Práctica 1: Modelos Lineales

Ismael Marin Molina

23 de marzo de 2018

Compañero: Manuel Herrera Ojeda

Búsqueda iterativa de óptimos

Para la busqueda del óptimo en estos problemas haremos uso de modelos de regresión lineal, los cuales buscan encontrar $h : E_{out}(h) = min(\mathbb{E}[(h(x) - y)^2])$ es decir busca encontrar un hiperplano en el espacio de problema que minimize el error entre la variable predicha y la etiqueta que queremos encontrar. Un problema que esto tiene es que en la mayoría de los casos E_{out} no puede ser procesada, o bien por falta de ese conocimiento o por que su extención es demasiado grande como para ser posible su cómputo, es por eso que se opta por explorar el error en una muestra representativa de la población, E_{in}

Entre los algoritmos existentes hemos programado el **Gradiente descendente** en el cual iterativamente se van ajustando los pesos para adecuarse a la poblacion.

```
#Funcion para un gradiente descendente generico
# expr -> Exprecion del error a evaluar
# expr.D -> Derivada/s de la expresion de error
# variables -> variables de la operacion es decir los pesos
# theta -> Tasa de aprendizaje
# epsilon -> error asumible
gradiente <- function(expr, expr.D, variables, theta, epsilon){

  err = eval(expr)^2

  while(err > epsilon){
    variables = variables - theta * eval(expr.Du)

    err = eval(expr)^2
  }

  variables

}
```

En este algoritmo dada una funcion $E_{in}(w)$ busca minimizar esa función lo máximo posible para encontrar un punto mínimo. El gradiente descendente es un algoritmo que se basa en el uso de la derivada de la funcion y en la tasa de aprendizaje para generar un movimiento de e ir acercandose a un mínimo, es decir:

$$\bigtriangledown E_{in} \leq \mu ||E_{in}(w(0))||$$

donde μ es nuestra tasa de aprendizaje que determinara los pasos que el algoritmo dara con cada iteracion.

Apartado 2

Dada una función de error como la siguiente $E(u, v) = (u^3 e^{(v-2)} - 4v^3 e^{-u})^2$ debemos calcular el mínimo empezando en el punto $(u, v) = (1, 1)$ y una tasa de aprendizaje $\mu = 0.1$.

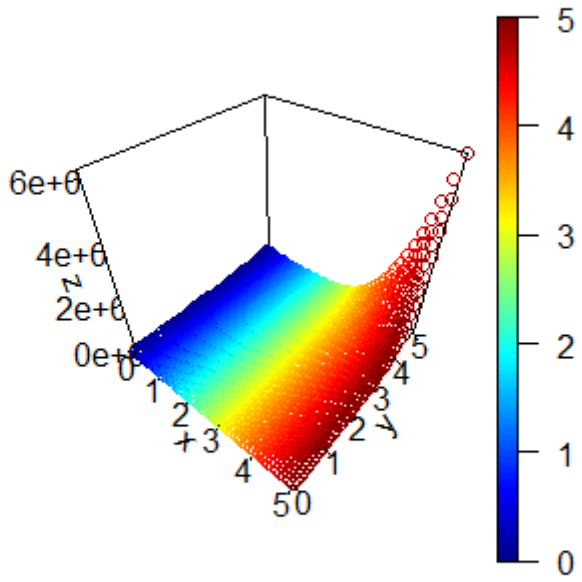


Figure 1: $E(u, v) = (u^3 e^{(v-2)} - 4v^3 e^{-u})^2$

Esta funcion tiene la siguiente gráfica

En el cual podemos ver que la función parece tener un mínimo $(u, v) = (0, 0)$ para comprobarlo usaremos la funcion como $E_{in}(u, v)$ intentando minimizar la función, con este propósito calculamos las derivadas de la función:

$$\frac{\delta}{\delta(u,v)} E_{in}(u,v) = \begin{cases} 2(e^{(v-2)}u^3 - 4e^{-u}v^3)(4e^{-u}v^3 + 3e^{(v-2)}u^2) \\ 2(e^{(v-2)}u^3 - 4e^{-u}v^3)(e^{(v-2)}u^3 - 12e^{-u}v^2) \end{cases}$$

Aquí esta la funcion para realizar la labor correspondiente

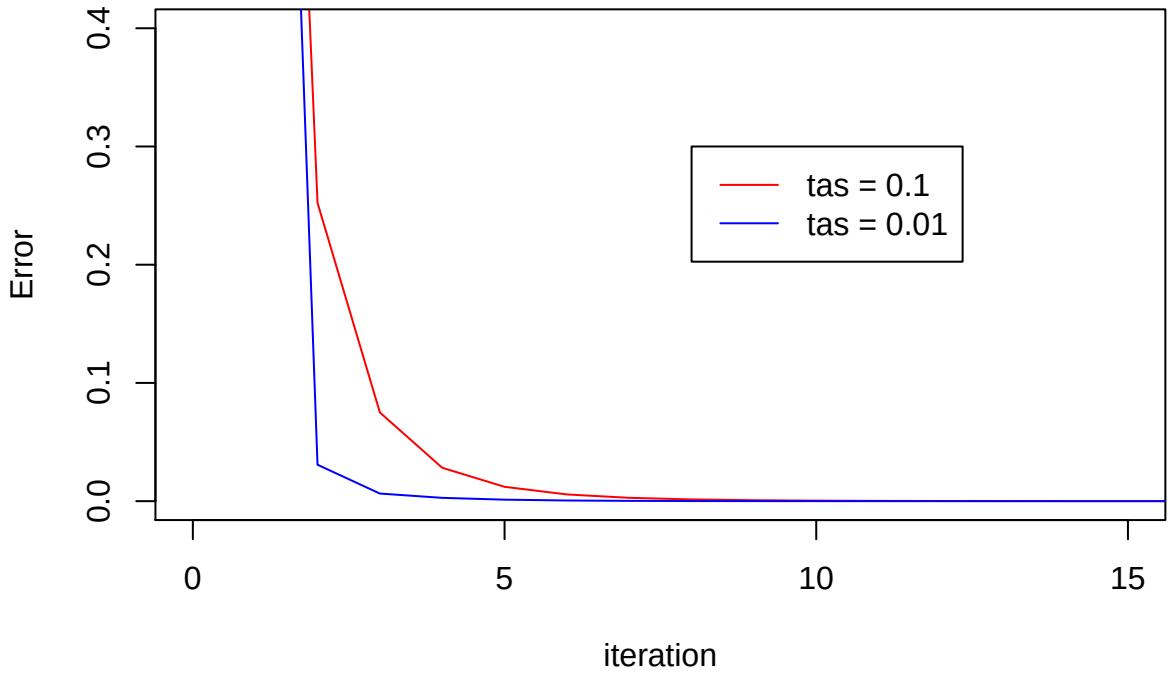
```
library(utils)
library(base)

gradiente <- function(u, v, theta, epsilon){
  expr=expression((  (u^3 * exp(v-2)) - (4 * v^3 * exp(-u))  )^2)
  expr.Du = D(expr, "u")
  expr.Dv = D(expr, "v")

  err = eval(expr)^2

  it = 1

  valores_u = c(u)
  valores_v = c(v)
  error     = c(err)

  while(err > epsilon){
    # Mirar como cambiarlo para varias caracteristicas
    u = u - theta * eval(expr.Du)
    v = v - theta * eval(expr.Dv)


    err = eval(expr)^2
    it = it +1  #Cuenta iteraciones


    valores_u = append(valores_u,u)
    valores_v = append(valores_v,v)
    error     = append(error,err)
  }

  data.frame(valores_u, valores_v,error)

}
```

El algoritmo acepta como entrada los valores iniciales de u y v , como tambien el valor de la tasa de aprendizaje y la tasa de aceptación del error. Calcula las derivadas de la función y evalua el error inicial antes de iniciar el bucle en el cual va alterando los valores de u y v para aproximarlos a el mínimo de la función para el calculo de los nuevos pesos usa la siguiente fórmula $w_{new} = w_{old} - \mu * \frac{\delta}{\delta w} E_{in}(w)$ finalmente devuleve un historico de errores y valores de la función.

Los resultados finales son los siguientes:

Table 1: Tabla de resultados

u	v
0.9720762	0.5161040
1.2176655	0.7620631



Podemos ver que la convergencia de la tasa más pequeña es anterior que la de la mayor y por un punto más proximo que $\mu = 0.1$ salto, debido a esto la funcion con una tasa mayo dio un mayor recorrido y termino en **56** iteraciones mientras que la de $\mu = 0.01$ termino en **23**.

Apartado 3

Para la segunda parte se nos pide calcular los mínimos de $E_{in}(x, y) = (x-2)^2 + 2(y+2) + 2sin(2\pi x)sin(2\pi y)$ en $\mu = 0.01$ parecido al apartado anterior pero en este podemos ver uno de los problemas que presenta el algoritmo del gradiente descendente y es que al no tener mecanismos para evitar mínimos locales es susceptible de caer en ellos.

$$\frac{\delta}{\delta(x,y)} E_{in}(x,y) = \begin{cases} 2(x-2) \\ 4(2+y) + 4\pi cos(2\pi y) \end{cases}$$

Como podemos ver estamos ante una funcion con varios minimos locales por lo tanto esta función dependera de donde empieze la operación Para ello usamos los siguientes valores de entrada en el algoritmo.

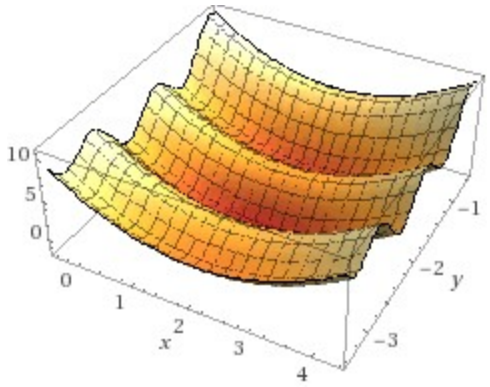
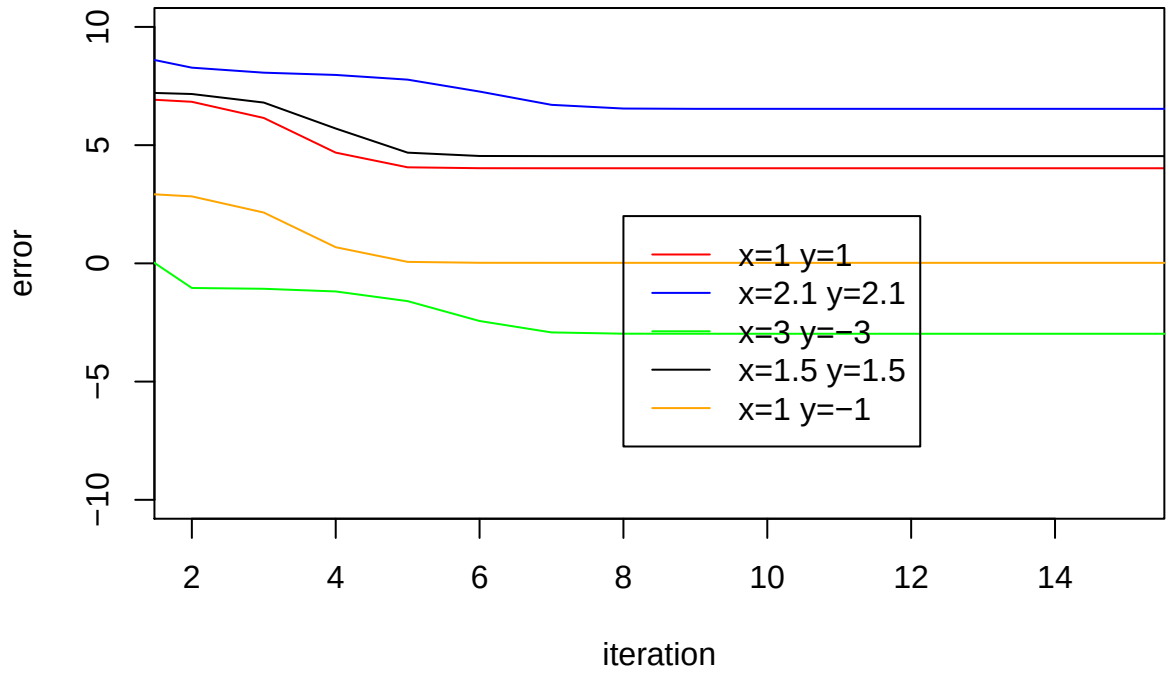


Figure 2: $E_{in}(x, y) = (x-2)^2 + 2(y+2) + 2sin(2\pi x)sin(2\pi y)$

x	y
1	1
2.1	2.1
3	-3
1.5	1.5
1	-1



Como podemos ver en la gráfica cada recta se estanca en diferentes puntos y con diferente número de iteraciones (lo hemos dejado en 14 ya que todos se estancan apartir de ese punto)

Table 3: Tabla de resultados

a	b	c	d	e
7.000000	8.900983	1.000000	7.250000	3.0000000
6.833067	8.276567	-1.041407	7.162425	2.8330672
6.149774	8.064636	-1.074692	6.801111	2.1497740
4.683565	7.969418	-1.187772	5.703461	0.6835650
4.062749	7.770690	-1.596815	4.683087	0.0627487
4.024668	7.266137	-2.437401	4.541702	0.0246685
4.023103	6.705320	-2.920281	4.535806	0.0231032
4.023031	6.548349	-2.974331	4.535563	0.0230310
4.023028	6.536166	-2.976862	4.535553	0.0230276
4.023027	6.535579	-2.976968	4.535552	0.0230274
4.023027	6.535553	-2.976972	4.535552	0.0230274
4.023027	6.535552	-2.976973	4.535552	0.0230274
4.023027	6.535552	-2.976973	4.535552	0.0230274
4.023027	6.535552	-2.976973	4.535552	0.0230274
4.023027	6.535552	-2.976973	4.535552	0.0230274
4.023027	6.535552	-2.976973	4.535552	0.0230274
4.023027	6.535552	-2.976973	4.535552	0.0230274
4.023027	6.535552	-2.976973	4.535552	0.0230274
4.023027	6.535552	-2.976973	4.535552	0.0230274
4.023027	6.535552	-2.976973	4.535552	0.0230274

Table 4: Puntos donde se obtienen los valores

X	Y	Value
1.268808	0.7243813	4.0230274
1.756256	2.2245417	6.5355521
2.731192	-2.7756187	-2.9769726
1.756256	1.2245417	4.5355521
1.268808	-1.2756187	0.0230274

X	Y	Value

El algoritmo que se ha encargado del calculo es el siguiente

```
gradiente2 <- function(x, y, theta, iteration){
  expr=expression( (x-2)^2 + 2*(y+2) + 2*sin(2*pi*x) * sin(2*pi*y) )
  expr.Dx = D(expr, "x")
  expr.Dy = D(expr, "y")

  err = abs(eval(expr))

  valores_x = c(x)
  valores_y = c(y)
  error     = c(err)

  for(i in 1:iteration){
    # Mirar como cambiarlo para varias caracteristicas
    x = x - theta * eval(expr.Dx)
    y = y - theta * eval(expr.Dy)
    err = eval(expr)^2

    valores_x = append(valores_x,x)
    valores_y = append(valores_y,y)
    error     = append(error,eval(expr))

  }

  data.frame(valores_x, valores_y,error)


}
```

Apartado 4

Todos estos valores nos llevan a la conclusion de que obtener el mínimo de una funcion depende entre otras cosas del punto de inicio de la existencia de mínimos locales y de que la tasa de aprendizaje sea la adecuada ya que una tasa de aprendizaje muy alta puede llegar a obviar los puntos mínimos y una tasa muy pequeña harán necesarias muchas más iteracciones del algoritmo por lo que en práctica es mejor utilizar un μ adaptativo, este podría ser $\mu = \mu||\bigtriangledown E_{in}||$ haciendo que μ se adapte a como de cerca esta la solución cambiando. Pero el punto más complicado de evitar es el de los óptimos locales y saber evitarlos ,para eso solo podemos evaluar la funcion en diferentes partes y comprobar si convergen hacia un único punto.

Regresion lineal

Para estos apartados se nos ha pedido que trabajemos sobre números escritos en 16x16, lo cual denotaremos com **D** para intentar distinguir dos números dados usando x_n la simetría del número y y_n su intensidad de color , presentan un problema de clasificación en el cual nuestro objetivo es el siguiente. Dado un **D** en el cual los valores en el cual podemos clasificar los datos de manera binaria $f : \mathcal{X} \rightarrow \{1, -1\}$ queremos encontrar

una funcion tal que

$$\min_{w \in \mathbb{R}^{d+1}} = \frac{1}{N} \sum_{n=1}^{N} [sign(w^t x_n) \neq y_n]$$

o lo que es lo mismo que el signo del valor predecido y el signo de la etiqueta sea el mismo el máximo numero de veces posible pero para ello las muestras deben ser linealmente separables conforme a los valores de $\mathcal{X}$.

Para este propósito usaremos el gradiente estocastico una variante del gradiente descendente que itera sobre **minibatches** para cambiar el peso y no tener que iterar sobre el conjunto total de los valores de la muestra.

El otro algoritmo es la descomposicion en valores singulares del problema el cual es un calculo algebraico y nos da unos pesos óptimos.

Código de cada uno de los problemas

```
#Codigo de la Pseudo inversa
################################################
pseudoInverse <- function( X ){

  udv <- svd( t(X) %*% X) #udv <- svd( X ) %*% t(X) ?
  tmp <- udv$v %*% diag( 1 / ( udv$d ) ) %*%  t( udv$v )
  tmp %*% t(X)
}

linearRegression <- function( X, y ){

  H <- pseudoInverse( X )

  H %*% y
}
############################################

#Código del gradiente estocástico
######################################################################
# Como parametros de entrada tiene la tasa de aprendizaje
# el tamaño proporcional de los minibatches que por defecto es un 10%
# del tamaño de la muestra.
# El numero de iteraciones el cual es por defecto el numero de iteraciones que necesite hasta
# completar el tamño muestral
SGD <- function( X, y, learningRate = 0.05, t = 0.1, itera = 1/t){

  N <- length(y)
  T <- as.integer(N * t)
  #w <- as.vector(t(linearRegression(X,y)))
  w <- rnorm(dim(X)[2])

  positive_exa <-which(y==1)
  negative_exa <-which(y==-1)
  N_po <- length(positive_exa)
  N_ne <- length(negative_exa)

  w_pre = w
  scored = as.numeric(error_in(X,y,w))

  for (a in 1:itera) {
    v <- 0
```

```
    for ( i in 1:T ){
      #obtener una misma proporcion de cada tipo en los minibatches

      pos <- ifelse((i %% 2) > 0,sample(N_po,1),sample(N_ne,1))
      pos <- ifelse((i %% 2) > 0,positive_exa[pos],negative_exa[pos])
      h <- t(w) %*% X[pos,]
      er <- (h * y[pos])

      if(er < 0){
        v = v + (X[pos,] * as.numeric(h-y[pos]))
      }
      #er <- er %*% t(X[pos,])
      #er <- 2*er/N
      #v  <- v - er
      #v <- -( a / b )
    }

    w <- w - learningRate * (v/N)
    w <- as.vector(w)
    # if(scored > error_in(X,y,w)){
    #   scored = as.numeric(error_in(X,y,w))
    #   w_pre = w
    # }else{
    #   w = w_pre
    # }
  }
  #result<-w - learningRate * v
  #as.vector(result)
  w
}
```

El resultado de ejecutar el código ha dado los siguientes pesos:

$$w_{sgd} = [-1.25868581, 0.0201459, 0.68104408] w_{svd} = [0.73182914, 0.01402067, 1.71412717]$$

con las siguientes aciertos:

tipo	train 1_5	test 1_5	train 1_8	test 1_8	train 2_8	test 2_8
SGD	0.996661	0.9591837	0.9583952	0.9482759	0.547059	0.5714286
SVD	0.998331	0.9591837	0.9494799	0.9655172	0.696078	0.5396825

Las gráficas de la funcion son las siguientes:

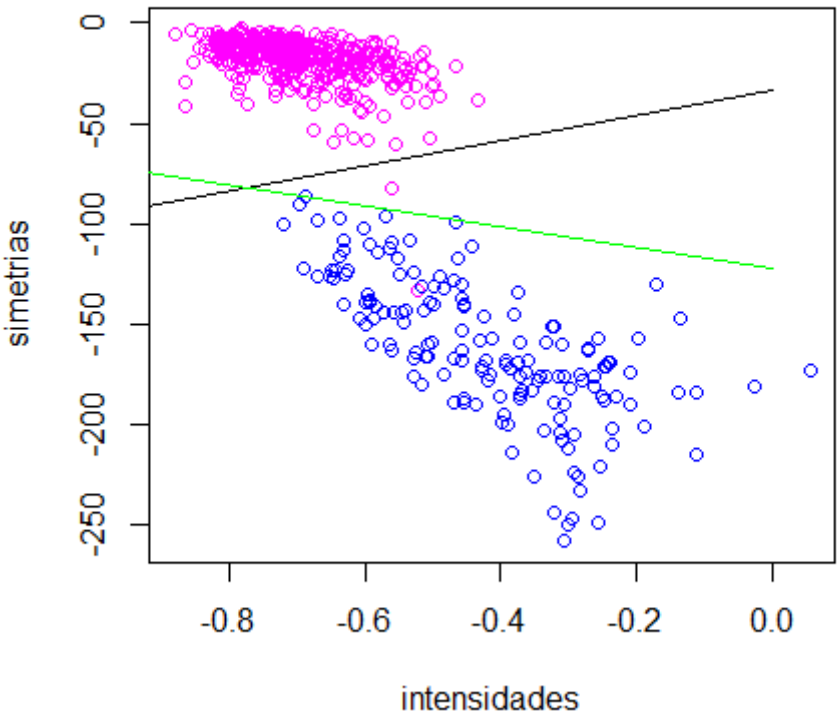


Figure 3: Clasificacion de los 1 y los 5

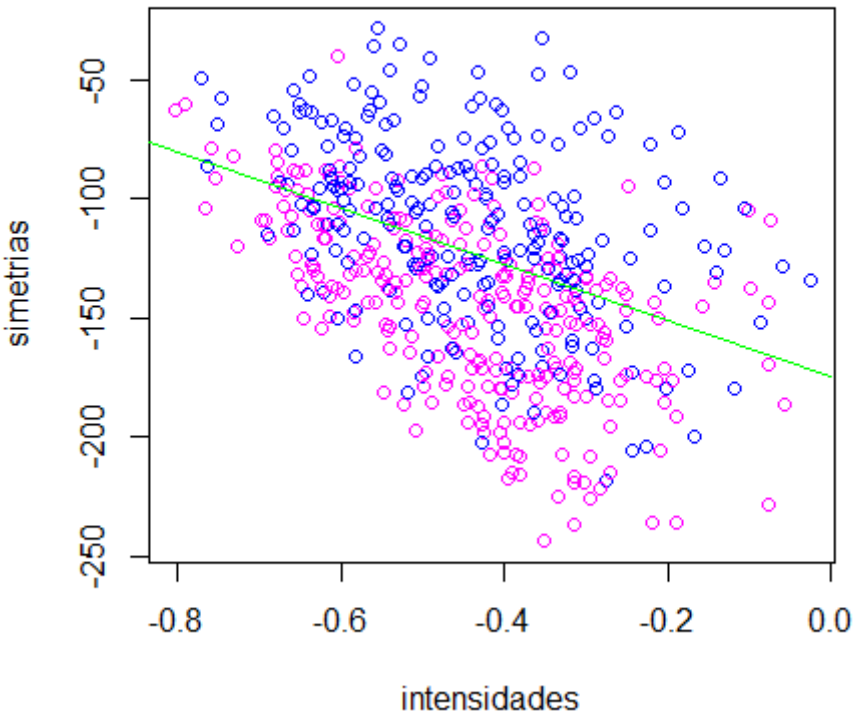


Figure 4: Clasificacion de los 2 y los 8

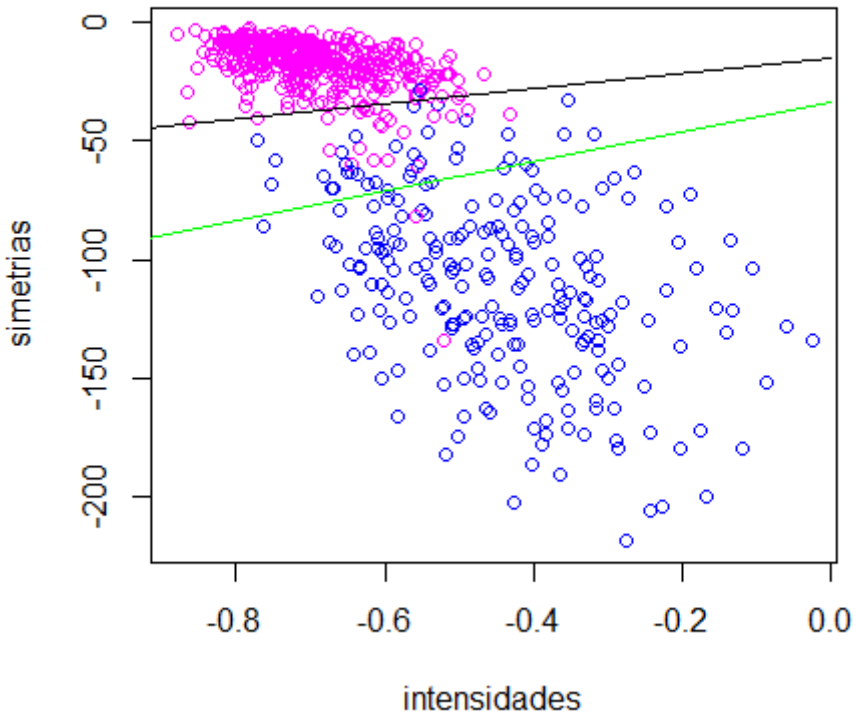


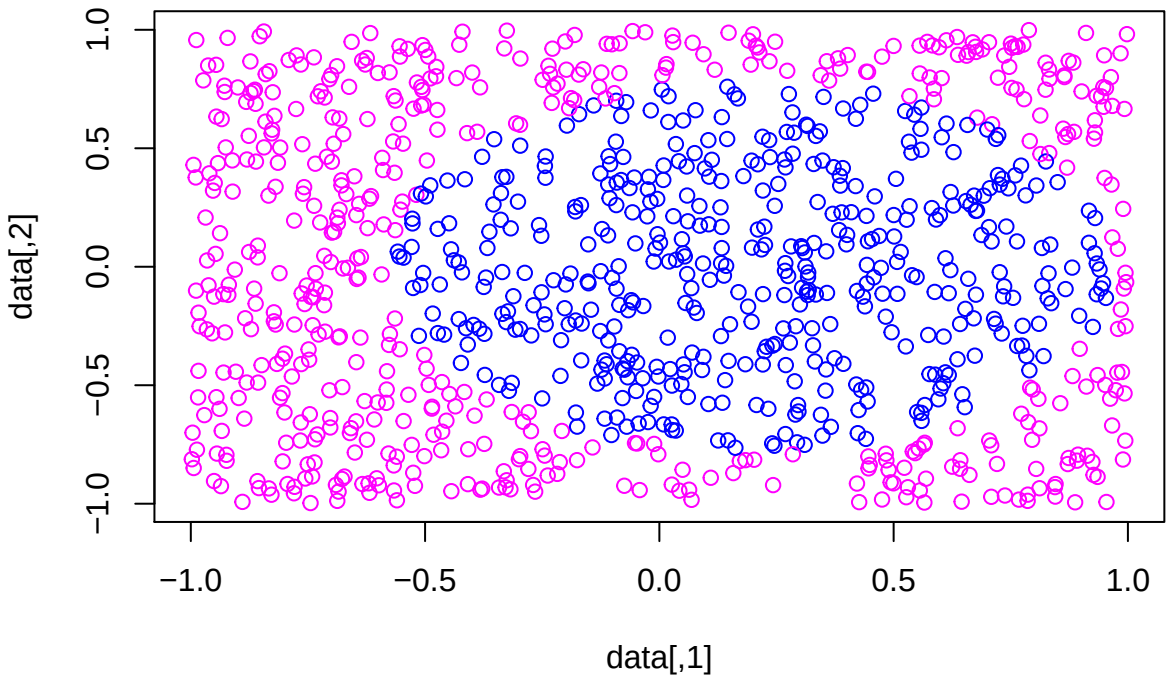
Figure 5: Clasificacion de los 1 y los 8

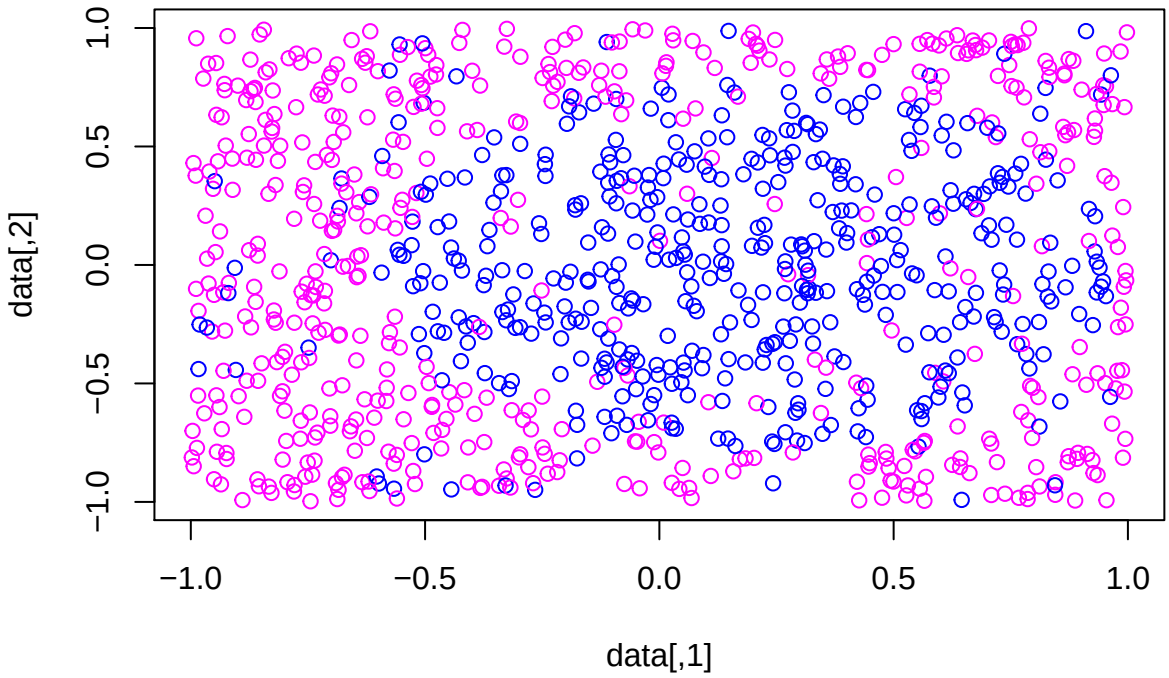
Apartado 2

Para este ultimo apartado se nos pide que hagamos un experimento con datos generados aleatoriamente haciendo uso de la funcion

```
simula_unif = function (N=2,dims=2, rango = c(0,1)){
  m = matrix(runif(N*dims, min=rango[1], max=rango[2]),
             nrow = N, ncol=dims, byrow=T)
    m
}

N <- 1000
size <- 1
data <- simula_unif( N, rango = c(-size,size) )
```

Obtenemos una grafica de puntos a la cual le introducimos ruido a traves de la función $y = sign((x_1 - 0.2)^2 + x_2^2 - 0.6)$ de esta forma introducimos valores mal clasificado en la funcion quedando esta gráfica como sigue.

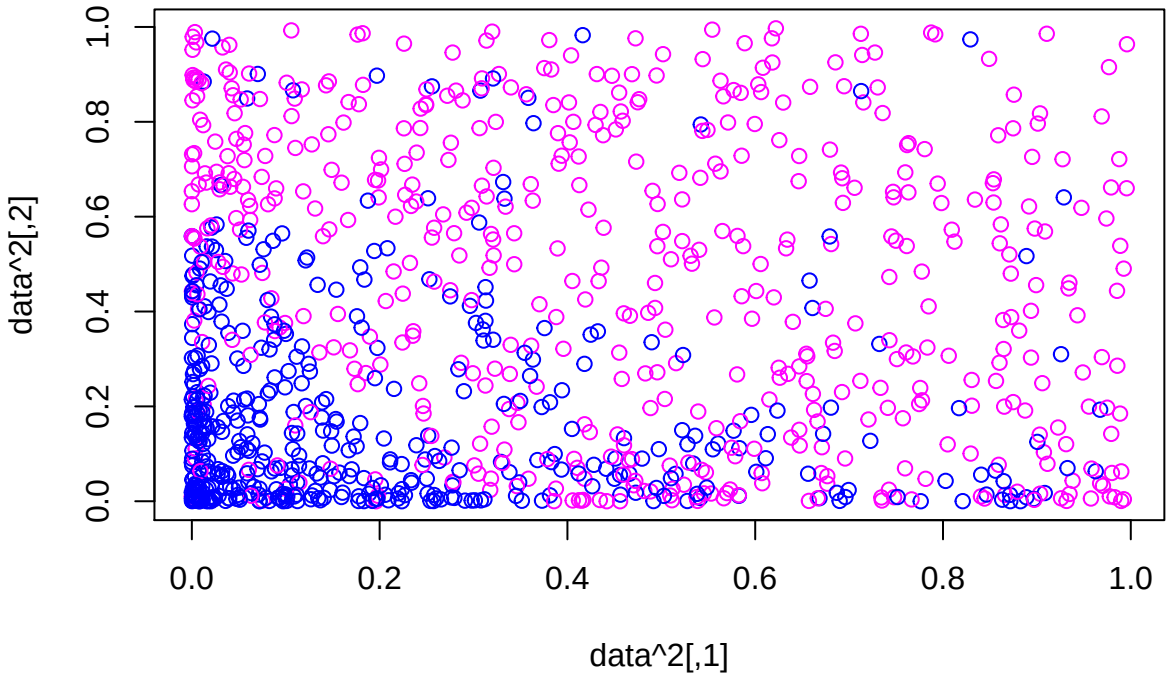




Tras usar el gradiente estocastico 1000 veces sobre mapas de puntos generados en cada iteracion el error dado ha sido el siguiente, hemos usado la descomposicion en valores singulares tambien para poder observar la comparativa.

tipo	E_{in}	E_{out}
SGD	0.497087	0.497583
SVD	0.606083	0.504341

Podemos observar que los errores estan entorno al 50% lo cual tiene sentido ya que de los valores generados aquellos que tienen la etiqueta -1 se encuentran en los extremos de la gráfica mientras que los valores centrales los ocupan los valores de 1 formando un cluster que no es posible distinguirlo tal como esta pero si aplicamos una tranformacion a los datos tal que $\phi(\mathcal{X}) = \mathcal{X}^2$ se nos queda una gráfica como la siguiente



Mejorando la predicion del SVD y el SGD en un 10 por ciento. Hemos tranformado el conjunto $\mathcal{D}$ que no era linealmente separable ha un nuevo espacio de trabajo que en el que si lo es el cual denotaremos como $\mathcal{Z}$