Aprendizaje Automático (2015-2016)
Grado en Ingeniería Informática
Universidad de Granada

Práctica 1

Laura Tirado López

26 de marzo de 2016

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 $rango$.

Para generar los números aleatorios he utilizado la función sample, pasándole como parámetros los valores del rango y la dimensión.

```
simula_unif <- function(N,dim,rango){

  lista <- list()

  for(i in 1:N){
    v <- sample(rango[[1]]:rango[[2]],dim,rep=TRUE)
    lista[[i]] <- v
  }

  lista
}
```

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 $sigma$.

El esquema de la función es el mismo que el del apartado anterior, con la diferencia de que generamos los números aleatorios gaussianos utilizando la función rnorm.

```
simula_gauss <- function(N,dim,sigma){

  lista <- list()

  for(i in 1:N){
    v <- rnorm(dim,0,sigma)
    lista[[i]] <- v
  }

  lista
}
```

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

Para dibujar la gráfica utilizamos la función plot. Al pasarle como argumento la llamada a la función lo hacemos con unlist para que nos convierta la lista en un vector y de esa forma plot pueda dibujar la gráfica, dado que no acepta la lista directamente.

```
muestra1 <- simula_unif(50,2,c(-50,50))
plot(do.call(rbind,muestra1),main='Gr\áfica de lista generada con
    simula_unif', xlab='Eje X',ylab='Eje Y')
```

La gráfica resultante es la siguiente:

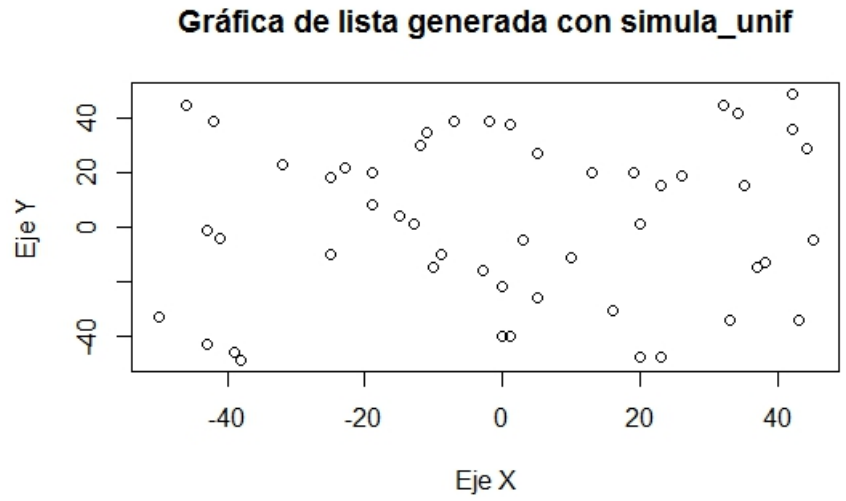

Figura 0.1: Gráfica de los datos generados por la función *simula_unif*

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

Para dibujar la gráfica lo hacemos igual que en el apartado anterior, pero llamando a la función *simula_gauss*

```
muestra2 <- simula_gauss(50,2,c(5,7))
plot(do.call(rbind,muestra2),main='Gr\áfica de lista generada con
    simula_gauss',xlab='Eje X',ylab='Eje Y',xlim=c(-50,50),ylim=c
    (-50,50))
```

La gráfica resultante es la siguiente:

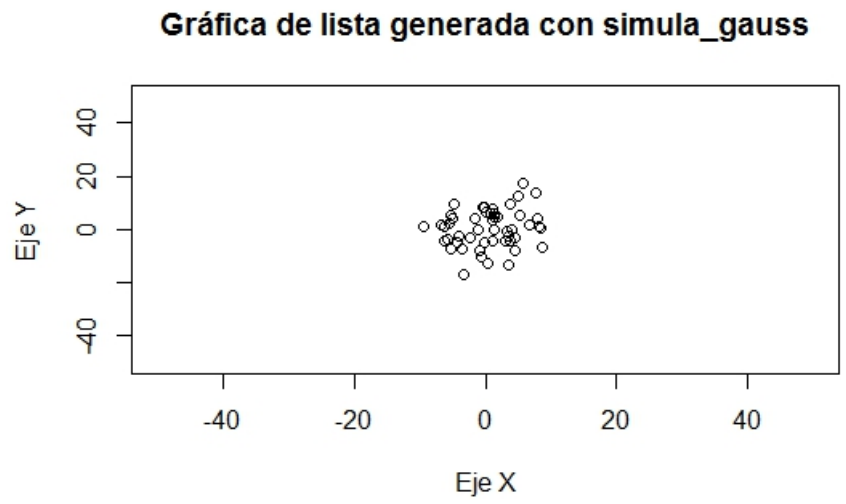

Figura 0.2: Gráfica de los datos generados por la función *simula_gauss*

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 $[-50, 50] \times [-50; 50]$. (Ayuda: Para calcular la recta simular las coordenadas de dos puntos dentro del cuadrado y calcular la recta que pasa por ellos).

Para calcular los valores a y b, primero generamos dos puntos aleatorios dentro del *intervalo*.
El parámetro a, es la pendiente de nuestra recta, el cual se puede calcular como $(y)/(x) = (y_2 - y_1)/(x_1 - x_2)$.
Una vez calculado a, para calcular b simplemente despejamos y se nos queda la expresión $b = y - ax$. Sustituimos los valores x e y por las coordenadas de unos de los puntos que generamos y obtenemos b.
Por último, mostramos el resultado obtenido.

```
simula_recta <- function(intervalo){

  punto1 <- sample(intervalo[[1]]:intervalo[[2]],2,rep=TRUE)
  punto2 <- sample(intervalo[[1]]:intervalo[[2]],2,rep=TRUE)

  a <- (punto2[[2]]-punto1[[2]])/(punto2[[1]]-punto1[[1]])
  b <- punto1[[2]] - a*punto1[[1]]

```

```
  v <- c(a,b)
  v
}
```

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.

Primero generamos la muestra 2D de puntos. Para trabajar más fácilmente con la muestra, la generamos como dos vectores de dimensión dim, de manera que el primer vector contendría las abcisas x de los puntos y el segundo las ordenadas y. Después, calculamos los parámetros a y b de la recta definida en el intervalo $rango$.
A continuación, clasificamos las variables según el signo utilizando la función $sign$, la cual recibe como argumento la expresión de la función que utilizamos para evaluar: $y - ax - b$. Los cálculos en R, se realizan de forma vectorial, es decir, para cada componente del vector. Por esta razón generamos la muestra como dos vectores, abcisas y ordenadas, para que al aplicarle la función se calculen los valores para cada coordenada.
Por último, dibujamos la gráfica de la función y las variables como puntos. Los puntos verdes se clasifican con la etiqueta 1 y los rojos con la etiqueta -1.

Como valores de los parámetros, defino el rango de valores entre $[-30, 30]$ y genero 200 variables.

```
rango <- c(-30,30)
variables <- simula_unif(2,100,rango)
v <- simula_recta(rango)

clasificadas <- sign(variables[[2]]-v[[1]]*variables[[1]]-v[[2]])

plot(seq(rango[[1]],rango[[2]]),seq(rango[[1]],rango[[2]])*v[[1]]+
    v[[2]],type='l',main='Funci\ón apartado 6',ylim=rango,xlab='Eje
     X',ylab='Eje Y')
points((variables[[1]])[clasificadas==1],(variables[[2]])[
    clasificadas==1],col='green')
points((variables[[1]])[clasificadas==-1],(variables[[2]])[
    clasificadas==-1],col='red')
```

La gráfica resultante es la siguiente:

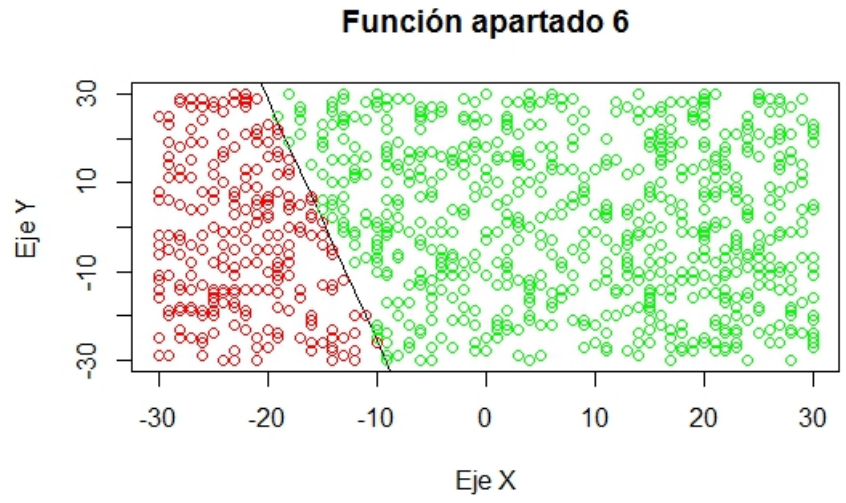

Figura 0.3: Gráfica de los datos clasificados del apartado 6

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

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

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?

Para etiquetar la muestra seguimos el mismo procedimiento que en el apartado anterior utilizando la función *sign*. Para hacerlo de una forma más sencilla, defino primero todas las funciones y luego evalúo el signo de cada función con los datos de la muestra.
Para mostrar las gráficas, utilizo además de la función *plot*, la función *par* para mostrar todas las gráficas juntas.

```
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
}

c1 <- sign(f1(unlist(variables[[1]]),unlist(variables[[2]])))
c2 <- sign(f2(unlist(variables[[1]]),unlist(variables[[2]])))
c3 <- sign(f3(unlist(variables[[1]]),unlist(variables[[2]])))
c4 <- sign(f4(unlist(variables[[1]]),unlist(variables[[2]])))

par(mfrow=c(2,2))
plot(variables[[1]],variables[[2]],main='Funci?n 1',xlab="Eje X",
    ylab="Eje Y",col='red')
points((variables[[1]])[c1==1],(variables[[2]])[c1==1],col='green'
    )
lines(rango[[1]]:rango[[2]],f1(rango[[1]]:rango[[2]],rango[[1]]:
    rango[[2]]))

plot(variables[[1]],variables[[2]],main='Funci?n 2',xlab="Eje X",
    ylab="Eje Y",col='red')
points((variables[[1]])[c2==1],(variables[[2]])[c2==1],col='green'
    )
lines(rango[[1]]:rango[[2]],f2(rango[[1]]:rango[[2]],rango[[1]]:
    rango[[2]]))

plot(variables[[1]],variables[[2]],main='Funci?n 3',xlab="Eje X",
    ylab="Eje Y",col='red')
points((variables[[1]])[c3==1],(variables[[2]])[c3==1],col='green'
    )
lines(rango[[1]]:rango[[2]],f3(rango[[1]]:rango[[2]],rango[[1]]:
    rango[[2]]))

plot(variables[[1]],variables[[2]],main='Funci?n 4',xlab="Eje X",
    ylab="Eje Y",col='red')
points((variables[[1]])[c4==1],(variables[[2]])[c4==1],col='green'
    )
lines(rango[[1]]:rango[[2]],f4(rango[[1]]:rango[[2]],rango[[1]]:
    rango[[2]]))
```

Las gráficas resultantes son las siguientes:

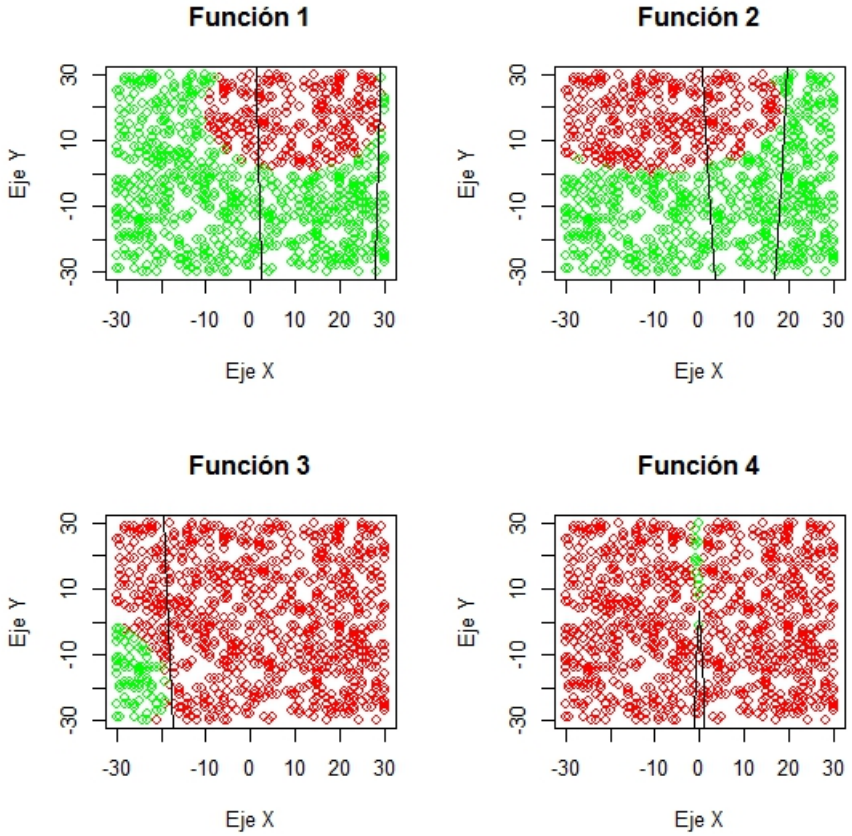

Figura 0.4: Gráfica de las clasificaciones de datos del apartado 7

A partir de las gráficas, podemos ver como los puntos de las regiones positiva o negativa dependiendo de la gráfica se agrupan de forma más o menos circular o de forma curva, debido a que las hemos etiquetado con funciones cuadráticas.

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.

- Visualice los puntos con las nuevas etiquetas y la recta del apartado.6.
- En una gráfica aparte visualice de nuevo los mismos puntos pero junto con las funciones del apartado.7.

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

Para modificar las etiquetas, he creado una función $modifica_etiquetas$ y dos funciones auxiliares para calcular el porcentaje de puntos que hay que cambiar de cada región. Las funciones auxiliares simplemente cuentan el número de etiquetas de cada tipo y calculan el 10 % de cada región. La función $modifica_etiqueta$, llama a las funciones auxiliares y modifica los valores de tantas etiquetas de cada región como indiquen los porcentajes. Llamamos a la función $modifica_etiqueta$ para cada función que necesitemos y pintamos de nuevo su gráfica.

```
porcentaje1 <- function(c){

  cont1 <- 0

  for(i in c){
    if(i==1){
      cont1 <- cont1+1
    }
  }

  round(cont1*0.1)
}

porcentaje_1 <- function(c){

  cont1 <- 0

  for(i in c){
    if(i==-1){
      cont1 <- cont1+1
    }
  }

  round(cont1*0.1)
}

modifica_etiquetas <- function(c){

  p1 <- porcentaje1(c)
  p <- porcentaje_1(c)

  j <- 0
  i <- 0
  k <- 1

  while(i < p1 || j < p){

```

```
38      if(c[[k]]==1){
39        c[[k]] <- -1
40        i <- i+1
41      }
42      else{
43        c[[k]] <- 1
44        j <- j+1
45      }
46
47      k <- k+1
48    }
49
50    c
51  }
52
53  copia_clasificadas <- modifica_etiquetas(clasificadas)
54  par(mfrow=c(1,1))
55  plot(seq(rango[[1]],rango[[2]]),seq(rango[[1]],rango[[2]])*v[[1]]+
        v[[2]],type='l',main='Funci?n apartado 6',ylim=rango,xlab="Eje
        X",ylab="Eje Y")
56  points((variables[[1]])[copia_clasificadas==1],(variables[[2]])[
        copia_clasificadas==1],col='green')
57  points((variables[[1]])[copia_clasificadas==-1],(variables[[2]])[
        copia_clasificadas==-1],col='red')
58
59
60  copia_c1 <- modifica_etiquetas(c1)
61  copia_c2 <- modifica_etiquetas(c2)
62  copia_c3 <- modifica_etiquetas(c3)
63  copia_c4 <- modifica_etiquetas(c4)
64
65  par(mfrow=c(2,2))
66  plot(variables[[1]],variables[[2]],main='Funci?n 1',xlab="Eje X",
        ylab="Eje Y",col='red')
67  points((variables[[1]])[copia_c1==1],(variables[[2]])[copia_c1
        ==1],col='green')
68  lines(rango[[1]]:rango[[2]],f1(rango[[1]]:rango[[2]],rango[[1]]:
        rango[[2]]))
69
70  plot(variables[[1]],variables[[2]],main='Funci?n 2',xlab="Eje X",
        ylab="Eje Y",col='red')
71  points((variables[[1]])[copia_c2==1],(variables[[2]])[copia_c2
        ==1],col='green')
72  lines(rango[[1]]:rango[[2]],f2(rango[[1]]:rango[[2]],rango[[1]]:
        rango[[2]]))
73
74  plot(variables[[1]],variables[[2]],main='Funci?n 3',xlab="Eje X",
        ylab="Eje Y",col='red')
75  points((variables[[1]])[copia_c3==1],(variables[[2]])[copia_c3
```

```
    ==1],col='green')
lines(rango[[1]]:rango[[2]],f3(rango[[1]]:rango[[2]],rango[[1]]:
    rango[[2]]))

plot(variables[[1]],variables[[2]],main='Funci?n 4',xlab="Eje X",
    ylab="Eje Y",col='red')
points((variables[[1]])[copia_c4==1],(variables[[2]])[copia_c4
    ==1],col='green')
lines(rango[[1]]:rango[[2]],f4(rango[[1]]:rango[[2]],rango[[1]]:
    rango[[2]]))
}
```

La gráfica del apartado 6 sería la siguiente:

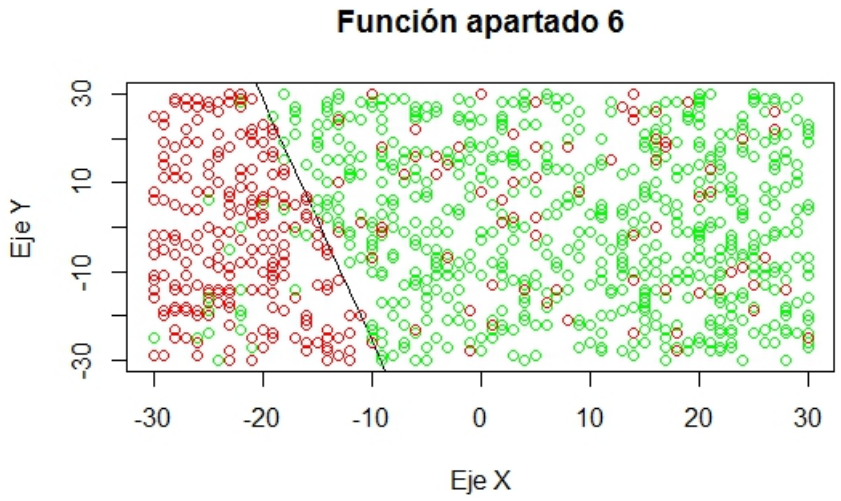

Figura 0.5: Gráfica de las clasificaciones de datos del apartado 6 tras modificar las etiquetas

Las gráficas del apartado 7 serían:

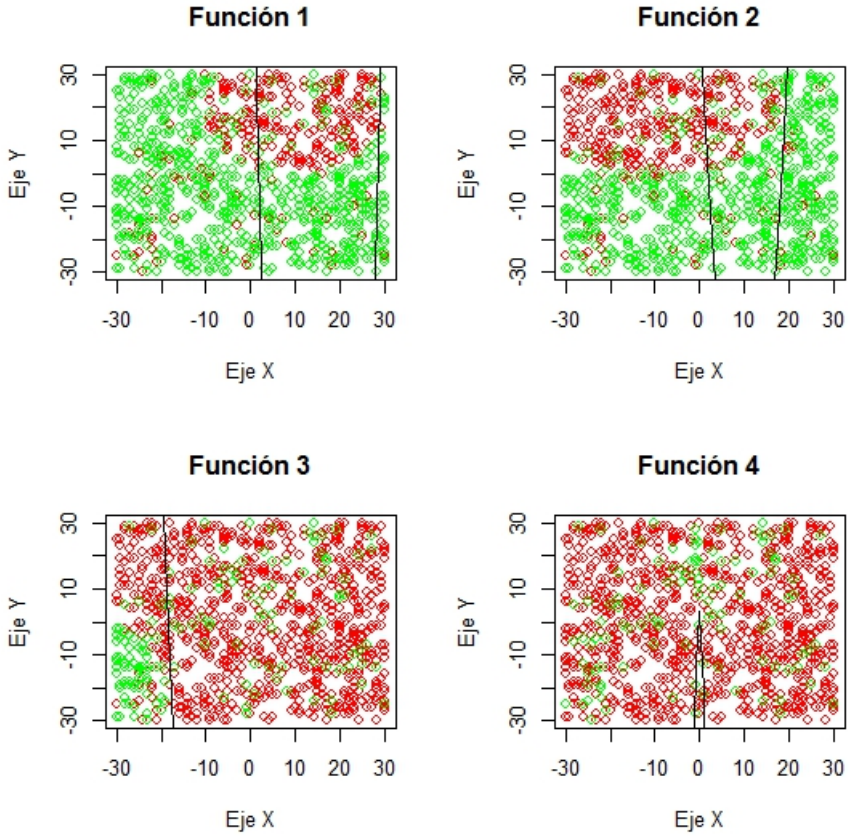

Figura 0.6: Gráfica de las clasificaciones de datos del apartado 7 tras modificar las etiquetas

Como podemos ver, al modificar los valores de las etiquetas, las funciones no clasifican de forma correcta los datos.

Ejercicio de Ajuste del Algoritmo Perceptron

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 ó -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.

El algoritmo PLA ajusta el vector de pesos mientras haya errores al comprobar las etiquetas, es decir, que haya algún punto mal clasificado o lleguemos al número maximo de iteraciones que le indicamos. Dentro de este primer bucle, para cada punto comprobamos si está bien clasificado. Si no lo está, ajustamos los pesos. Por último, calculamos la pendiente y el coeficiente de posición (el punto donde de intersección de la recta con el eje y). La ecuación general de la recta es $Ax + By + C = 0$, donde la pendiente se calcula como $m = -A/B$ y el coeficiente de posición como $n = -C/B$. Los valores de los pesos se corresponden directamente con A, B y C, por lo que $m = -w_1/w_2$ y $n = -w_3/w_2$.

```
ajusta_PLA <- function(datos,label,max_iter,vini){

  pesos <- matrix(vini,nrow=3,ncol=1)
  niter <- 0
  coef <- c(0,0)
  hay_error <- TRUE
  errores <- 0

  while(hay_error & niter <= max_iter){
    hay_error <- FALSE

    for(i in 1:nrow(datos)){

      punto <- c(datos[i,],1)

      if(sign(sum(t(pesos)%*%punto)) != label[i]){

        pesos <- pesos + label[i]*punto
        hay_error <- TRUE
        errores <- errores+1
      }
    }

    niter <- niter+1
  }

  coef[1] = -pesos[1]/pesos[2]
```

```
28    coef[2] = -pesos[3]/pesos[2]
29
30    return(c(coef[1],coef[2],niter,errores))
31  }
```

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

Tras ejecutar el algoritmo PLA, el número medio de iteraciones necesarias para converger es 312,8. Este valor por sí sólo realmente no nos aporta mucha información. Si nos fijamos además en el número de iteraciones de cada simulación, los cuales son 801, 59, 128, 128, 17, 801, 131, 801 y 131 respectivamente, podemos deducir que el valor inicial de los pesos influye en el número de iteraciones que tarda el algoritmo PLA en converger hacia la solución.

En la siguiente imagen podemos ver el resultado gráfico del algoritmo PLA:

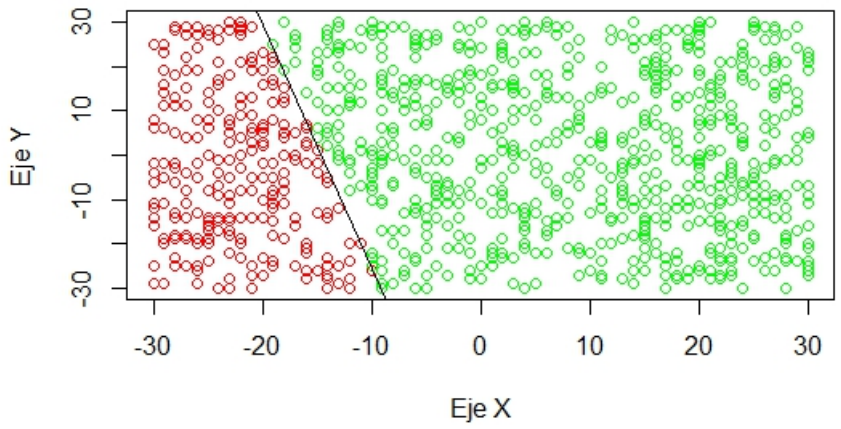

Figura 0.7: Gráfica del algoritmo PLA

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.

Para 10 iteraciones, el número de errores ha sido 1826. El resultado se ve en la siguiente imagen:

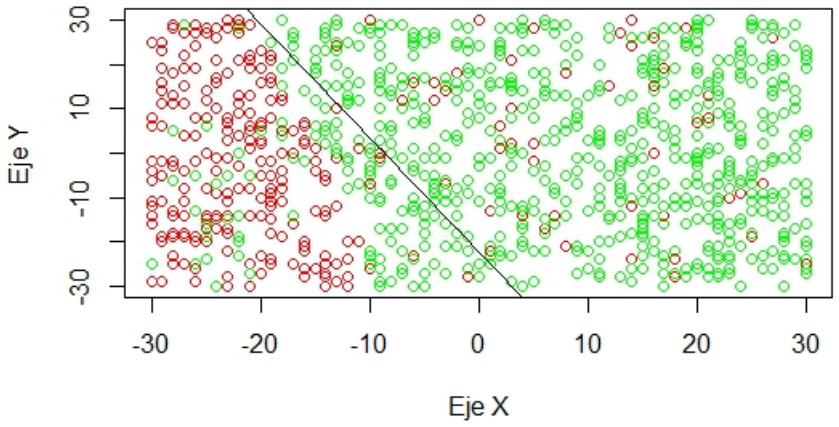

Figura 0.8: Gráfica del algoritmo PLA con 10 iteraciones

Para 100 iteraciones, el número de errores ha sido 14731. El resultado se ve en la siguiente imagen:

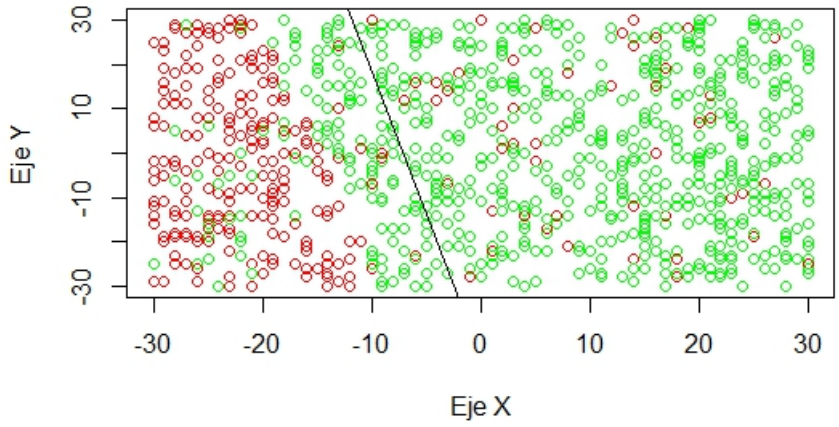

Figura 0.9: Gráfica del algoritmo PLA con 100 iteraciones

Para 1000 iteraciones, el número de errores ha sido 143582. El resultado se ve en la siguiente imagen:

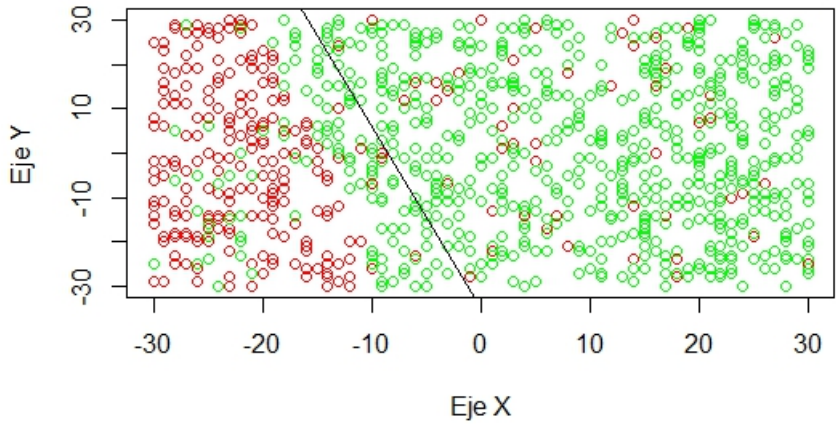

Figura 0.10: Gráfica del algoritmo PLA con 1000 iteraciones

Las tres simulaciones clasifican los datos de manera similar. Ninguna los clasifica completamente dado que los datos no son linealmente separables. En mi opinión, la tercera

simulación con 1000 iteraciones es la que mejor clasifica los datos, aunque la segunda da un resultado similar.

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

Para 10 iteraciones, el número de errores ha sido 2033. El resultado se ve en la siguiente imagen:

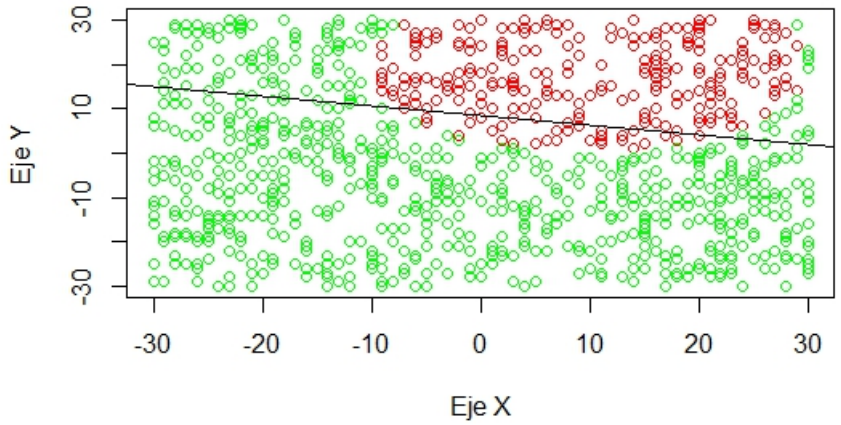

Figura 0.11: Gráfica del algoritmo PLA con 10 iteraciones

Para 100 iteraciones, el número de errores ha sido 15993. El resultado se ve en la siguiente imagen:

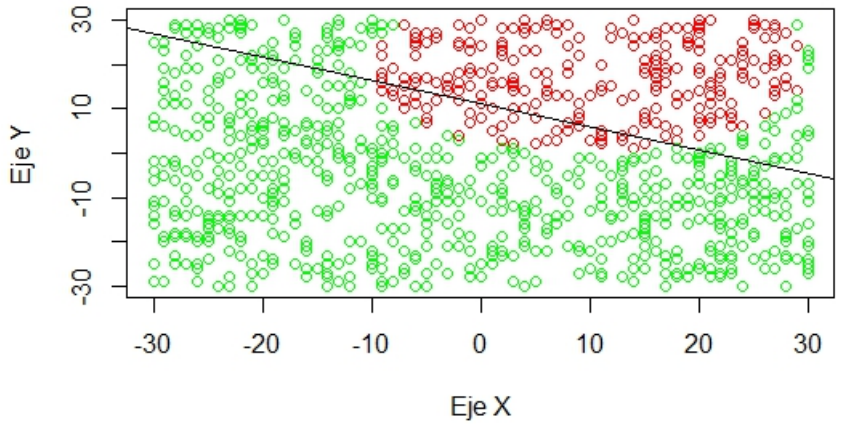

Figura 0.12: Gráfica del algoritmo PLA con 100 iteraciones

Para 1000 iteraciones, el número de errores ha sido 153693. El resultado se ve en la siguiente imagen:

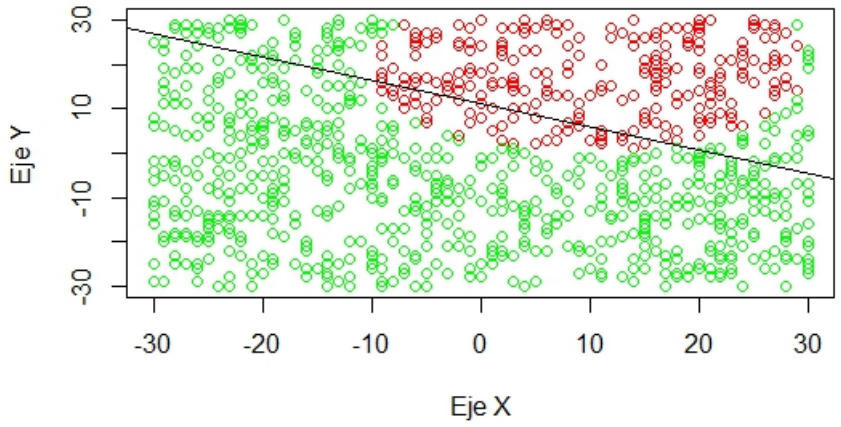

Figura 0.13: Gráfica del algoritmo PLA con 1000 iteraciones

En este caso, la simulación de 100 iteraciones y la de 1000 iteraciones convergen en la misma solución, la cual es bastante aceptable teniendo en cuenta que los datos han sido

clasificados con una función cuadrática e intentamos separarlos con una función lineal. Aún así, la simulación con 10 iteraciones también da un resultado parecido aunque peor que el de las otras dos simulaciones.

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

La modificación del algoritmo consiste en añadir dentro del bucle while las líneas necesarias para pintar los puntos y las gráficas:

```
ajusta_PLA2 <- function(datos,label,max_iter,vini){

  pesos <- matrix(vini,nrow=3,ncol=1)
  niter <- 0
  coef <- c(0,0)
  hay_error <- TRUE
  errores <- 0

  while(hay_error & niter <= max_iter){
    hay_error <- FALSE

    for(i in 1:nrow(datos)){

      punto <- c(datos[i,],1)

      \#Actualizamos los pesos
      if(sign(sum(t(pesos)%*%punto)) != label[i]){

        pesos <- pesos + label[i]*punto
        hay_error <- TRUE
        errores <- errores+1
      }

    }

    if(pesos[2]==0){
      coef[1] = 0
      coef[2] = 0
    }
    else{
      coef[1] = -pesos[1]/pesos[2]
      coef[2] = -pesos[3]/pesos[2]
    }

    plot(datos,xlabel="Eje X",ylabel="Eje Y",col='red')
    points((datos[,1])[label==1],(datos[,2])[label==1], col='green
```

```
        ')
37      abline(coef[2],coef[1])
38      niter <- niter+1
39    }
40
41    return(c(coef[1],coef[2],niter,errores))
42  }
```

Los resultados son los mismos que los del apartado 3.

Ejercicio sobre Regresión Lineal

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.

Para visualizar las imágenes primero leemos el fichero como una tabla con la función *read.table*. A continuación las dibujamos con *levelplot*.

```
1  digits <- read.table("zip.train",header = FALSE)
2
3  levelplot((digit.data[,1:9]) ~ rep(rep(1:16, 9), 16) + rep(rep
       (1:16, each = 16), 9) | gl(9, 256), col.regions = rev(gray.
       colors(256)),at = c(-1, 0, 1), labels = FALSE, ylim = c(17, 0))
```

Las imágenes obtenidas se muestran en la siguiente figura:

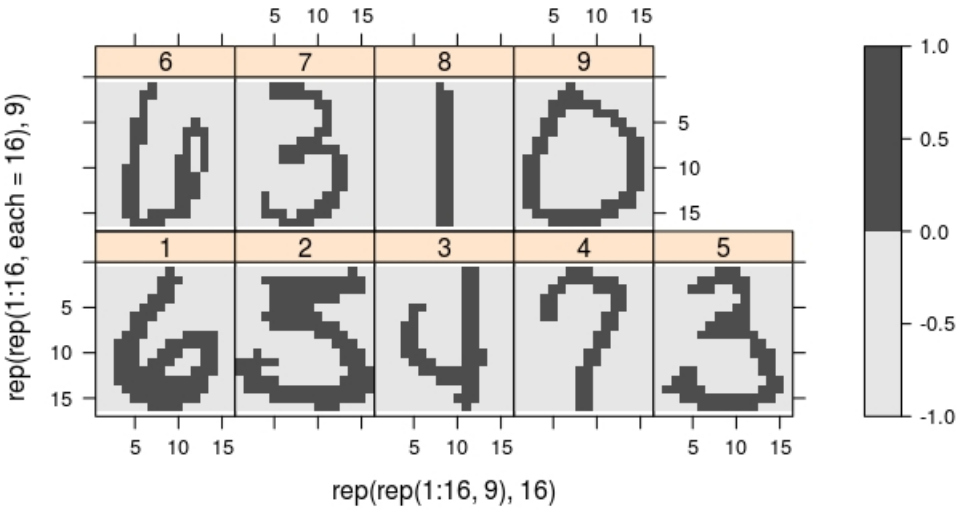

Figura 0.14: Lectura de las imágenes del archivo zipdigits