Aprendizaje Automático: Memoria Práctica 1

Óscar López Arcos. 75571640

1. Ejercicio sobre la búsqueda iterativa de óptimos

1. Implementar algoritmo de gradiente descendente

```
# Ejercicio 1

gradientDescent <- function(w,n,maxiters,epsilon,funcion,derivada){

  X <- w
  Y <- funcion(X)

  error <- (1/N)*( norm((X%*%w)-Y,"F")^2 )
  i=1
  while( (i < maxiters) & (epsilon < error) ){
    w = w - n * derivada(w)

    error <- (1/N)*( norm((X%*%w)-Y,"F")^2 )
    i=i+1
  }
  w
}
```

2. Considerar la función $E(u,v) = (u^3e^{v-2} - 4v^3e^{-u})^2$

a) Calcular analíticamente y mostrar la expresión del gradiente de la función $E(u,v)$

"$\nabla E = \left(\frac{\partial E}{\partial u}, \frac{\partial E}{\partial v}\right) = \left(2((3u^2e^{(v-2)} + 4v^3e^{-u})(u^3e^{(v-2)} - 4v^3e^{-u})), 2((u^3e^{(v-2)} - 12v^2e^{-u})(u^3e^{(v-2)} - 4v^3e^{-u}))\right)$ "

b) ¿Cuántas iteraciones tarda el algoritmo en obtener por primera vez un valor de E(u, v) inferior a 10^{-14}?

c) ¿En qué coordenadas?

```
# Ejercicios 1.2bc

E <- function(u,v){
  (((u^3)*exp(v-2))-(4*(v^3)*exp(-u)))^2
}

DerivadaE <- function(w){
  u = w[1]
  v = w[2]

  du = 2 * ((3 * u^2 * exp(v - 2) + 4 * (v^3) * exp(-u)) *
              (((u^3) * exp(v - 2)) - (4 * (v^3) * exp(-u))))
```

```
  dv = 2 * (((u^3) * exp(v - 2) - 4 * (3 * v^2) * exp(-u)) *
              (((u^3) * exp(v - 2)) - (4 * (v^3) * exp(-u))))

  c(du,dv)
}

gradientDescent <- function(w,n,maxiters,funcion,derivada){

  control <- 1000000

  i=1
  fin = FALSE
  while(i < maxiters & !fin){
    w = w - n * derivada(w)

    if(i == control){
      sprintf("Iteracion: %i \n", i)
      control <- (control+control)
    }

    if(funcion(w[1],w[2]) < 10^(-14)){
      result = c(i, w)
      fin = TRUE
    }
    i=i+1
  }
  names(result) <- c("Iteración","Coordenada w[1]", "Coordenada w[2]")
  result
}

w <- c(1,1)
result <- gradientDescent(w,0.05,300000000000,E,DerivadaE)
result
```

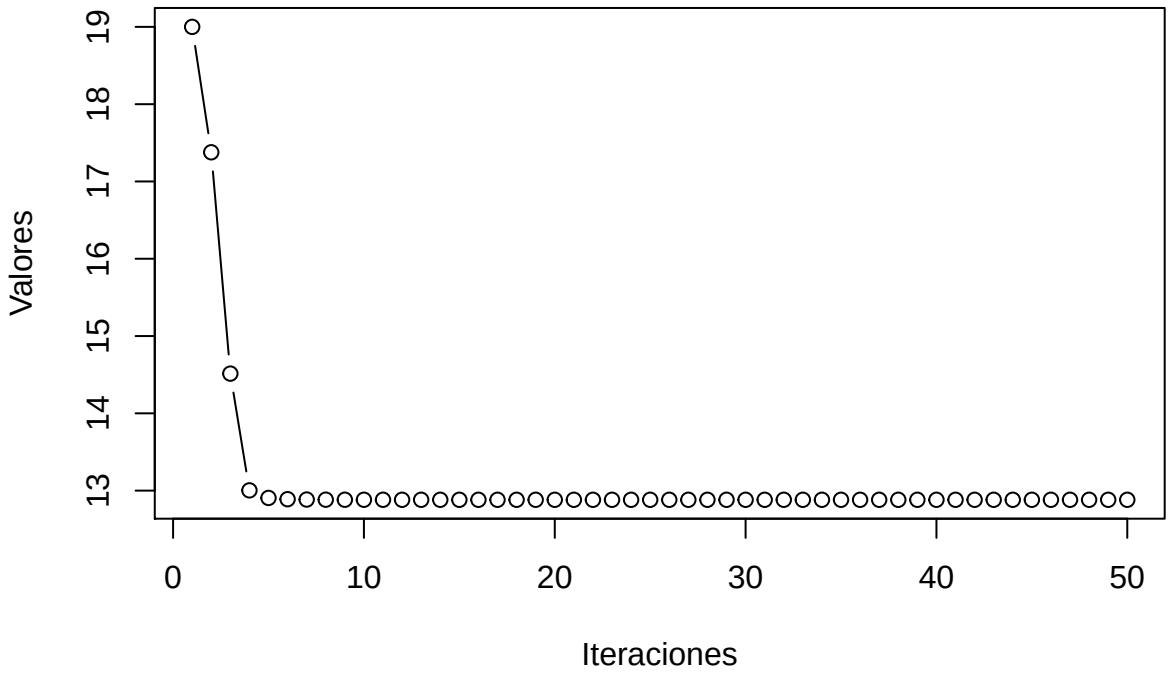
```
##       Iteración Coordenada w[1] Coordenada w[2]
##      38.0000000       1.1195439       0.6539881
```

3. Considerar ahora la función $f(x, y) = (x - 2)^2 + 2(y + 2)2 + 2sin(2\pi x)sin(2\pi y)$

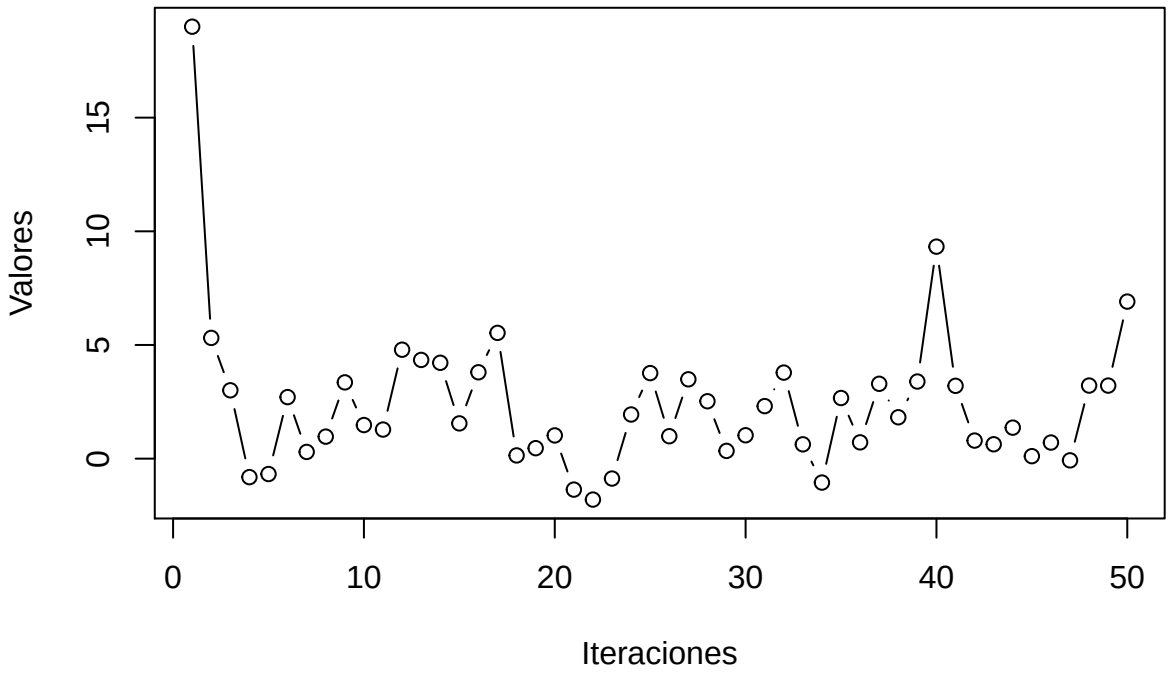
a) Usar gradiente descendente para minimizar esta función. Repetir el experimento para $\eta = 0,01$ y $\eta = 0,1$, comentar las diferencias y su dependencia de η.

```
F <- function(x,y){
  (x-2)^2 + 2*(y + 2)^2 + 2*sin(2*pi*x)*sin(2*pi*y)
}

DerivadaF <- function(w){
  x = w[1]
  y = w[2]

  dx = 2*(2*pi*cos(2*pi*x)*sin(2*pi*y)+x-2)
  dy = 4*(pi*sin(2*pi*x)*cos(2*pi*y)+y+2)
```

```
  c(dx,dy)
}

gradientDescent3a <- function(w,n,maxiters,funcion,derivada){

  i=1
  iteraciones=0
  valores=0

  while(i <= maxiters ){

    valores[i] <- funcion(w[1],w[2])
    w = w - n * derivada(w)

    i=i+1
  }

  cbind(Iteraciones=1:maxiters, Valores=valores)
}

w <- c(1,1)
result1 <- gradientDescent3a(w,0.01,50,F,DerivadaF)
result2 <- gradientDescent3a(w,0.1,50,F,DerivadaF)
plot(result1,type="b") # resultados para μ = 0.01
```




```
plot(result2,type="b") # resultados para μ = 0.1
```



A la vista de los resultados, la correcta minimización es muy dependiente de la tasa de aprendizaje. Con una tasa demasiado grande, puede que el ajuste oscile de un lado al otro del mínimo, tardando mucho más en encontrarlo.

b)

```
# Ejercicio 1.3b

gradientDescent3b <- function(w,n,maxiters,funcion,derivada){

  i=1
  coordenadas = 0
  min=3000

  while(i <= maxiters ){

    w = w - n * derivada(w)

    aux <- funcion(w[1],w[2])

    if(aux < min){
      min <- aux
      coordenadas[1] <- w[1]
      coordenadas[2] <- w[2]
    }

    i=i+1
  }

  cbind(Mínimo=min, X=coordenadas[1],Y=coordenadas[2])
}
```

```
w<-c(2.1,-2.1)
result1 <- gradientDescent3b(w,0.01,300,F,DerivadaF)
w<-c(3,-3)
result2 <- gradientDescent3b(w,0.01,300,F,DerivadaF)
w<-c(1.5,1.5)
result3 <- gradientDescent3b(w,0.01,300,F,DerivadaF)
w<-c(1,-1)
result4 <- gradientDescent3b(w,0.01,300,F,DerivadaF)

result <- rbind(result1,result2,result3,result4)
result
```

```
##          Mínimo        X         Y
## [1,] -1.8200785 2.243805 -2.237926
## [2,] -0.3812495 2.730936 -2.713279
## [3,] 18.0420723 1.779119  1.030946
## [4,] -0.3812495 1.269064 -1.286721
```

4. ¿Cuál sería su conclusión sobre la verdadera dificultad de encontrar el mínimo global de una función arbitraria?

El ajuste de parámetros tales como la tasa de aprendizaje y el punto de inicio serían la mayor dificultad, ya que a priori no tenemos ninguna información que nos indique qué valor sería el óptimo.

Ejercicio sobre Regresión Lineal

1. Estimar un modelo de regresión lineal a partir de los datos proporcionados los números 5 y 1

Algoritmos de estimación:

```
# Pseudo Inversa
Pseudo_inversa <- function(X, Y){

  X = as.matrix(X)
  Y = as.matrix(Y)

  X <- cbind(X, 1) # cbind(X,1)

  x = t(X) %*% X
  pseudo = svd(x)
  aux = pseudo$v%*%diag(1/pseudo$d)%*%t(pseudo$v)
  pseudoinversa = aux%*%t(X)
  w = pseudoinversa%*%Y
  w
}

# Gradiente Descendente Estocástico

SGD <- function(X, Y, eta = 0.01, umbral = 0.01, max_i = 1000, seed = NULL, p=0.25){
```

```
  X = cbind(X,1)
  theta = sample(0,ncol(X),replace=TRUE)
  fin = TRUE

  i = 0
  indices = c(1:nrow(X))
  set.seed(seed)
  while(i < max_i & fin){
    temporaryTheta = theta
    indices_desordenado = sample(indices)
    stochasticList <- indices_desordenado[1:((p*length(indices_desordenado)))]

    for(i in stochasticList){
      theta = theta - eta*(-Y[i]*X[i,])/as.vector((1+exp(Y[i]*(theta%*%X[i,]))))
    }

    Yprima <- sign(X%*%theta)
    Ein <- Err(Yprima,Y,binary_error)
    if( Ein < umbral )
      fin = FALSE

    if(sqrt(sum((temporaryTheta - theta)^2)) < umbral)
      fin = FALSE
    i = i + 1
  }
  set.seed(NULL)
  theta
}
```

Funciones para la clasificación:

```
# Clasifica según el ajuste
Clasificar <- function(X,w){
  X <- cbind(X, 1) # cbind(X,1)
  sign(X%*%w)
}

## Error binario
binary_error <- function(Yprima, Y){
  error <- sample(0,length(Y),replace = T)
  error[ Yprima != Y ] = 1
  error
}

# Calculo de Ein y Eout
Err <- function(Yprima, Y, error){
  sum(error(Yprima,Y))/length(Y)
}

# Calcular la recta asociada al vector de pesos
pasoARecta= function(w){
  if(length(w)!= 3)
    stop("Solo tiene sentido con 3 pesos")
```

```
  a = -w[1]/w[2]
  b = -w[3]/w[2]
  c(a,b)
}

# Calcula la coordenada Y de X dado una pendiente y desplazamiento
recta <- function(x,a,b){
  y = a*(x)+b
}
```

Lectura de los datos proporcionados para la muestra de entrenamiento:

```
# Cargar los datos:

digit.train <- read.table("./datos/zip.train", quote="\"",
                          comment.char="", stringsAsFactors=FALSE)
```

```
## Warning in scan(file = file, what = what, sep = sep, quote = quote, dec =
## dec, : número de items leídos no es múltiplo del número de columnas
```

```
digitos15.train = digit.train[digit.train$V1==1 | digit.train$V1==5,]
digitos = digitos15.train[,1]
ndigitos = nrow(digitos15.train)

grises = array(unlist(subset(digitos15.train,select=-V1)),c(ndigitos,16,16))

rm(digit.train)
rm(digitos15.train)

# Definir simetría
fsimetria <- function(A){
  A = abs(A-A[,ncol(A):1])
  -sum(A)
}

# Cargar Intensidad y Simetría
intensidadtr <- apply(grises,1,mean)
simetriatr <- apply(grises,1,fsimetria)

#Guardar datos y etiquetas (X,Y)
datosTr = as.matrix(cbind(intensidadtr,simetriatr))
etiquetasTr = digitos
etiquetasTr[etiquetasTr==5]=-1
etiquetasTr <- as.matrix(etiquetasTr)
```

Aplicación de algoritmos para la muestra de entrenamiento:

```
# Obtención de pesos
w_pseudo <- Pseudo_inversa(datosTr,etiquetasTr)
w_sgd <- SGD(datosTr,etiquetasTr,seed=1)

# Clasificación
EtiquetasPrima_pseudo <- Clasificar(datosTr,w_pseudo)
EtiquetasPrima_sgd <- Clasificar(datosTr,w_sgd)
```

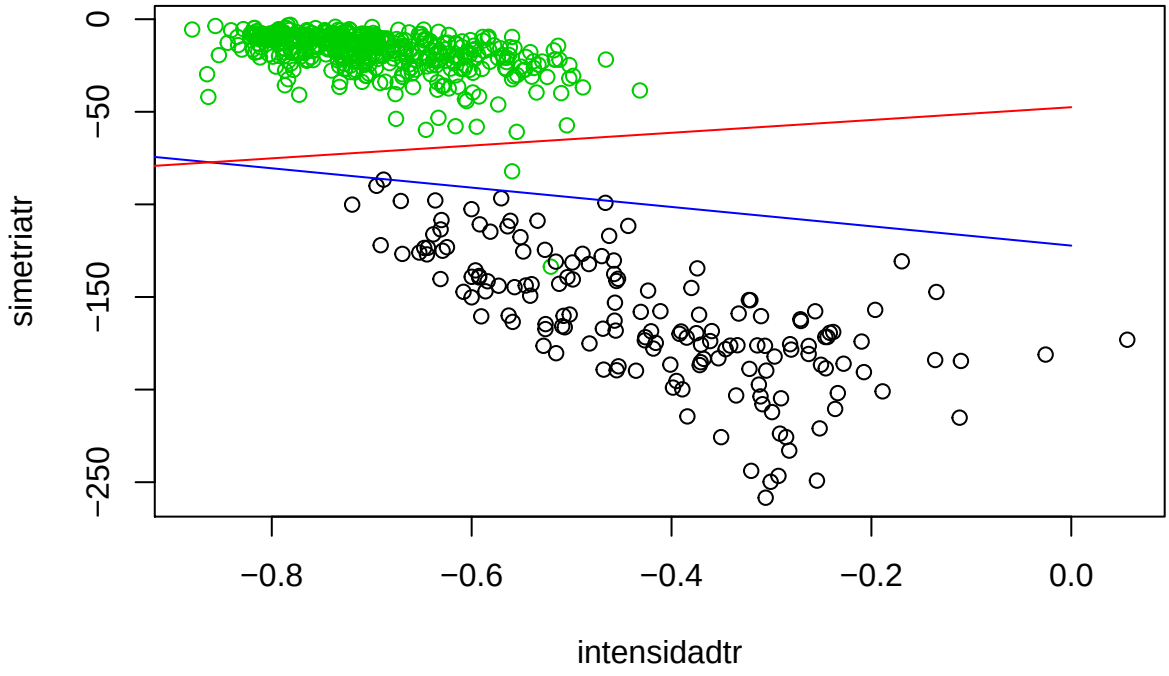
```
# Error interno
Ein_pseudo <- Err(EtiquetasPrima_pseudo,etiquetasTr,binary_error)
Ein_pseudo
```

```
## [1] 0.001669449
```

```
Ein_sgd <- Err(EtiquetasPrima_sgd,etiquetasTr,binary_error)
Ein_sgd
```

```
## [1] 0.003338898
```

```
# Obtención de la recta asociada a los pesos y representación gráfica

recta_pseudo <- pasoARecta(w_pseudo)
recta_sgd <- pasoARecta(w_sgd)

# Obtención de puntos
x1=0
y1_ps = recta(x1,recta_pseudo[1],recta_pseudo[2])
y1_sg = recta(x1,recta_sgd[1],recta_sgd[2])

x2=-5000
y2_ps= recta(x2,recta_pseudo[1],recta_pseudo[2])
y2_sg= recta(x2,recta_sgd[1],recta_sgd[2])

# Dibujo datosTr y ambas rectas (Pseudo = AZUL, SGD = ROJA)
plot( intensidadtr, simetriatr, col=etiquetasTr+2)
lines(c(x1,x2),c(y1_ps,y2_ps), col="blue",type="l")
lines(c(x1,x2),c(y1_sg,y2_sg), col="red",type="l")
```



Comprobación en la muestra Test:

```r
# Cargar los datos test:

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

```
## Warning in scan(file = file, what = what, sep = sep, quote = quote, dec =
## dec, : número de items leídos no es múltiplo del número de columnas
```

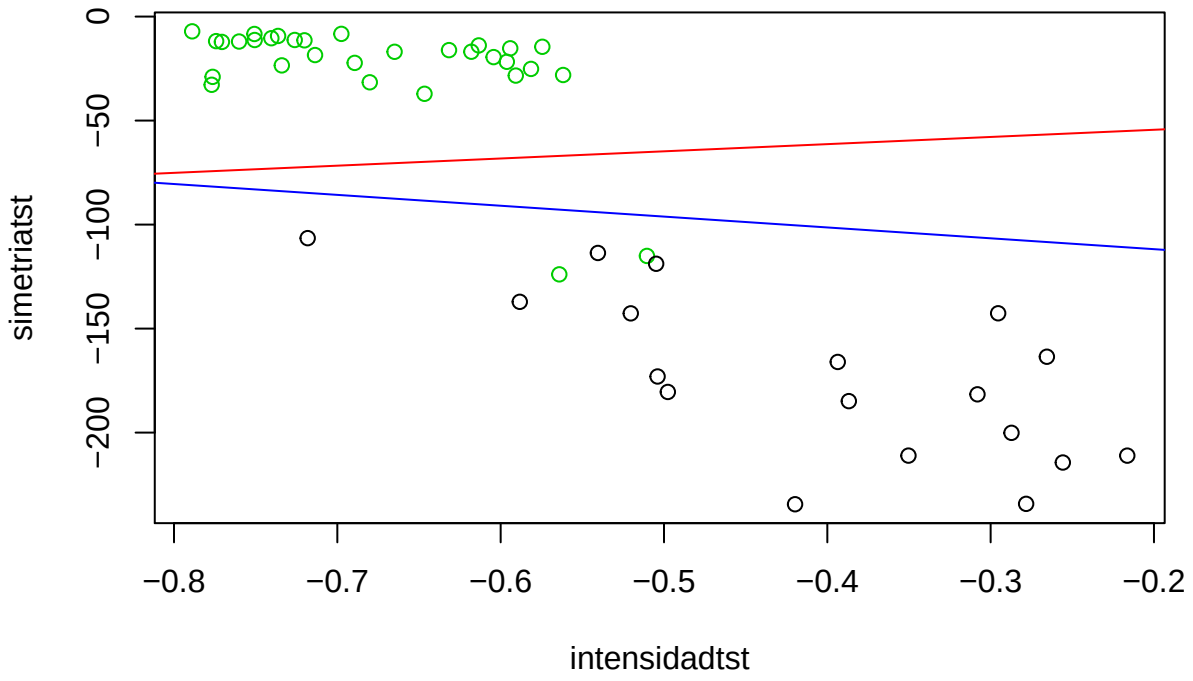
```r
digitos15.test = digit.test[digit.test$V1==1 | digit.test$V1==5,]
digitos = digitos15.test[,1]    # vector de etiquetas del train
ndigitos = nrow(digitos15.test)

grises = array(unlist(subset(digitos15.test,select=-V1)),c(ndigitos,16,16))
grises = as.numeric(grises)
dim(grises)=c(49,16,16)
rm(digit.test)
rm(digitos15.test)

intensidadtst <- apply(grises,1,mean)
simetriatst <- apply(grises,1,fsimetria)

datosTest = as.matrix(cbind(intensidadtst,simetriatst))
etiquetasTest = digitos
etiquetasTest[etiquetasTest==5]=-1
etiquetasTest <- as.matrix(etiquetasTest)

# Clasificación
EtiquetasPrima_pseudo <- Clasificar(datosTest,w_pseudo)
EtiquetasPrima_sgd <- Clasificar(datosTest,w_sgd)

# Error
Eout_pseudo <- Err(EtiquetasPrima_pseudo,etiquetasTest,binary_error)
Eout_pseudo
```

```
## [1] 0.04081633
```

```r
Eout_sgd <- Err(EtiquetasPrima_sgd,etiquetasTest,binary_error)
Eout_sgd
```

```
## [1] 0.04081633
```

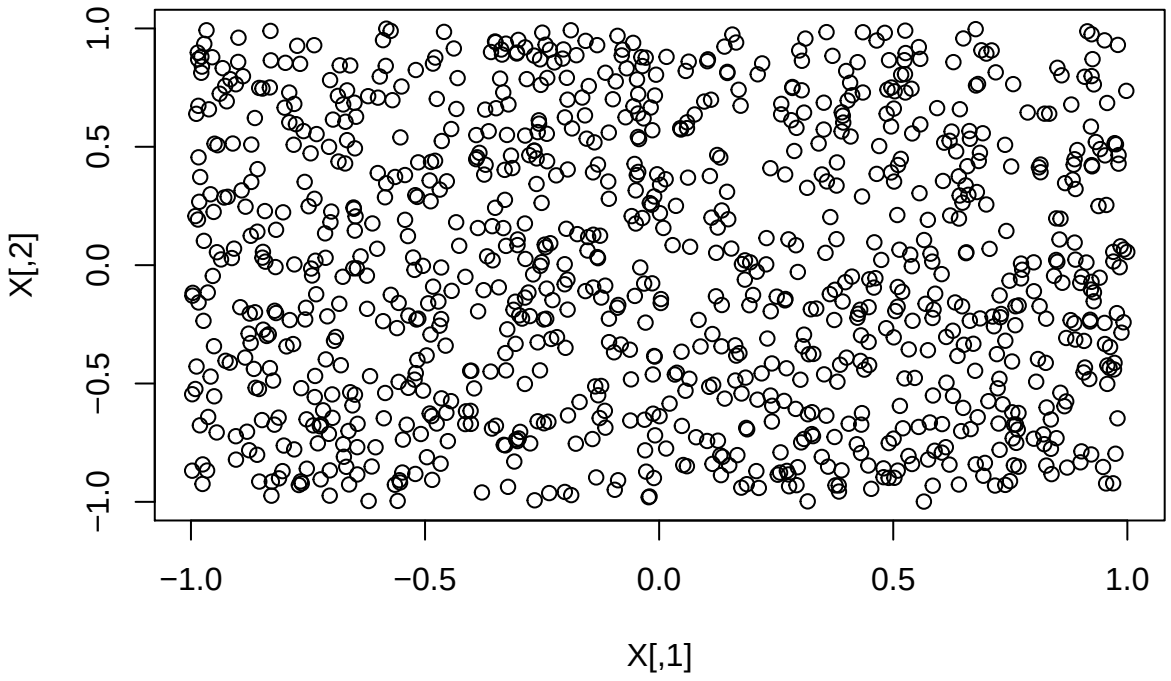
```r
# Dibujo datosTest y ambas rectas (Pseudo = AZUL, SGD = ROJA)
plot( intensidadtst, simetriatst, col=etiquetasTest+2)
lines(c(x1,x2),c(y1_ps,y2_ps), col="blue",type="l")
lines(c(x1,x2),c(y1_sg,y2_sg), col="red",type="l")
```



2. Explorar cómo se transforman los errores E_{in} y E_{out} cuando aumentamos la complejidad del modelo lineal usado

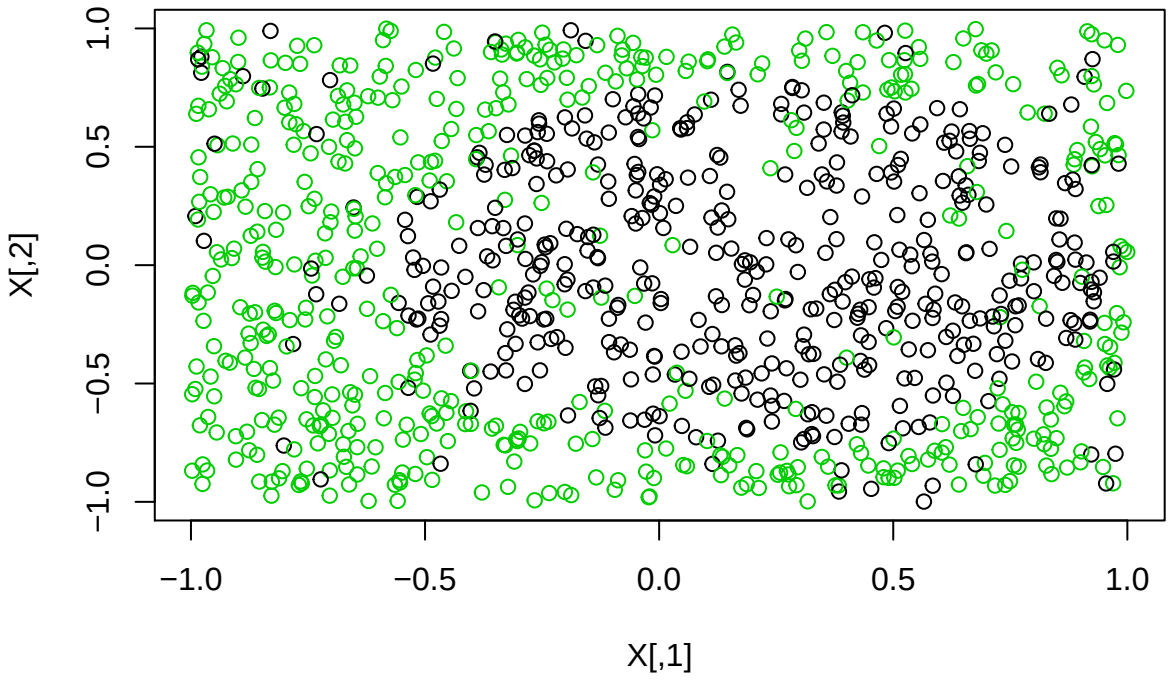
a) Generar muestra de entrenamiento de N=1000 puntos en el cuadrado $X = [-1, 1]x[-1, 1]$.

```
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
}

set.seed(1)

X <- simula_unif(1000,2,c(-1,1))
plot(X)
```



b) Aplicar la función $f(x_1, x_2) = sign((x_1 - 0,2)2 + x_2^2 - 0,6)$ a los puntos con un error del 10%

```
f <- function(x1,x2){
  sign( (x1-0.2)^2 + x2^2 - 0.6 )
}

Y <- matrix(f(X[,1],X[,2]))

noise <- function(label, p){
  result <- label*sample(c(1, -1), size=length(label), replace=TRUE, prob= c(1-p,p))
  result
}

Y <- noise(Y,0.1)

plot(X, col=Y+2)
```

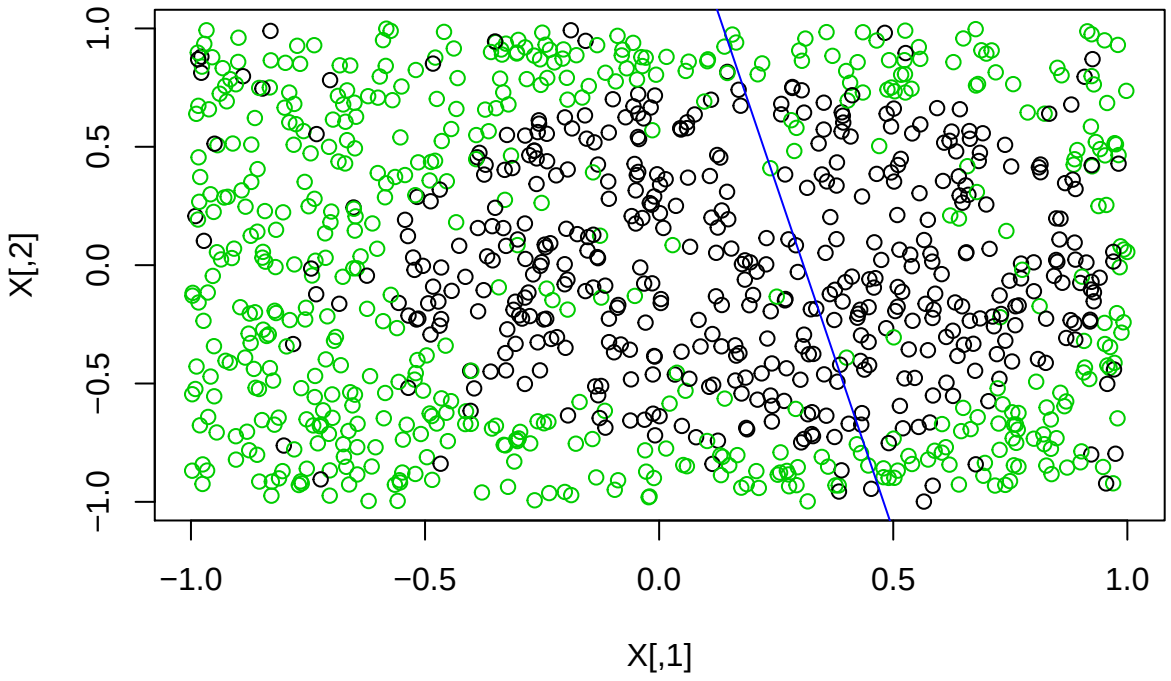


c) Ajustar un modelo de regresion lineal al conjunto de datos generado y estimar los pesos w. Estimar E_{in} usando SGD

```
w <- SGD(X,Y,seed=1)

Yprima <- Clasificar(X,w)

Ein <- Err(Yprima,Y,binary_error)

Ein
```

```
## [1] 0.424
```

```
#Obtención recta
R <- pasoARecta(w)

# Obtención de puntos
x_1=-1
y_1 = recta(x_1,R[1],R[2])
x_2= 1
y_2= recta(x_2,R[1],R[2])

# Dibujo datosTr y ambas rectas (Pseudo = AZUL)
plot(X, col=Y+2)
lines(c(x_1,x_2),c(y_1,y_2), col="blue",type="l")
```



d) Ejecutar todo el experimento definido por (a)-(c) 1000 veces. Generando 1000 puntos nuevos por cada iteración para calcular con ellos el valor de E_{out}. Calcular el valor medio de E_{in} y E_{out}

```
vectorEi=0
vectorEo=0

set.seed(1)
for(i in 1:1000){
  X <- simula_unif(1000,2,c(-1,1))
  Y <- matrix(f(X[,1],X[,2]))
  Y <- noise(Y,0.1)

  w <- SGD(X,Y,seed=1,max_i = 100)
  Yprima <- Clasificar(X,w)
  vectorEi[i] <- Err(Yprima,Y,binary_error)

  X <- simula_unif(1000,2,c(-1,1))
  Y <- matrix(f(X[,1],X[,2]))
  Yprima <- Clasificar(X,w)
  vectorEo[i] <- Err(Yprima,Y,binary_error)

}

ValorMedio_Ein = sum(vectorEi)/length(vectorEi)
ValorMedio_Eout = sum(vectorEo)/length(vectorEo)

ValorMedio_Ein
```

```
## [1] 0.423985
```

```
ValorMedio_Eout
```

```
## [1] 0.411399
```

e) Valore que tan bueno considera que es el ajuste con este modelo lineal a la vista de los valores medios obtenidos de E_{in} y E_{out}

El ajuste realizado por la clase de funciones lineales no es muy adecuado, con un error bastante elevado en ambos casos. Esto se debe a la disposición de los puntos, imposibles de dividir de forma significativa por un hiperplano.