mucho sentido que apliquemos dicho algoritmo (los datos vuelven a no ser linealmente separables). Da igual el número de iteraciones que realicemos que el resultado seguirá siguiendo malo.

5. Modifique la función ajusta_PLA para que le permita visualizar los datos y soluciones que va encontrando a lo largo de las iteraciones. Ejecute con la nueva versión el apartado.3 del ejercicio.4.2

```
PLA_grafica <- function(datos, label, max_iter=10, vini, intervalo){
  #Dibujo los datos etiquetados
  plot(datos[,1], datos[,2],main="4.3.5: Ajuste PLA",
       xlim = c(intervalo[1], intervalo[length(intervalo)]),
       ylim = c(intervalo[1], intervalo[length(intervalo)]),
       col = (-label +5)/2)

  vini = c(vini, 1)
  d = dim(datos)
  datos = cbind(datos, rep(1,d[1]))
  contador = 0

  while(contador<max_iter){
    H = sign(datos%*%vini)
    indexa = which(H!=label) #Guarda los indices de los que son distintos
    if(length(indexa)==0){
      print("Se ha encontrado un buen ajuste.")
      break
    }else{
      i = sample(indexa, 1)
      vini = vini + datos[i,]*label[i]
      abline(-vini[3]/vini[2], -vini[1]/vini[2], col="thistle1")
    }
    contador= contador+1
  }
  if(contador==max_iter)
    print("No se ha encontrado un buen ajuste.")

  H = sign(datos%*%vini)
  indexa = which(H!=label) #Guarda los indices de los que son distintos
  print("Numero de errores:")
  print(length(indexa))

  abline(-vini[3]/vini[2], -vini[1]/vini[2], col="thistle3")
  print("Vector de parada:")
  vini
```

A diferencia de la función PLA que implementamos anteriormente, en esta lo primero que hacemos es representar las muestras junto con sus etiquetas.

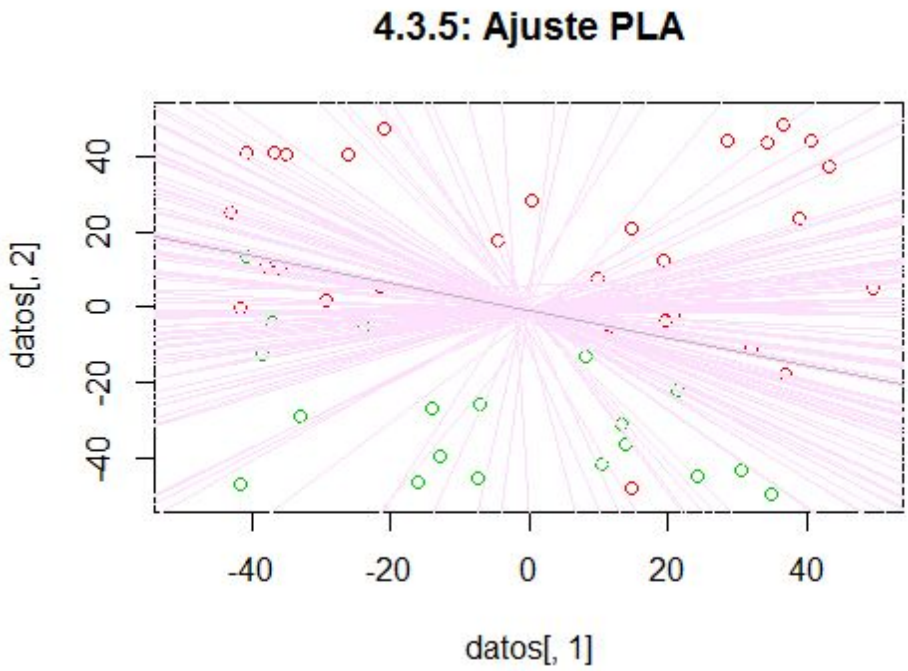
Conforme vamos encontramos un nuevo vector que define al hiperplano de separación, vamos representándolo en la misma gráfica de las muestras.

El hiperplano final lo representamos en un tono más oscuro para que destaque. Además contabilizamos el número de errores que se obtienen al considerar dicho hiperplano final.

Resultados:

```
[1] "**********************************Ejercicio 4.3.5**************************

***Ajuste PLA con vector inicial 0:***
[1] "No se ha encontrado un buen ajuste."
[1] "Numero de errores:"
[1] 8
[1] "Vector de parada:"
lista61x lista61y
20.96882 58.28446 57.00000
```

FIG 19



Vemos que, al igual que antes, el hiperplano nunca va a conseguir llegar a ajustar bien a la muestra etiquetada. En este caso tenemos 8 datos que se encuentran mal con respecto al hiperplano.

6. A la vista de la conducta de las soluciones observada en el apartado anterior, proponga e implemente una modificación de la función original sol = ajusta_PLA_MOD(:::) que permita obtener soluciones razonables sobre datos no linealmente separables. Mostrar y valorar el resultado encontrado usando los datos del apartado.7 del ejercicio.4.2

```
PLA_grafica_MOD <- function(datos, label, max_iter=10, vini, intervalo){
  #Dibujo los datos etiquetados
  plot(datos[,1], datos[,2],main="4.3.6: Ajuste PLA",
       xlim = c(intervalo[1], intervalo[length(intervalo)]),
       ylim = c(intervalo[1], intervalo[length(intervalo)]),
       col = (-label +5)/2)

  vini = c(vini, 1)
  d = dim(datos)
  datos = cbind(datos, rep(1,d[1]))
  contador = 0
  mejorvini=NULL
  mejorbuenas=length(label)
  indexa=NULL
  while(contador<max_iter){
    H = sign(datos%*%vini)
    indexa = which(H!=label) #Guarda los indices de los que son distintos
    if(length(indexa)==0){
      print("Se ha encontrado un buen ajuste.")
      break
    }else{
      i = sample(indexa, 1)
      vini = vini + datos[i,]*label[i]
      if(length(which(sign(datos%*%vini)!=label))<mejorbuenas){
        H = sign(datos%*%vini)
        indexa = which(H!=label) #Guarda los indices de los que son distintos
        mejorvini=vini
        mejorbuenas=length(indexa)
      }
      abline(-vini[3]/vini[2], -vini[1]/vini[2], col="thistle1")

    }
    contador= contador+1
  }
  if(contador==max_iter)
    print("No se ha encontrado un buen ajuste.")

  print("Numero de errores:")
  print(mejorbuenas)
  abline(-mejorvini[3]/mejorvini[2], -mejorvini[1]/mejorvini[2], col="thistle3")
  print("Vector de parada:")
  mejorvini
}


print("*********************************Ejercicio 4.3.6*****************************")
matriz_datoss2e6 = matriz_datoss2e3 #creo la matriz de datos
mi_labels2e6 = mi_labels2e5
ds2e6 = dim(matriz_datoss2e6)

vector_inicial0s2e6 = rep(0,ds2e6[2])
writeLines("\n***Ajuste PLA con vector inicial 0:***")
print(PLA_grafica_MOD(matriz_datoss2e6,mi_labels2e6, max_itera100,
                      vector_inicial0s2e6, intervalors2e3))
```

En esta nueva implementación, actualizo el vector del hiperplano sólamente si mejora el ajuste de las muestras. Para ello hago uso de mejorvini que nos irá guardando el mejor vector del hiperplano y mejorbuenas que nos irá guardando el menor número de etiquetas que se encuentran cambiadas por el hiperplano.

Resultados:

FIG 20

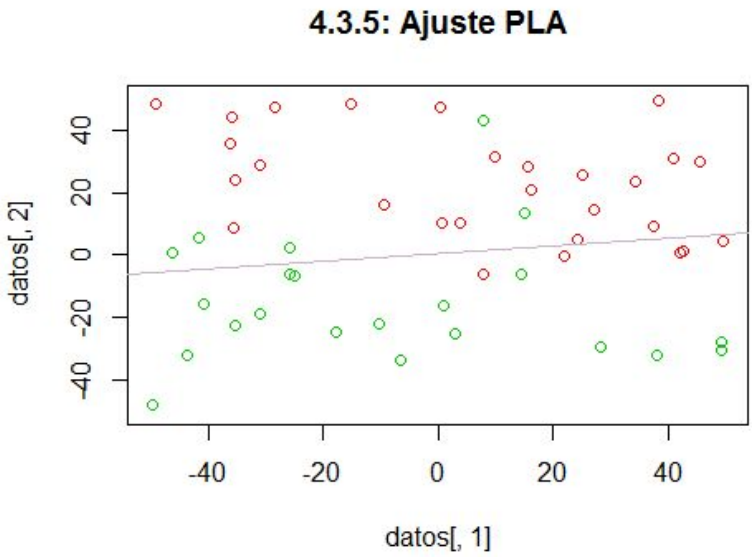
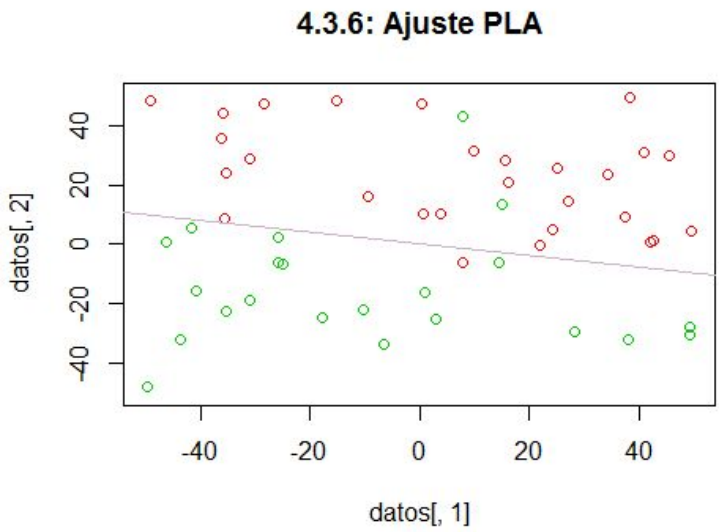


FIG 21



En la fig 21 hemos aplicado el algoritmo sin la mejora y obtenemos un error de 10 muestras. Al aplicar el algoritmo visto en este ejercicio obtenemos un error de 3 muestras. Por tanto, se verifica que nuestro algoritmo modificado ha mejorado al algoritmo PLA de los ejercicios anteriores.
Hemos hecho varias pruebas y prácticamente siempre conseguimos mejores resultados.

********** SECCIÓN 4 **********

EJERCICIO SOBRE REGRESIÓN LINEAL.

1. Abra el fichero ZipDigits.info disponible en la web del curso y lea la descripción de la representación numérica de la base de datos de números manuscritos que hay en el fichero ZipDigits.train.

Tras ver el fichero zip.train, vemos que hay una gran cantidad de datos que representan distintos números. Vemos que cada fila representa una matriz que representa al número en cuestión.

2. Lea el fichero ZipDigits.train dentro de su código y visualize las imágenes. Seleccione solo las instancias de los números 1 y 5. Guardelas como matrices de tamaíso 16x16.

```
lectura_fichero <- read.table("zip.train", header=FALSE)

#indices de los vectores de datos que representan 1's o 5's
indicess3e2= which((lectura_fichero[,1]) ==5 |  (lectura_fichero[,1]) ==1)

#matrizdatostodos es una lista con las matrices que representan las imagenes de 1's y 5's
matrizdatostodos = NULL
for(k in 1:length(indicess3e2)){
    matrizdatos=matrix(as.numeric(lectura_fichero[indicess3e2[k],2:ncol(lectura_fichero)]),
                       nrow=16,ncol=16)
    matrizdatostodos = c(matrizdatostodos, list(matrizdatos))
}
image(matrizdatostodos[[100]]) #Así veo la imagen (matriz) 100 de la lista
```

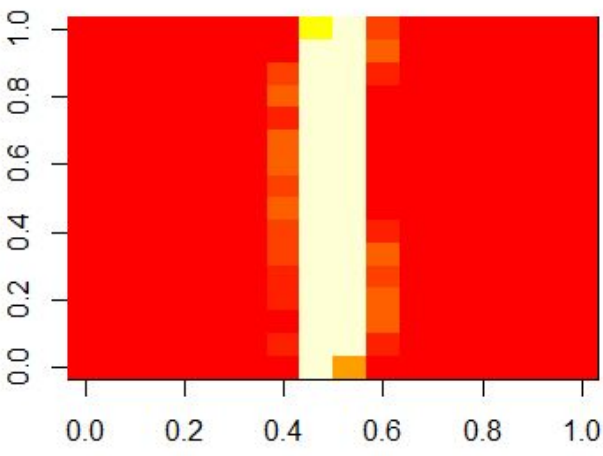
Leemos el fichero usando read.table y nos quedamos con aquellas filas del ficheros que representen 1's o 5's: serán aquellas filas cuyo primer elemento sea un 1 o un 5. Los almacenamos en indicess3e2.

A continuación, para cada fila del fichero que representa un 1 o un 5, creamos una matriz 16x16 que contendrá los valores que representan al número. Vamos guardando estas matrices en matrizdatostodos.

Para visualizar las imágenes usamos el comando image. Aquí sólo visualizamos la imagen 100 de la lista de matrices de números.

Resultados:

Vemos que dicha matriz representa a un 1.



3. Para cada matriz de números calcularemos su valor medio y su grado de simetría vertical. Para calcular la simetría calculamos la suma del valor absoluto de las diferencias en cada píxel entre la imagen original y la imagen que obtenemos invertiendo el orden de las columnas. Finalmente le cambiamos el signo.

```
vectormedias = NULL
for(k in 1:length(indicess3e2)){
   vectormedias = c(vectormedias, mean(apply(matrizdatostodos[[k]],1, mean)))
}

#Vector con grado de simetria vertical para cada matriz
total=ncol(matrizdatostodos[[1]])
mitad= total/2
vectorsimetrias =NULL
for(k in 1:length(indicess3e2)){
   sumadifabs = 0
   for(i in 1: mitad){
      vectorderecho = matrizdatostodos[[k]][,i]
      vectorizquierdo=matrizdatostodos[[k]][,total - i+1]
      sumadifabs = sumadifabs + 2*(sum(abs(vectorderecho - vectorizquierdo)))
   }
   vectorsimetrias = c(vectorsimetrias, sumadifabs)
}
vectorsimetrias = -vectorsimetrias
```

En primer lugar creamos el vector de medias haciendo la media de las filas de las columnas, y luego la media de estos valores.

En segundo lugar creamos el vector de simetría vertical. Para ello calculo la suma del valor absoluto de las diferencias de los pixeles por la izquierda y por la derecha de la imagen (es simetría vertical). Como nos piden que le cambiemos el signo, finalmente lo cambiamos.

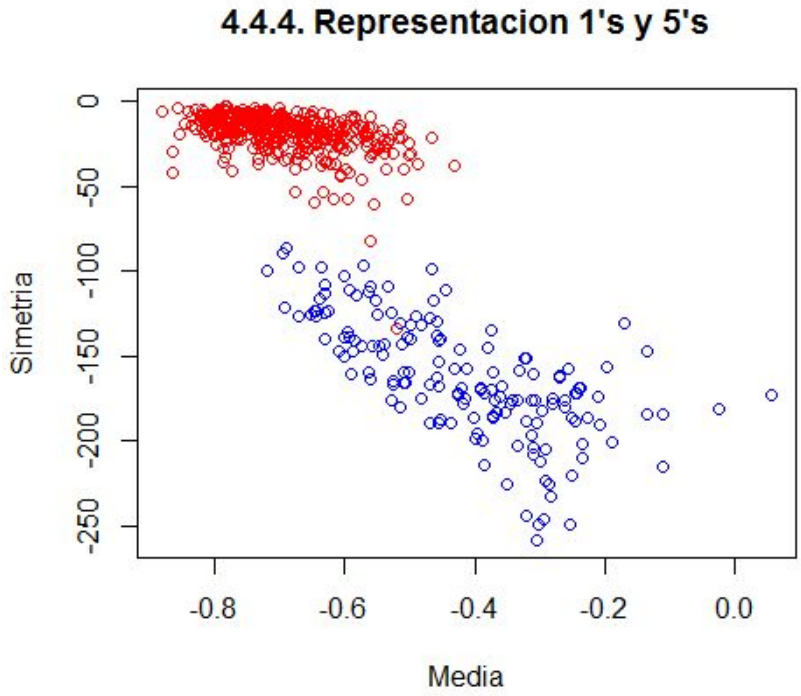
Resultados:
En este apartado no hay resultados que mostar.

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

```
matrizdatoss3e4 = cbind(vectormedias, vectorsimetrias)
etiquetass3e4 = lectura_fichero[indicess3e2,1]
plot(matrizdatoss3e4[,1], matrizdatoss3e4[,2],
     main="4.4.4. Representacion 1's y 5's", col=(etiquetass3e4-1)/2 +2)
```

Para representar intensidad y simetría creamos una matriz con ambos vectores. Además creamos un vector con las etiquetas que serán los 1's y 5's.

Resultados:



Visualmente ya vemos que usando la media y la simetría podemos distinguir con cierta fiabilidad entre lo que es un 1 y lo que es un 5.

5. Implementar la función sol = Regress_Lin(datos; label) que permita ajustar un modelo de regresion lineal (usar SVD). Los datos de entrada se interpretan igual que en clasificación.

```
regresion <- function(MatrizDatos, Etiquetas){
  for(i in 1:length(Etiquetas)){
   if(Etiquetas[i] == 5)
     Etiquetas[i] =-1
  }
  d = dim(MatrizDatos)
  MatrizDatos = cbind(rep(1,d[1]), MatrizDatos)

  X = MatrizDatos
  sv = svd(t(X)%*%X)
  U = sv$u
  D = diag(sv$d)
  V = sv$v

  XtraX_inv = V%*%solve(D)%*%t(V)
  Xpseudo = XtraX_inv%*%t(X)
  #Obtengo w:
  w = Xpseudo%*%Etiquetas
  return(w)
}
```

Como vamos a hacer una clasificación binaria, hacemos que las etiquetas con valor 5, pasen a tomar el valor -1. (Se podrá sacar de la propia función posteriormente pero ahora no nos hace falta)

A la matriz de datos que contiene las medias y las simetrías, hemos de añadirle una componente inicial con el valor 1, igual que hacíamos en PLA.

Hacemos la descomposición SVD de la matriz traspuesta de los datos por la matriz de los datos. Y ahora su inversa a partir de la descomposición. Esta será nuestra matriz XtraX_inv. Tal y como hemos visto en el algoritmo de clase, multiplicamos dicha matriz por la matriz traspuesta de los datos y finalmente por las etiquetas. Ya tenemos los coeficientes del hiperplano.

Hemos hecho uso del método lm(Etiquetas ~ MatrizDatos(sin los 1's iniciales)) para comprobar que, efectivamente, se hace la regresión correctamente.

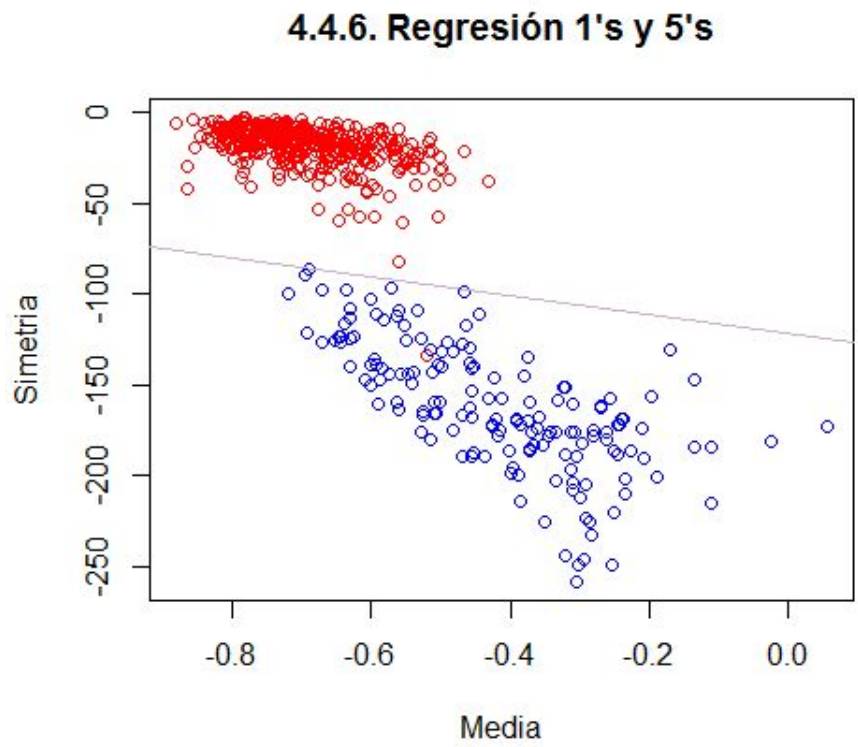
Resultados:
No hay resultados que mostrar en este apartado.

6. Ajustar un modelo de regresión lineal a los datos de (Intensidad promedio, Simetria) y pintar la solución junto con los datos. Valorar el resultado.

```
hiperplanow = regresion(matrizdatoss3e4, etiquetass3e4)

plot(matrizdatoss3e4[,1], matrizdatoss3e4[,2], main="4.4.6. Regresión 1's y 5's",
     col=(etiquetass3e4-1)/2 +2)

abline( a=-hiperplanow[1,]/hiperplanow[3,],
        b=-hiperplanow[2,]/hiperplanow[3,],
        col="thistle3")
```

Obtenemos los coeficientes del hiperplano que separa los datos de 1's y 5's. Representamos los datos y el hiperplano.

Resultados:



Vemos que el resultado es realmente bueno, ya que parece que sólamente hay un dato que se encuentra en la zona incorrecta del hiperplano.