EJERCICIOS GRADIENTE

Manuel Carmona

01 de diciembre de 2017

```
setwd("C:/Users/Manuel/Desktop/CUNEF/MACHINE LEARNING/clase04/")
datos<- read.csv("4_1_data.csv" , header = TRUE)
```

1- Análisis exploratorio

```
summary(datos)
```

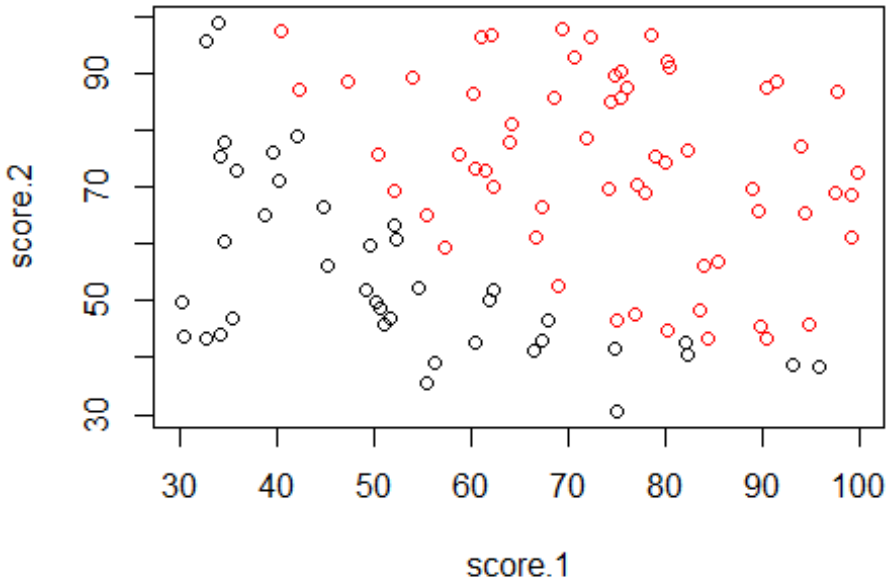
```
##     score.1         score.2          label
##  Min.   :30.06   Min.   :30.60   Min.   :0.0
##  1st Qu.:50.92   1st Qu.:48.18   1st Qu.:0.0
##  Median :67.03   Median :67.68   Median :1.0
##  Mean   :65.64   Mean   :66.22   Mean   :0.6
##  3rd Qu.:80.21   3rd Qu.:79.36   3rd Qu.:1.0
##  Max.   :99.83   Max.   :98.87   Max.   :1.0
```

```
colSums(is.na(datos))
```

```
## score.1 score.2   label
##       0       0       0
```

Graficamos los datos

```
plot(datos$score.1, datos$score.2, col = as.factor(datos$label), xlab =
"score.1", ylab = "score.2")
```



Dataset de entrenamiento y test

```
# tomamos una muestra de training y otra de test
set.seed(123)
n = nrow(datos)
id_train <- sample(1:n, 0.80*n)  # Asignamos el 80 % al entrenamiento.
datos.train <- datos[id_train,]
datos.test <- datos[-id_train,]
```

Creo las variables X e Y de entrenamiento y test.

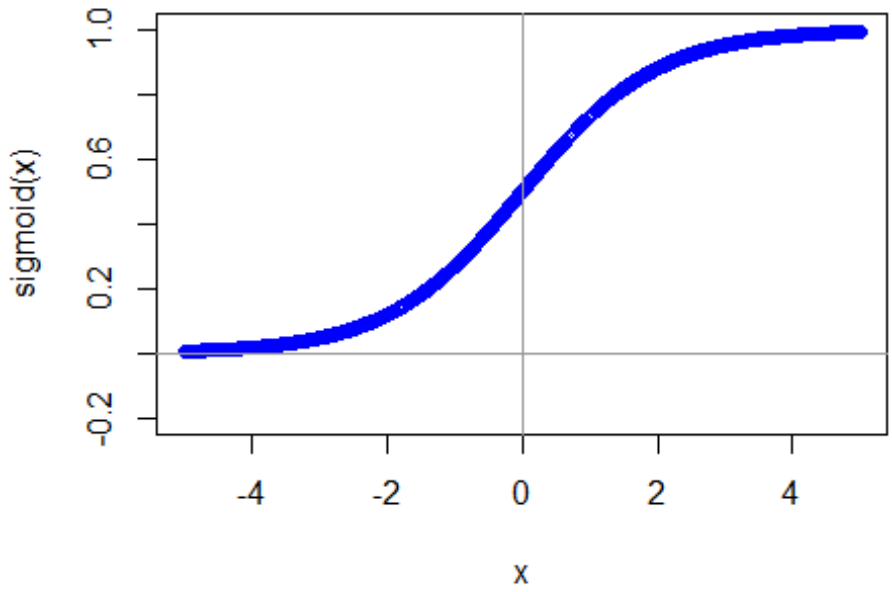
```
# Matrices de Train

Xtrain <- as.matrix(datos.train[,c(1,2)])
Xtrain <- cbind(rep(1,nrow(Xtrain)),Xtrain)
Ytrain <- as.matrix(datos.train$label)

# Matrices de Test

Xtest <- as.matrix(datos.test[,c(1,2)])
Xtest <- cbind(rep(1,nrow(Xtest)),Xtest)
Ytest <- as.matrix(datos.test$label)
```

3- Sigmoide

Defino la funcion (Pi/(1-Pi)) y las matrices Xi e Yi de mi modelo logit. Aun engo que estimar las Betas.

La ecuación de una regresión logística es ln(Pi/(1-Pi))=beta1+beta1*x1+...+u. Por tanto, si queremos saber pi, hay que despejar la función haciendo: pi=1/(1+exp(-z))

```
# Ecuación de la sigmoide.

sigmoid<- function(x){
         1/(1+exp(-x))
}

# grafico de la sigmoide

 x<- seq(-5,5,0.01)
 plot(x,sigmoid(x), col="blue", ylim= c(-.2,1))
 abline(h=0, v=0, col = "gray60")
```



4- Funcion de costes

La función de costes es nuestra función objetivo, es decir la función que queremos minimizar reduciendo su coste.En éste caso, el coste representa los errores que se cometen al estimar la variable dependiente y (label). Para ello, se deberán de estimar

los parámetros adecuados para que ésta función tome el mínimo valor. En la función de regresión los parámetros representan los betas por los que se multiplica el valor de cada observación de las variables explicativas, en este caso, score.1 y score.2:

```
funcionCostes <- function(parametros, x, y) {

  n <- nrow(x)
  g <- sigmoid(x %*% parametros)  # %*% este simbolo se define como
multiplicaci?n de matrices
  j <-  ((-1)/n)*sum((y*log(g))+(1-y)*log(1-g))
}
```

5- Inicio los parametros y calculo el coste inicial con estos parámetros

Si se toma como parámetros iniciales (0,0,0), el cote inicial de la función es 0.69, el objetivo es reducir éste porcentaje de coste.

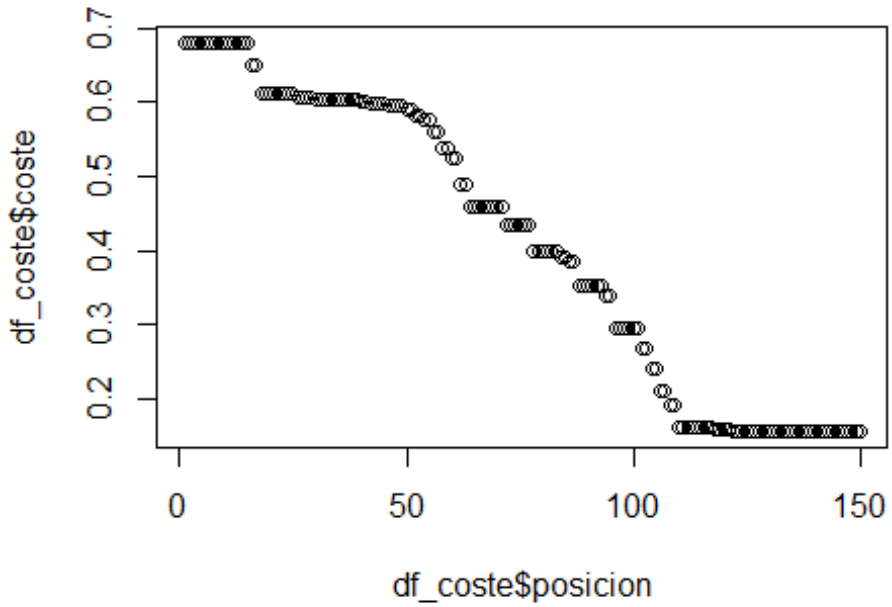
```
initial_parameters <- rep(0, ncol(Xtrain))
# Hago una columna de 0 para el término independiente. La b quiero
tratarla como una w. La b no está multiplicada por X en la ecuación. Para
hacer producto matricial hago esto. Simplifico las operaciones y lo uno
todo en un producto de matrices.

funcionCostes(initial_parameters, Xtrain, Ytrain)
# El error que me saldría si fijo los parámetros a 0. Estás acertando que
no entre nadie a la universidad.

print(paste("El coste inicial de la funcion es: ",
              convergence <- c(funcionCostes(initial_parameters, Xtrain,
Ytrain)), sep = ""))

## [1] "El coste inicial de la funcion es: 0.693147180559945"
```

6- Descenso del gradiente.

Gráfica nºiter/coste

```
# vamos a crear una funcion para obtener un mapa de puntos de las
iteraciones y poder representar como influyen el numero de iteraciones en
la funcion de costes y en el numero optimo de parametros
GraficaCosteIteraciones=function(iteraciones){
        posicion <- NULL
        coste <- NULL
        contador <- 0
        for(i in (1: iteraciones)) {
                contador <- contador + 1
                parametros_optimizados <- optim(par = initial_parameters,
```

```
fn = funcionCostes, x = Xtrain, y = Ytrain, control = list(maxit =
contador))
                parametros <- parametros_optimizados$par
                posicion <- c(posicion,i)
                coste <- c(coste,funcionCostes(parametros,Xtrain,Ytrain))
        }
        df_coste <- data.frame(posicion,coste)

        print(plot(df_coste$posicion,df_coste$coste))

        return(df_coste)
}

resultados <- GraficaCosteIteraciones(150)
```



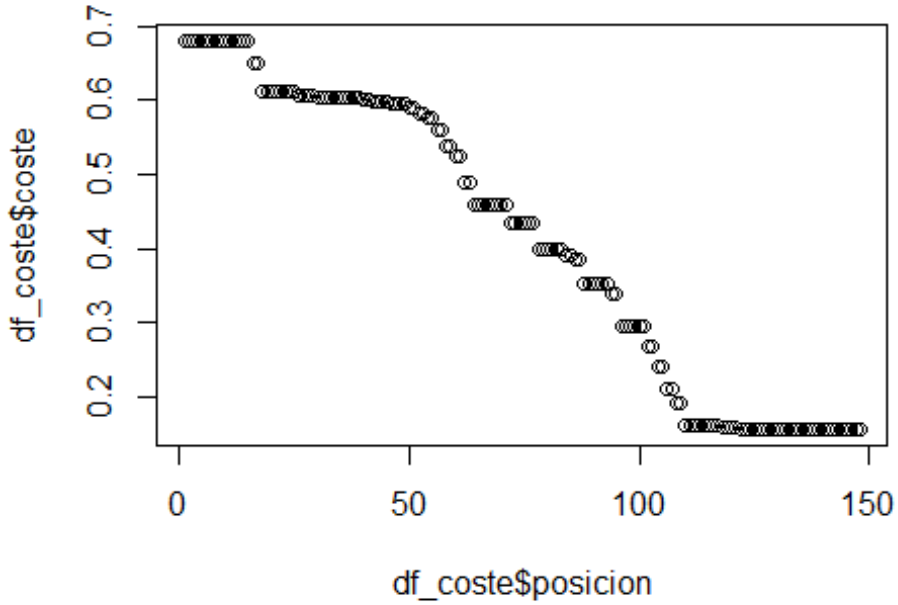

```
## NULL

df_iter_coste <- data.frame(resultados$posicion,resultados$coste)
```

Podemos ver que el numero de iteraciones que minimiza el coste es 148

```
df_iter_coste[df_iter_coste$resultados.coste ==
min(df_iter_coste$resultados.coste),]

##     resultados.posicion resultados.coste
## 148                 148        0.1565738
## 149                 149        0.1565738
## 150                 150        0.1565738
```

EJERCICIO2

```
GraficaCosteIteraciones(148)
```




```
## NULL

##     posicion     coste
## 1          1 0.6806467
## 2          2 0.6806467
## 3          3 0.6806467
## 4          4 0.6806467
## 5          5 0.6806467
## 6          6 0.6806467
## 7          7 0.6806467
## 8          8 0.6806467
## 9          9 0.6806467
## 10        10 0.6806467
## 11        11 0.6806467
## 12        12 0.6806467
## 13        13 0.6806467
## 14        14 0.6806467
## 15        15 0.6806467
## 16        16 0.6513365
## 17        17 0.6513365
## 18        18 0.6122224
## 19        19 0.6122224
## 20        20 0.6122224
```

```
## 21        21 0.6122224
## 22        22 0.6122224
## 23        23 0.6122224
## 24        24 0.6122224
## 25        25 0.6122224
## 26        26 0.6080095
## 27        27 0.6080095
## 28        28 0.6057034
## 29        29 0.6057034
## 30        30 0.6047512
## 31        31 0.6047512
## 32        32 0.6040795
## 33        33 0.6040795
## 34        34 0.6034974
## 35        35 0.6034974
## 36        36 0.6034660
## 37        37 0.6034660
## 38        38 0.6029176
## 39        39 0.6029176
## 40        40 0.6007062
## 41        41 0.6007062
## 42        42 0.5996258
## 43        43 0.5996258
## 44        44 0.5996258
## 45        45 0.5996258
## 46        46 0.5949896
## 47        47 0.5949896
## 48        48 0.5949896
## 49        49 0.5949896
## 50        50 0.5900909
## 51        51 0.5900909
## 52        52 0.5826476
## 53        53 0.5826476
## 54        54 0.5758320
## 55        55 0.5758320
## 56        56 0.5601036
## 57        57 0.5601036
## 58        58 0.5386558
## 59        59 0.5386558
## 60        60 0.5237450
## 61        61 0.5237450
## 62        62 0.4902556
## 63        63 0.4902556
## 64        64 0.4591889
## 65        65 0.4591889
## 66        66 0.4591889
## 67        67 0.4591889
## 68        68 0.4591889
## 69        69 0.4591889
## 70        70 0.4591889
```

```
## 71         71 0.4591889
## 72         72 0.4353760
## 73         73 0.4353760
## 74         74 0.4353760
## 75         75 0.4353760
## 76         76 0.4353760
## 77         77 0.4353760
## 78         78 0.3984260
## 79         79 0.3984260
## 80         80 0.3984260
## 81         81 0.3984260
## 82         82 0.3984260
## 83         83 0.3984260
## 84         84 0.3908706
## 85         85 0.3908706
## 86         86 0.3859505
## 87         87 0.3859505
## 88         88 0.3522976
## 89         89 0.3522976
## 90         90 0.3522976
## 91         91 0.3522976
## 92         92 0.3522976
## 93         93 0.3522976
## 94         94 0.3380975
## 95         95 0.3380975
## 96         96 0.2956420
## 97         97 0.2956420
## 98         98 0.2956420
## 99         99 0.2956420
## 100       100 0.2956420
## 101       101 0.2956420
## 102       102 0.2694402
## 103       103 0.2694402
## 104       104 0.2407056
## 105       105 0.2407056
## 106       106 0.2111299
## 107       107 0.2111299
## 108       108 0.1907474
## 109       109 0.1907474
## 110       110 0.1626427
## 111       111 0.1626427
## 112       112 0.1626427
## 113       113 0.1626427
## 114       114 0.1626427
## 115       115 0.1626427
## 116       116 0.1626427
## 117       117 0.1626427
## 118       118 0.1585326
## 119       119 0.1585326
## 120       120 0.1585326
```

```
## 121      121 0.1585326
## 122      122 0.1568378
## 123      123 0.1568378
## 124      124 0.1568378
## 125      125 0.1568378
## 126      126 0.1568378
## 127      127 0.1568378
## 128      128 0.1568378
## 129      129 0.1568378
## 130      130 0.1568378
## 131      131 0.1568378
## 132      132 0.1567751
## 133      133 0.1567751
## 134      134 0.1567197
## 135      135 0.1567197
## 136      136 0.1566327
## 137      137 0.1566327
## 138      138 0.1566327
## 139      139 0.1566327
## 140      140 0.1566033
## 141      141 0.1566033
## 142      142 0.1566033
## 143      143 0.1566033
## 144      144 0.1565941
## 145      145 0.1565941
## 146      146 0.1565811
## 147      147 0.1565811
## 148      148 0.1565738
```

Creamos la funcion TestGradiente

Incluimos las definiciones anteriores en una funcion que facilite el trabajo.

```
TestGradiente <- function(iteraciones, x, y){

        parametros_optimizados <- optim(par = initial_parameters, fn = funcionCostes,
                                        x = Xtrain, y = Ytrain, control = list(maxit = iteraciones))
        print(parametros_optimizados)

        parametros <- parametros_optimizados$par

        print(paste("Final Cost Function value: ",
                    convergence <- c(funcionCostes(parametros, x, y)), sep = ""))

        return(parametros )

}
```

```
TestGradiente(150, Xtrain, Ytrain)
```

```
## $par
## [1] -31.5061345   0.2350493   0.2826581
##
## $value
## [1] 0.1565738
##
## $counts
## function gradient
##      152       NA
##
## $convergence
## [1] 1
##
## $message
## NULL
##
## [1] "Final Cost Function value: 0.156573776933154"
```

```
## [1] -31.5061345   0.2350493   0.2826581
```

7 - Comprobamos que los resultados son correctos

Llamamos parametros a los parámetros óptimos.

```
# install.packages("testthat")
library(testthat)

parametros <- TestGradiente(150,x = Xtrain, y = Ytrain)
```

```
## $par
## [1] -31.5061345   0.2350493   0.2826581
##
## $value
## [1] 0.1565738
##
## $counts
## function gradient
##      152       NA
##
## $convergence
## [1] 1
##
## $message
## NULL
##
## [1] "Final Cost Function value: 0.156573776933154"
```

```r
# probability of admission for student (1 = b, for the calculos)
new_student <- c(1,25,78)
print("Probability of admission for student:")
```

```
## [1] "Probability of admission for student:"
```

```r
print(prob_new_student <- sigmoid(t(new_student) %*% parametros))
```

```
##             [,1]
## [1,] 0.02705205
```

```r
test_that("Test TestGradiente",{
  parametros <- TestGradiente(150, x = Xtrain, y = Ytrain)

  new_student <- c(1,25,78)
  prob_new_student <- sigmoid(t(new_student) %*% parametros)
  print(prob_new_student)
  expect_equal(as.numeric(round(prob_new_student, digits = 8)),
0.02705205)

})
```

```
## $par
## [1] -31.5061345   0.2350493   0.2826581
##
## $value
## [1] 0.1565738
##
## $counts
## function gradient
##      152       NA
##
## $convergence
## [1] 1
##
## $message
## NULL
##
## [1] "Final Cost Function value: 0.156573776933154"
##             [,1]
## [1,] 0.02705205
```

8-Matriz de confusión y % acierto

En primer lugar debemos predecir con la muestra de testing como paso previo a hacer la matriz de confusión.

```r
prediccion_prob <- sigmoid(Xtest %*% parametros)
# probabilidades
prediccion_prob<-data.frame(prediccion_prob)
```

Creamos una funcion que calcule la matriz de confusion:

```
#definimos una funcion para crearla en funcion de prediccion_prob. Pongo
que me devuelva tambien el acurracy
matrizConfusion = function(prediccion_prob){
        resultado1 <- NULL
        for (i in (1:nrow(prediccion_prob))){
                ifelse(1- prediccion_prob[i,1] < 0.5, resultado1 <-
c(resultado1,1),resultado1 <- c(resultado1,0))
        }
        prediccion_prob[,"resultado"] <- resultado1
        prediccion_test <- prediccion_prob[,2]
        accuracy<-100*sum(diag(table(Ytest,
prediccion_prob$prediccion_prob)))/sum(table(Ytest,
prediccion_prob$prediccion_prob))

        print(paste("El accuracy es: ", accuracy))

        return(table(Ytest,prediccion_test))
}
matrizConfusion(prediccion_prob)

## [1] "El accuracy es:  5"

##      prediccion_test
## Ytest 0 1
##     0 9 2
##     1 1 8
```

Una forma alternativa de calcular la matriz:

```
prediccion_prob[abs(prediccion_prob-1)<abs(prediccion_prob-0)]=1
prediccion_prob[abs(prediccion_prob-1)>=abs(prediccion_prob-0)]=0
table(Ytest, prediccion_prob$prediccion_prob)

##
## Ytest 0 1
##     0 9 2
##     1 1 8
```

Comprobamos la estabilidad del modelo

La pregunta que nos realizamos ahora es: ¿Es estable el modelo? Para responderla vamos a cambiar las muestras de training y test y compararemos resultados. La inestabilidad se da cuando hay un porcentaje significativo de diferencia entre lo que dice un modelo y lo que dice otro. No existe un limite concreto que defina ese porcentaje, depende del contexto.

Dataset de entrenamiento y test

Cambiamos la semilla para ver si el modelo es estable

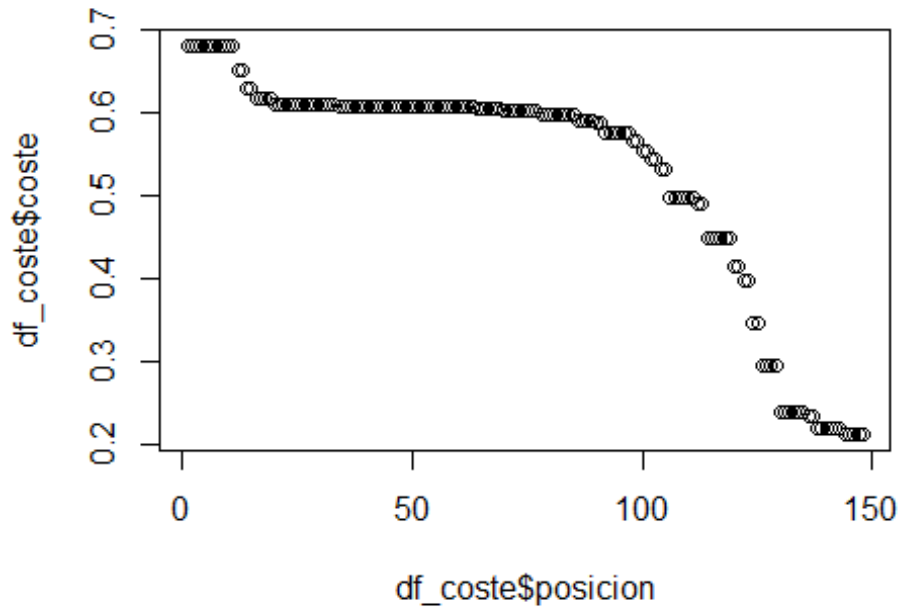
```r
# tomamos una muestra de training y otra de test
set.seed(456)
n = nrow(datos)
id_train <- sample(1:n, 0.80*n)  # Asignamos el 80 % al entrenamiento.
datos.train <- datos[id_train,]
datos.test <- datos[-id_train,]
```

Creo las variables X e Y de entrenamiento y test.

```r
# Matrices de Train

Xtrain <- as.matrix(datos.train[,c(1,2)])
Xtrain <- cbind(rep(1,nrow(Xtrain)),Xtrain)
Ytrain <- as.matrix(datos.train$label)

# Matrices de Test

Xtest <- as.matrix(datos.test[,c(1,2)])
Xtest <- cbind(rep(1,nrow(Xtest)),Xtest)
Ytest <- as.matrix(datos.test$label)
```

Vemos la grafica de nuevo

```r
GraficaCosteIteraciones(148)
```




```
## NULL
```

```
##     posicion     coste
## 1          1 0.6818967
## 2          2 0.6818967
## 3          3 0.6818967
## 4          4 0.6818967
## 5          5 0.6818967
## 6          6 0.6818967
## 7          7 0.6818967
## 8          8 0.6818967
## 9          9 0.6818967
## 10        10 0.6818967
## 11        11 0.6818967
## 12        12 0.6533920
## 13        13 0.6533920
## 14        14 0.6302648
## 15        15 0.6302648
## 16        16 0.6191973
## 17        17 0.6191973
## 18        18 0.6191973
## 19        19 0.6191973
## 20        20 0.6115639
## 21        21 0.6115639
## 22        22 0.6115639
## 23        23 0.6115639
## 24        24 0.6115639
## 25        25 0.6115639
## 26        26 0.6115639
## 27        27 0.6115639
## 28        28 0.6115490
## 29        29 0.6115490
## 30        30 0.6103453
## 31        31 0.6103453
## 32        32 0.6099182
## 33        33 0.6099182
## 34        34 0.6094318
## 35        35 0.6094318
## 36        36 0.6094118
## 37        37 0.6094118
## 38        38 0.6087551
## 39        39 0.6087551
## 40        40 0.6086016
## 41        41 0.6086016
## 42        42 0.6083616
## 43        43 0.6083616
## 44        44 0.6083616
## 45        45 0.6083616
## 46        46 0.6081381
## 47        47 0.6081381
## 48        48 0.6080387
## 49        49 0.6080387
```

```
## 50        50 0.6079197
## 51        51 0.6079197
## 52        52 0.6079197
## 53        53 0.6079197
## 54        54 0.6079056
## 55        55 0.6079056
## 56        56 0.6079056
## 57        57 0.6079056
## 58        58 0.6076713
## 59        59 0.6076713
## 60        60 0.6075724
## 61        61 0.6075724
## 62        62 0.6074108
## 63        63 0.6074108
## 64        64 0.6067288
## 65        65 0.6067288
## 66        66 0.6067288
## 67        67 0.6067288
## 68        68 0.6062456
## 69        69 0.6062456
## 70        70 0.6046231
## 71        71 0.6046231
## 72        72 0.6046231
## 73        73 0.6046231
## 74        74 0.6042642
## 75        75 0.6042642
## 76        76 0.6025134
## 77        77 0.6025134
## 78        78 0.5995382
## 79        79 0.5995382
## 80        80 0.5995382
## 81        81 0.5995382
## 82        82 0.5995382
## 83        83 0.5995382
## 84        84 0.5974742
## 85        85 0.5974742
## 86        86 0.5918102
## 87        87 0.5918102
## 88        88 0.5918102
## 89        89 0.5918102
## 90        90 0.5897307
## 91        91 0.5897307
## 92        92 0.5771632
## 93        93 0.5771632
## 94        94 0.5771632
## 95        95 0.5771632
## 96        96 0.5771632
## 97        97 0.5771632
## 98        98 0.5660052
## 99        99 0.5660052
```

```
## 100      100 0.5556367
## 101      101 0.5556367
## 102      102 0.5453129
## 103      103 0.5453129
## 104      104 0.5335756
## 105      105 0.5335756
## 106      106 0.4987912
## 107      107 0.4987912
## 108      108 0.4987912
## 109      109 0.4987912
## 110      110 0.4987912
## 111      111 0.4987912
## 112      112 0.4898658
## 113      113 0.4898658
## 114      114 0.4498773
## 115      115 0.4498773
## 116      116 0.4498773
## 117      117 0.4498773
## 118      118 0.4498773
## 119      119 0.4498773
## 120      120 0.4147346
## 121      121 0.4147346
## 122      122 0.3983670
## 123      123 0.3983670
## 124      124 0.3461576
## 125      125 0.3461576
## 126      126 0.2944135
## 127      127 0.2944135
## 128      128 0.2944135
## 129      129 0.2944135
## 130      130 0.2403346
## 131      131 0.2403346
## 132      132 0.2403346
## 133      133 0.2403346
## 134      134 0.2403346
## 135      135 0.2403346
## 136      136 0.2352497
## 137      137 0.2352497
## 138      138 0.2187477
## 139      139 0.2187477
## 140      140 0.2187477
## 141      141 0.2187477
## 142      142 0.2187477
## 143      143 0.2187477
## 144      144 0.2126700
## 145      145 0.2126700
## 146      146 0.2126700
## 147      147 0.2126700
## 148      148 0.2126700
```

Como vemos, la gráfica cambia. Tenemos que comprobar si la matriz de confusion (output del modelo) cambia de forma significativa o no.

Aplicamos la función TestGradiente para comprobarlo:

```
TestGradiente(150, Xtrain, Ytrain)
```

```
## $par
## [1] -26.0078373   0.2208838   0.2047452
##
## $value
## [1] 0.21267
##
## $counts
## function gradient
##      152       NA
##
## $convergence
## [1] 1
##
## $message
## NULL
##
## [1] "Final Cost Function value: 0.212670016607215"
```

```
## [1] -26.0078373   0.2208838   0.2047452
```

```
parametros <- TestGradiente(150, x = Xtrain, y = Ytrain)
```

```
## $par
## [1] -26.0078373   0.2208838   0.2047452
##
## $value
## [1] 0.21267
##
## $counts
## function gradient
##      152       NA
##
## $convergence
## [1] 1
##
## $message
## NULL
##
## [1] "Final Cost Function value: 0.212670016607215"
```

```
# probability of admission for student (1 = b, for the calculos)
new_student <- c(1,25,78)
print("Probability of admission for student:")
```

```
## [1] "Probability of admission for student:"
```

```
print(prob_new_student <- sigmoid(t(new_student) %*% parametros))
```

```
##            [,1]
## [1,] 0.01081858
```

```
test_that("Test TestGradiente",{
  parametros <- TestGradiente(150, x = Xtrain, y = Ytrain)

  new_student <- c(1,25,78)
  prob_new_student <- sigmoid(t(new_student) %*% parametros)
  print(prob_new_student)
  expect_equal(as.numeric(round(prob_new_student, digits = 8)),
0.01081858)

})
```

```
## $par
## [1] -26.0078373   0.2208838   0.2047452
##
## $value
## [1] 0.21267
##
## $counts
## function gradient
##      152       NA
##
## $convergence
## [1] 1
##
## $message
## NULL
##
## [1] "Final Cost Function value: 0.212670016607215"
##            [,1]
## [1,] 0.01081858
```

```
prediccion_prob <- sigmoid(Xtest %*% parametros)
# probabilidades
prediccion_prob<-data.frame(prediccion_prob)

matrizConfusion(prediccion_prob)
```

```
## [1] "El accuracy es:  5"
```

```
##      prediccion_test
## Ytest  0  1
##     0  9  1
##     1  0 10
```

El modelo no es inestable ya que no se da un cambio significativo en el ouput.