Aprendizaje Automático (2015-16)
Grado en Ingeniería Informática
Universidad de Granada

INFORME TRABAJO DE PRÁCTICAS 1

Miguel López Campos

27 de marzo de 2016

Índice

Índice de figuras

1. Ejercicio 5.2: Generación y Visualización de datos

1.1. Apartados 1, 2, 3 y 4

Para crear las funciones simula_unif y simula_gauss he usado sapply. En sapply introduzco como argumentos un array que indica el número de veces que quiero que se ejecute la función deseada (runif o rnorm en nuestro caso), así como los argumentos de la función deseada. sapply además intenta simplificar lo máximo posible la estructura de datos en la que se devuelve el resultado.

Tras las ejecuciones de simula_unif y simula_gauss obtenemos las siguientes gráficas.

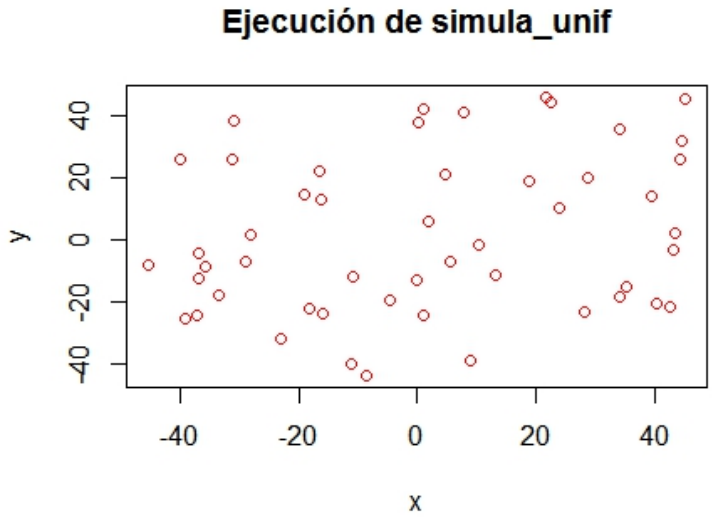

Figura 1.1: Ejecución de simula unif

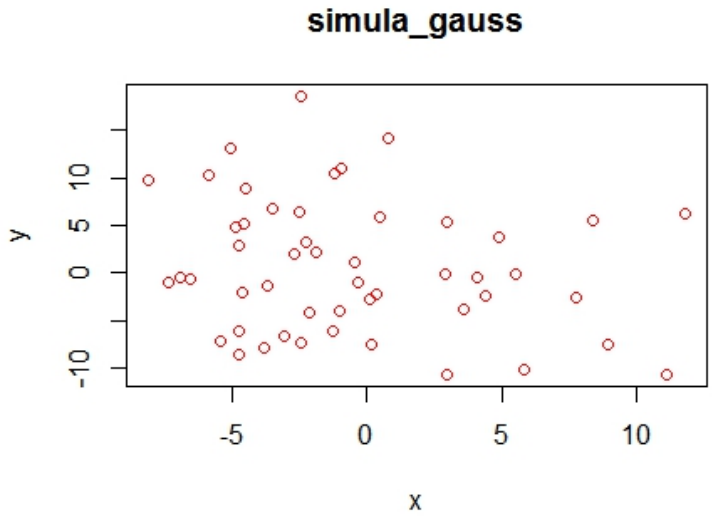

Figura 1.2: Ejecución de simula Gauss

1.2. Apartado 5: Construir recta aleatoria

Para crear la recta hay que generar dos puntos de forma aleatoria. La recta resultante será la que pasa por estos dos puntos. Teniendo en cuenta que la ecuación de una recta es y = ax+b, podemos resolver un sistema de ecuaciones cuyas incógnitas sean 'a' y 'b'. En la función simula_recta resuelvo este sistema de ecuaciones y devuelvo una lista con los valores aleatorios de 'a' y 'b'.

```
simula_recta <- function()
{
        intervalo <- c(-50,50)
        p <- simula_unif(N=2, dim=2, c(-50,50))
        a <- (p[2,2]-p[2,1])/(p[1,2]-p[1,1])
        b <- p[2,1]-p[1,1]*a
        return(list(a,b))
}
```

1.3. Apartado 6: Etiquetado de muestra aleatoria

Para realizar este apartado primero uso la función simula_unif para generar 1000 puntos de 2D. Al devolverme una matriz, convierto ésta a un data frame para tener una representación de la estructura de datos más clara, poniéndole de nombre a las columnas 'x' e 'y'. Simulo la recta y etiqueto siguiendo la función f(x, y) = y - ax - b. En la gráfica se puede ver el resultado. Los puntos verdes son +1 y los rojos -1.

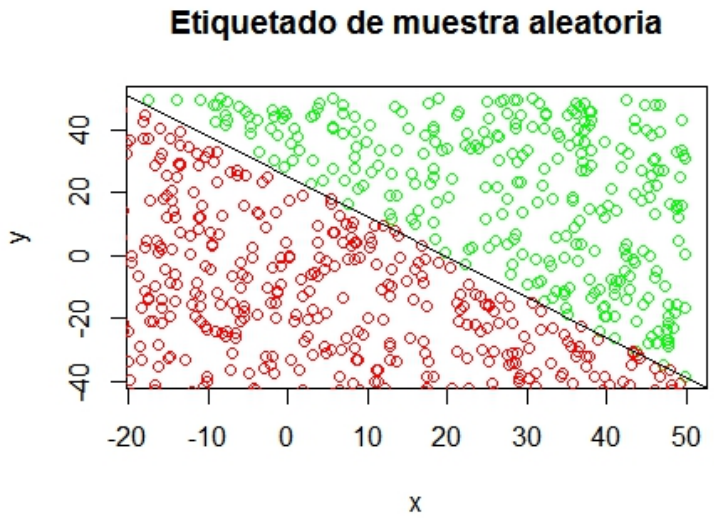

Figura 1.3: Etiquetado de muestra aleatoria con recta aleatoria

1.4. Apartado 7: Etiquetado para funciones no lineales

Las funciones son las siguientes (verde +1, rojo -1)

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

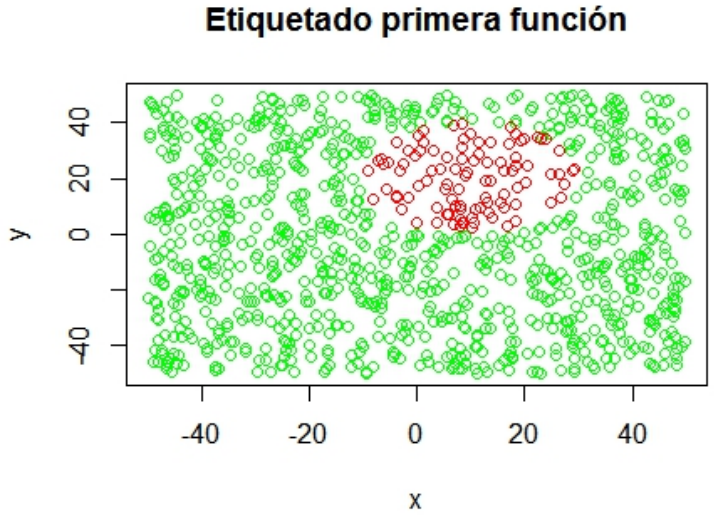

Figura 1.4: Apartado 7: primera funcion

- f(x, y) = 0.5(x + 10)^2 + $(y - 20)^2 - 400$

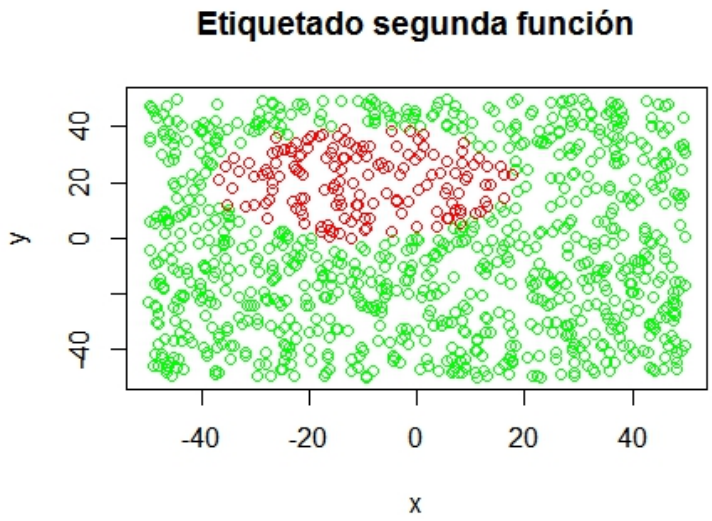

Figura 1.5: Apartado 7: segunda funcion

- f(x, y) = 0.5(x - 10)^2 $- (y + 20)^2 - 400$

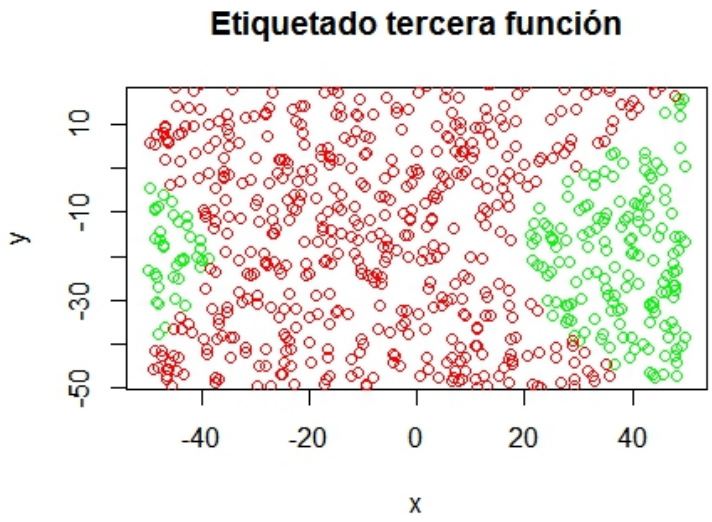

Figura 1.6: Apartado 7: tercera funcion

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

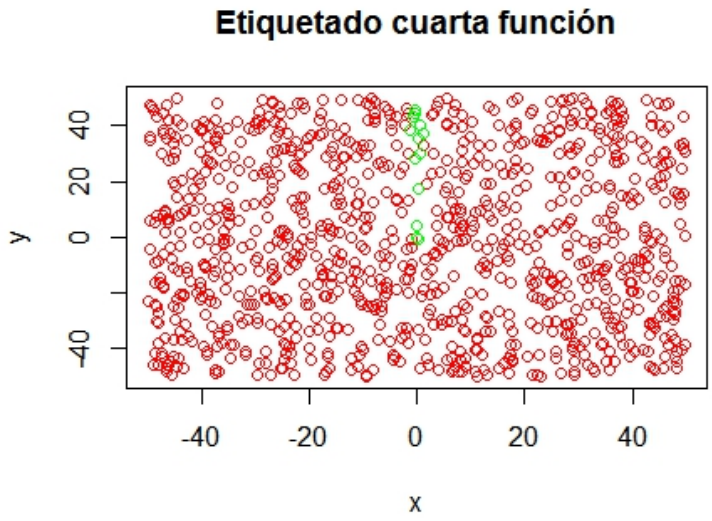

Figura 1.7: Apartado 7: cuarta funcion

Sobre estas funciones se puede deducir que no son lineales, es decir, no se pueden dividir los puntos etiquetados mediante una recta.

1.5. Apartado 8: Ruido en la muestra

Para cambiar el 10 % de los puntos positivos por negativos y viceversa he creado la siguiente función en la que lo que hago es extraer un 10 % de cada uno de los etiquetados mediante la función sample. Entonces lo que hago es tener un array de probabilidad que es probs y asigno probabilidad 0 a los negativos y 1 a los positivos (para cambiar negativos, a la inversa). De entrada a sample también le introduzco un vector (que son los índices del array de etiquetas) y la longitud de la muestra aleatoria de índices que quiero sacar (en mi caso el 10 %). Para esos índices obtenidos de manera aleatoria, cambio su etiqueta:

```
change_muestra <- function(x)
{

        n_positivos <- as.integer(length(x[x>0])/10)
        probs <- rep(1, length(x))
        probs[x<0] <- 0
        #Con sample obtengo aleatoriamente un vector de
            ndices  aleatorios correspondientes
        #con el 10 % de las etiquetas positivas. Estos
            ndices  los usar  para cambiar la etiqueta
        ind_aleatorios <- sample(x=(1:length(x)), prob=probs,
            size=n_positivos)
        new_muestra <-x
        new_muestra[ind_aleatorios] <- -1

        n_negativos <- as.integer(length(x[x<0])/10)
        probs <- rep(1, length(x))
        probs[x>0] <- 0
        ind_aleatorios <- sample(x=(1:length(x)), prob=probs,
            size=n_negativos)
        new_muestra[ind_aleatorios] <- 1

        return(new_muestra)
}
```

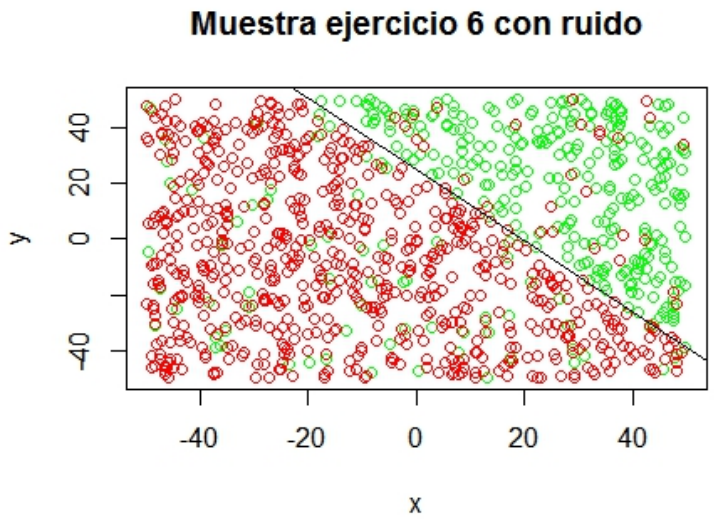

Figura 1.8: Hemos aplicado ruido

Como consecuencia de esta modificación, estamos introduciendo ruido a la muestra. Esto supone una pérdida de lineabilidad en los etiquetados y por consecuente una dificultad a la hora de realizar el aprendizaje.

2. Ejercicio 5.3: Ajuste del Algoritmo Perceptron

2.1. Apartado 1: Implementación

En mi implementación introduzco el vector de pesos inicial de manera separada donde el término de w independiente lo introduzco también independientemente. Esto lo hago para no tener que añadir siempre una columna de 1's a la matriz que introduzco de entrada. Dentro de la función ajusta_PLA tenemos un bucle while externo. Éste se ejecutará mientras haya cambios en el vector de pesos y mientras que no se supere el umbral dado como argumento. Dentro de este while tenemos un for que recorrerá todos los elementos de la muestra y comprobará si con ese vector de pesos está bien ajustado. Si no lo está, cambia el vector de pesos. Una vez acaba la función devolvemos el número de iteraciones necesarias para que converga a una solución y el vector de pesos.

```
ajusta_PLA <- function(datos, label, max_iter, vini, b)
{
        change <- 1
        w <- vini
        iter <- 0
        while(change == 1 && iter < max_iter)
        {

                change <- 0

                for(i in seq_along(label))
                {
                        if(sign(sum(datos[i,]*w)+b)!=sign(
                           label[i]))
                        {
                        change <- 1
                        w <- w + datos[i,]*label[i]
                        b <- b + label[i]
                        }
                }

                iter <- iter+1
        }

        length(w) <- length(w)+1
        w[length(w)] <- b
        cat("Se han necesitado ")
        cat(iter)
        cat(" iteraciones")
```

```
29
30                      return(list(iter,w))
31              }
```

2.2. Apartado 2: Ejecución PLA

Para hacer el número de iteraciones medio para que converja, hago un bucle for que se repita 10 veces y en él genero de forma aleatoria el vector de pesos inicial. En la figura hay un ejemplo de un número medio de iteraciones. Esto es un claro indicativo de que el vector inicial de pesos es importante a la hora de eficiencia y, cuando los datos no sean linealmente separables, lo será también para el resultado. Adjunto también una gráfica del resultado de una de las estimaciones.

```
99.2
> cat(" iteraciones\n")
 iteraciones
```

Figura 2.1: Número medio de iteraciones

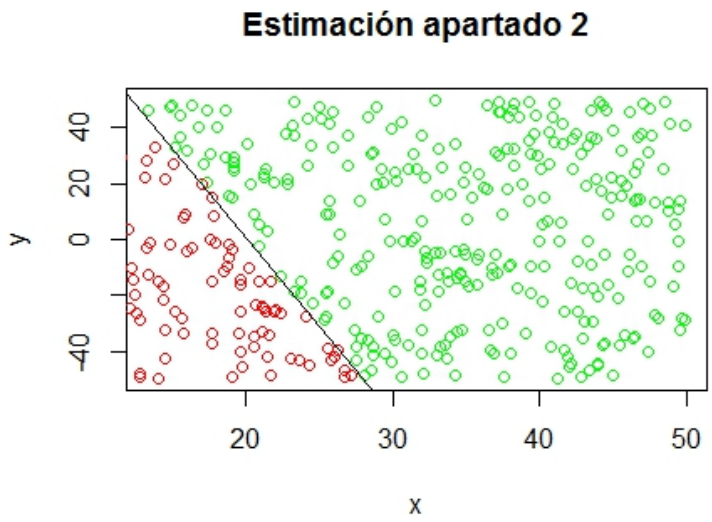

Figura 2.2: Estimación ejecutando PLA. Como vemos es muy buena.

2.3. Apartado 3: Ejecución PLA datos con ruido

Para realizar este apartado lo que hago es inicializar en cada una de las pruebas el vector de pesos inicial a 0. He obtenido, para un número máximo de iteraciones N=10, 187 errores; para N=100, 197 errores, y para N=1000, 215 errores. Esto nos indica que el perceptrón, cuando no hay una clara separabilidad lineal entre los datos (en este caso debido al ruido), aunque haga más iteraciones no tiene por qué dar una solución mejor.

2.4. Apartado 4: Ejecución PLA datos no linealmente separables

En este ejercicio realizo el mismo procedimiento que en el apartado anterior pero con la muestra del apartado 7 (primera función) del anterior ejercicio. En este caso obtengo para N=10, 418 errores. Para N=100, 112 errores, y para N=1000 340 errores. Como vemos, tenemos un porcentaje de error muy alto, incluso más que en el apartado anterior. Como hemos visto antes, la función del ejercicio 7 era no linealmente separable para las muestras. Por esto tenemos un porcentaje de error tan alto dentro de la muestra.

2.5. Apartado 5: Modificación de PLA

Para este apartado he realizado la siguiente versión modificada de PLA:

```
ajusta_PLA_modificado <- function(datos, label, max_iter,
    vini, b)
{
        change <- 1
        w <- vini
        iter <- 0
        while(change == 1 && iter < max_iter)
        {

                change <- 0

                for(i in seq_along(label))
                {


                        if(sign(sum(datos[i,]*w)+b)!=sign(
                            label[i]))
                        {
                        change <- 1
                        w <- w + datos[i,]*label[i]
                        b <- b + label[i]
                        }
                }
                positivos <- label>0
                plot(datos[positivos,], col="red")
                points(datos[!positivos,], col="green")
                abline(a=(-b/w[2]), b=(-w[1]/w[2]))
                Sys.sleep(0.1)

                iter <- iter+1
        }

        length(w) <- length(w)+1
        w[length(w)] <- b
        cat("Se han necesitado ")
        cat(iter)
        cat(" iteraciones")

        return(w)
}
```

Lo que hago, es después de haber recorrido toda la muestra en una iteración, muestro una gráfica representando los puntos (verdes +1 y rojos -1) así como la recta obtenida a partir de los pesos que se han calculado en esa iteración. También pongo un sys.sleep para que nos de tiempo a visualizar la gráfica.

En las dos siguientes figuras se ve la gráfica después de la primera iteración y la gráfica después de la segunda.

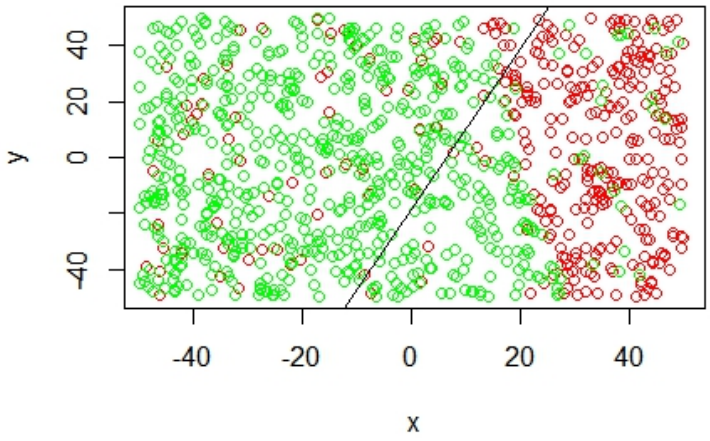

Figura 2.3: Iteración 1

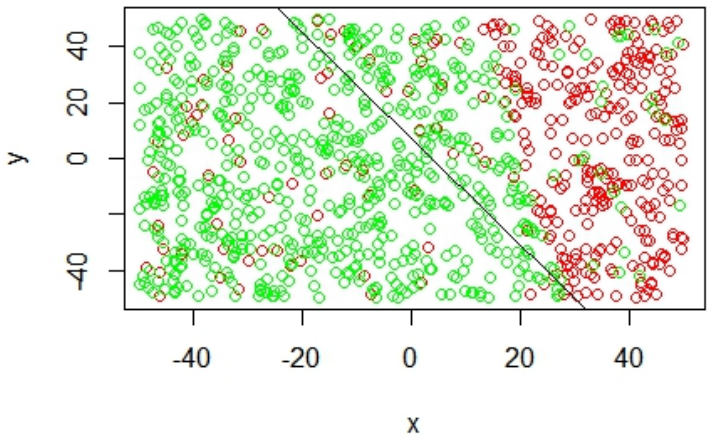

Figura 2.4: Iteración 2

```
> cat("El error es del ")
El error es del
> cat(Error*100)
11.2
> cat("%\n")
%
```

Figura 2.5: Error usando Pocket PLA sobre primera función del ejercicio 7

2.6. Apartado 6: Mejora para funciones no lineales

Para mejorar el PLA para funciones no lineales, he implementado el pocket PLA, que es el siguiente:

```
ajusta_PLA_MOD <- function(datos, label, t, max_iter_PLA)
{
        vini <- c(0,0)
        b <- 0
        w <- c(vini,b)
        error_mejor <- 1

        for(i in 1:t)
        {
                data_tmp <- as.matrix(datos)
                w_temp <- ajusta_PLA(data_tmp,label,
                   max_iter_PLA, w[1:2], w[3])[[2]]
                etiq_nuev <- datos$x*w_temp[1] + datos$y*
                   w_temp[2] + w_temp[3]
                etiq_nuev[etiq_nuev>0] <- 1
                etiq_nuev[etiq_nuev<0] <- -1
                errores <- label!=etiq_nuev
                errores <- label[errores]
                error_nuevo <- length(errores)/length(label)

                if(error_nuevo < error_mejor)
                {
                        error_mejor <- error_nuevo
                        w <- w_temp
                }
        }

return(w)
}
```

La función lo que hace es, inicializando el vector de pesos a 0, calcular reiteradamente pesos con PLA. Si los pesos que calcula son mejores que los anteriormente calculados, entonces actualizamos y en la siguiente iteración el vector inicial de pesos para PLA serán los mejores pesos. Para la primera función del ejercicio 7 obtenemos el siguiente resultado. Como vemos, el error es del 11 %, que comparado con el obtenido con PLA (41.8 %) es muchísimo mejor.

3. Ejercicio 5.4: Regresión lineal

3.1. Apartados 1 y 2: Leer datos y visualizarlos

Para leer el fichero de datos usamos la función read.table. Asigno a una variable este conjunto de datos. Después creo las matrices de 16x16 y las pongo todas en una lista. Las etiquetas las meto en una lista a parte.
Para visualizar la imagen, usamos la función image, introduciéndole como argumento la matriz deseada. En la figura se puede observar uno de los datos pintados.

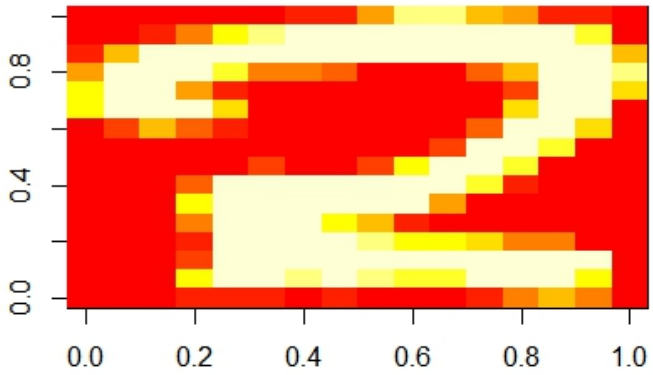

Figura 3.1: Visualizamos uno de los datos del conjunto de datos

Como vemos los datos son dígitos pintados a mano. Para quedarnos solo con los 1's y los 5's, lo que hago es que desde la lista de etiquetas me quedo sólo con éstos con el siguiente trozo de código:

```
accept1 <- labels_digits == 5
accept2 <- labels_digits == 1
accept <- accept1+accept2
m <- m[as.logical(accept)]
labels_digits <- labels_digits[as.logical(accept)]
```

3.2. Apartados 3 y 4: Simetría e intensidad de los datos y representación

Para calcular la intensidad media de los datos uso sapply de la siguiente manera:

```
        data_means <- sapply(m,mean)
```

Ésto lo que hace es devolver la media de cada una de las matrices (los datos) que están en la lista 'm'.

Para calcular la simetría empleo la siguiente función

```
        simmetry_function <- function(m)
        {
                sum <- 0
                for(i in 1:nrow(m))
                {
                        for(j in 1:ncol(m))
                        {
                                sum <- sum + abs(m[i,j]-m[i,ncol(m)-j
                                    +1])
                        }
                }

                sum <- -sum
                return(sum)
        }
```

En la función lo que hago es recorrer todas las filas de la matriz y para cada una de las columnas hago la sumatoria del valor absoluto de la diferencia de cada uno de los pixeles con el pixel que correspondería al mismo con la imagen invertida. Para aplicarla sobre todas las imágenes (cada uno de los datos de la muestra) empleo sapply.

En la siguiente gráfica muestro los 5's de color verde y los 1's de color rojo.

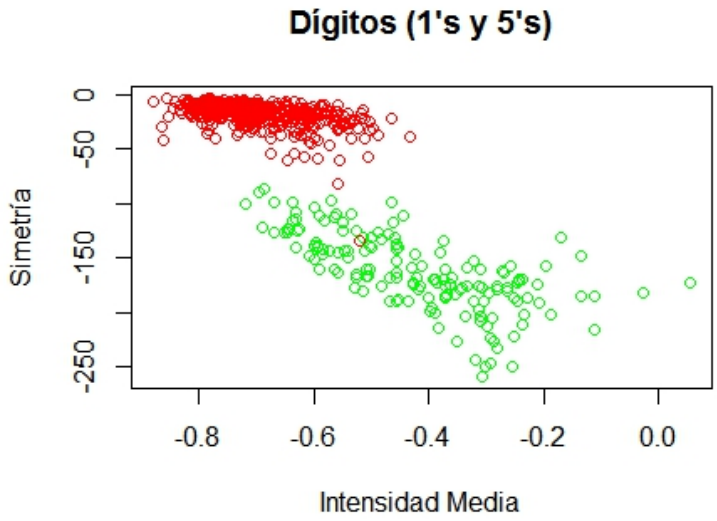

Figura 3.2: Gráfica de los 1's y los 5's según simetría e intensidad media

3.3. Apartados 5 y 6: Ajuste de regresión lineal

Para ajustar la regresión lineal he diseñado la siguiente función:

```
Regress_lin <- function(datos, label)
{
        descomposicion <- svd(datos)
        #La funci n diag construye una matriz diagonal a
           partir de un array
        D <- descomposicion$d
        for(i in 1:length(D))
        {
                if(D[i] > 10^-4)
                {
                D[i] <- 1/D[i]
                }else{D[i] <- 0}
        }
        D <- diag(D)
        V <- descomposicion$v
        U <- descomposicion$u
        #
        D <- t(D)
        pseudo_inverse <- V %*% D %*% t(U)
        w <- as.vector(pseudo_inverse %*% label)
}
```

Primero hago la descomposición en valores singulares con svd. Para hacer la pseudoinversa de la matriz D, tengo que hacer el inverso de cada uno de sus elementos y después hacer la traspuesta. Pongo un umbral a partir del cual si es más pequeño, directamente pongo 0. Formo una matriz diagonal mediante la función diag. A continuación calculo la pseudo inversa de X y por último la multiplico por la matriz label que contendrá las etiquetas en una matriz de una sola columna. Devuelvo los pesos en un vector.
En la siguiente figura muestro el resultado de aplicar regresión lineal sobre los datos de 1's y 5's con el código siguiente:

```
datos <- data.frame(data_means, data_simmetry, rep(1,length(
    data_means)))
datos <- as.matrix(datos)
etiquetas <- matrix(labels_digits, ncol = 1)
#Las etiquetas clasificar  los 5 como -1 y los 1's como +1
cincos <- etiquetas==5
etiquetas[cincos] = -1
pesos <- Regress_lin(datos, etiquetas)
```

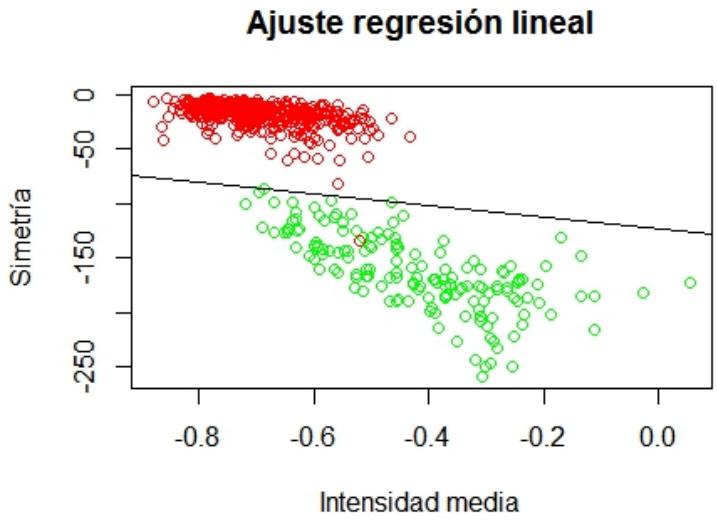

Figura 3.3: Regresión lineal sobre los datos de dígitos

3.4. Apartado 7

Para el apartado 'a' utilizo para etiquetar la función f(x,y) = y + ax +b , siendo 'a' y 'b' los coeficientes de una recta aleatoria. Aplico regresión lineal y calculo el error viendo el número de diferencias en relación al tamaño total de la muestra:

```
            errores <- as.vector(label_7a)!=etiquetados
            errores <- label_7a[errores]
            Error[i] <- length(errores)/100
```

Obtengo un error muy bajo, del 1.514

```
> cat(Error_medio*100)
1.514
> cat("%")
%
> cat("\n")
```

Figura 3.4: Error medio apartado a

Para calcular el error fuera de la muestra, genero en cada iteración una muestra de 1000 elementos. La etiqueto con la función 'f' y después la etiqueto con los pesos obtenidos mediante la regresión lineal con la muestra de 100 elementos. Calculo el número de errores y lo pongo en relación a los 1000 elementos de la muestra. Al finalizar el experimento obtengo un error medio del 1.77 %. Como vemos tras varias ejecuciones, casi siempre el error fuera de la muestra es mayor que el de dentro, algo que es lógico. Además, al ser los datos linealmente separables, obtenemos un error muy bajo, por lo que el resultado de aplicar regresión lineal en estos casos es bueno.

```
> cat(Error_medio*100)
1.7725
> cat("%")
%
> cat("\n")
```

Figura 3.5: Error medio apartado b

En el apartado c genero una muestra de tamaño N=10 y ajusto por regresión lineal. Los pesos obtenidos serán el vector de pesos inicial para el perceptrón. Como vemos se necesita de media muy pocas iteraciones para que converga (ver en la figura siguiente). Esto es debido a que al aplicar la regresión lineal estamos optimizando el error de dentro de la muestra, por lo que para el perceptrón será más fácil encontrar la solución.

```
El número medio de iteraciones es de
> cat(iters/1000)
2.868
> cat("\n")
```

Figura 3.6: Iteraciones medias perceptrón habiendo ajustado con regresión lineal antes

3.5. Apartado 8: Transformaciones no lineales

Primero generamos la muestra de N=1000 de forma aleatoria y aplicamos ruido (usando sample como en un apartado anterior). He empleado el siguiente código para realizar el experimento:

```
        Error <- vector()
        for(i in 1:1000)
        {
                muestra_8 <- simula_unif(N=1000, dim=2, c(-10,10))
                muestra_8 <- t(muestra_8)
                muestra_8 <- as.data.frame(muestra_8)
                names(muestra_8) <- c("x1","x2")
                label_8 <- muestra_8$x1*muestra_8$x1+muestra_8$x2*
                   muestra_8$x2-25.0
                label_8[label_8>0] <- 1
                label_8[label_8<0] <- -1
                #Extraigo un 10% aleatorio de la muestra con la
                   funci n sample
                probs <- rep(1,length(label_8))
                ind_aleatorios <- sample(x=(1:length(label_8)), prob=
                   probs, size=length(label_8)/10)
                #Le cambio el signo a la muestra aleatoria elegida
                label_8[ind_aleatorios] <- label_8[ind_aleatorios
                   ]*(-1)
                #Aplico el ajuste con la regresi n lineal
                muestra_8_reg <- data.frame(muestra_8, rep(1, length(
                   label_8)))
                muestra_8_reg <- as.matrix(muestra_8_reg)
                label_8_reg <- matrix(label_8, ncol=1)
                pesos_8 <- Regress_lin(muestra_8_reg, label_8_reg)
                #Eval o con los pesos calclados con la regresi n
                   lineal
                label_test_8 <- pesos_8[2]*muestra_8$x2+pesos_8[1]*
                   muestra_8$x1+pesos_8[3]
                label_test_8[label_test_8>0] <- 1
                label_test_8[label_test_8<0] <- -1
                #Contamos los errores
                errores <- label_test_8!=as.vector(label_8)
                errores <- label_test_8[errores]
        Error[i] <- length(errores)/1000
        }

        Error_medio <- mean(Error)
```

Como vemos primero creo la muestra y la etiqueto. A esas etiquetas les aplico ruido y posteriormente aplico regresión lineal. Al comprobar el error medio nos aparece un error muy alto (sobre el 25 %).

```
> cat(Error_medio*100)
25.7631
> cat("%")
%
```

Figura 3.7: Error ejercicio 8 (1)

El error es así de alto debido a que, además de que la función es no lineal, estamos también aplicando ruido a las etiquetas, por lo que se dificulta más aún el proceso de aprendizaje.

A continuación procedemos a linealizar el problema. Para ello tomamos como vector de características (1,x1,x2,x1x2,x1^2, $x2^2$).

```
1          muestra_8 <- simula_unif(N=1000, dim=2, c(-10,10))
2          muestra_8 <- t(muestra_8)
3          muestra_8 <- as.data.frame(muestra_8)
4          names(muestra_8) <- c("x1","x2")
5          #creo la nueva muestra con las variables del enunciado
6          muestra_8_b <- data.frame(muestra_8$x2^2, muestra_8$x1^2,
               muestra_8$x1*muestra_8$x2, muestra_8$x2, muestra_8$x1, rep
               (1, nrow(muestra_8)))
7          label_8_b <- (muestra_8$x1^2)+(muestra_8$x2^2)-25.0
8          label_8_b[label_8_b>0] <- 1
9          label_8_b[label_8_b<0] <- -1
10         probs <- rep(1,length(label_8_b))
11         ind_aleatorios <- sample(x=(1:length(label_8_b)), prob=probs,
               size=length(label_8_b)/10)
12         #Le cambio el signo a la muestra aleatoria elegida
13         label_8_b[ind_aleatorios] <- label_8_b[ind_aleatorios]*(-1)
14         label_8_b <- matrix(label_8_b, ncol=1)
15         #Ajusto la regresi n lineal
16         pesos <- Regress_lin(datos=muestra_8_b, label=label_8_b)
17         label_test_8b <- evaluacion(x=muestra_8_b, w=pesos)
18         errores <- label_test_8b!=as.vector(label_8_b)
19         errores <- label_test_8b[errores]
20         Error <- length(errores)/1000
21
22         cat("Los pesos obtenidos son: ")
23         cat(pesos)
24         cat("\n")
```

En este código realizo lo mismo que anteriormente; genero una muestra aleatoria, la etiqueto con la función dada en el enunciado aplicándole ruido, y posteriormente creo otro conjunto de datos con el vector de características que nos dice el apartado. Con este conjunto de datos y las etiquetas calculadas realizamos la regresión lineal. En este apartado no he calculado el error ya que no nos lo pide.

En el siguiente apartado ya sí tenemos que calcular el error fuera de la muestra. Realizamos

el mismo procedimiento y posteriormente creamos una nueva muestra y le aplicamos la función f para crear la etiqueta y después la etiquetamos con los pesos tras la regresión lineal (en esta muestra no genero ruido). Después obtenemos un error sobre el 7 %.

```
        Error <- vector()
        for(i in 1:1000)
        {
                #Genero una muestra aleatoria a partir de la cual aprender
                muestra_8 <- simula_unif(N=1000, dim=2, c(-10,10))
                muestra_8 <- t(muestra_8)
                muestra_8 <- as.data.frame(muestra_8)
                names(muestra_8) <- c("x1","x2")
                #creo la nueva muestra con las variables del enunciado
                muestra_8_c <- data.frame(muestra_8$x2^2, muestra_8$x1^2, muestra_8$x1*muestra_8$x2, muestra_8$x1, muestra_8$x2, rep(1, nrow(muestra_8)))
                label_8_c <- muestra_8$x1^2+muestra_8$x2^2-25.0
                label_8_c[label_8_c>0] <- 1
                label_8_c[label_8_c<0] <- -1
                probs <- rep(1,length(label_8_c))
                ind_aleatorios <- sample(x=(1:length(label_8_c)), prob=probs, size=length(label_8_c)/10)
                #Le cambio el signo a la muestra aleatoria elegida
                label_8_c[ind_aleatorios] <- label_8_c[ind_aleatorios]*(-1)
                label_8_c <- matrix(label_8_c, ncol=1)
                pesos_8c <- Regress_lin(datos=muestra_8_c, label=label_8_c)
                #Genero la muestra con la que averiguar el error fuera de la muestra
                muestra_8_out <- simula_unif(N=1000, dim=2, c(-10,10))
                muestra_8_out <- t(muestra_8_out)
                muestra_8_out <- as.data.frame(muestra_8_out)
                names(muestra_8_out) <- c("x1","x2")
                #Calculo su etiquetado para la funci n f
                label_8_out <- muestra_8_out$x1^2+muestra_8_out$x2^2-25.0
                label_8_out[label_8_out>0] <- 1
                label_8_out[label_8_out<0] <- -1
                #Convierto el conjunto de variables para linealizar la funci n
                muestra_8_out <- data.frame(muestra_8_out$x2^2, muestra_8_out$x1^2, muestra_8_out$x1*muestra_8_out$x2, muestra_8_out$x1, muestra_8_out$x2, rep(1, nrow(muestra_8_out)))
```

```
31                  #Eval o con los pesos obtenidos anteriormente
32                  label_test_8 <- evaluacion(x=muestra_8_out, w=
                       pesos_8c)
33                  errores <- label_8_out!=label_test_8
34                  errores <- label_8_out[errores]
35                  Error[i] <- length(errores)/1000
36          }
```

Como vemos obtenemos mucho menos error que en el primer apartado. Esto es debido a que hemos linealizado el problema. Además, si no hubiese ruido en las etiquetas con las que aprendemos, el error sería bastante más bajo.

```
> cat((Error_medio/1000)*100)
7.2679
> cat("%")
%
```

Figura 3.8: Error ejercicio 8(2)

4. Conclusiones

Como hemos visto PLA es un buen algoritmo de aprendizaje para datos linealmente separables. Si los datos no presentan una separabilidad lineal o ruido, el proceso de aprendizaje resulta de menos calidad, aunque podemos 'optimizar' el error usando variaciones como Pocket PLA.

En cuanto a la regresión lineal, también obtenemos buenas soluciones, ya que lo que hace es optimizar el error de dentro de la muestra. Además, combinado con PLA puede dar muy buenos resultados y resultar más eficiente al necesitar menos iteraciones el PLA para converger.