Práctica 3: Modelos de Regresión Lineal con R

Andrés Campos Cuiña

16/10/2021

Índice

1. Carga de las librerías necesarias

Primero cargamos las librerías necesarias:

```
library(tidyverse)
library(lattice)
```

2. Ajuste de los parámetros de un modelo de regresión lineal mediante métodos de optimización

2.1. Método de descenso de gradiente

Para simplificar el proceso consideraremos ahora un modelo de regresión lineal simple, como el siguiente:

$$Y = \beta_0 + \beta_1 X_1 + \epsilon$$

En primer lugar, vamos a simular una muestra con $n = 100$ observaciones, (x_i, y_i), de un modelo de regresión lineal simple con parámetros β_0 y β_1. Para ello, generamos en primer lugar los valores x_i a partir de una distribución uniforme. A continuación generamos los errores del modelo, ϵ_i, a partir de una distribución normal de media 0 y varianza σ^2. Los valores y_i se calcularán entonces como:

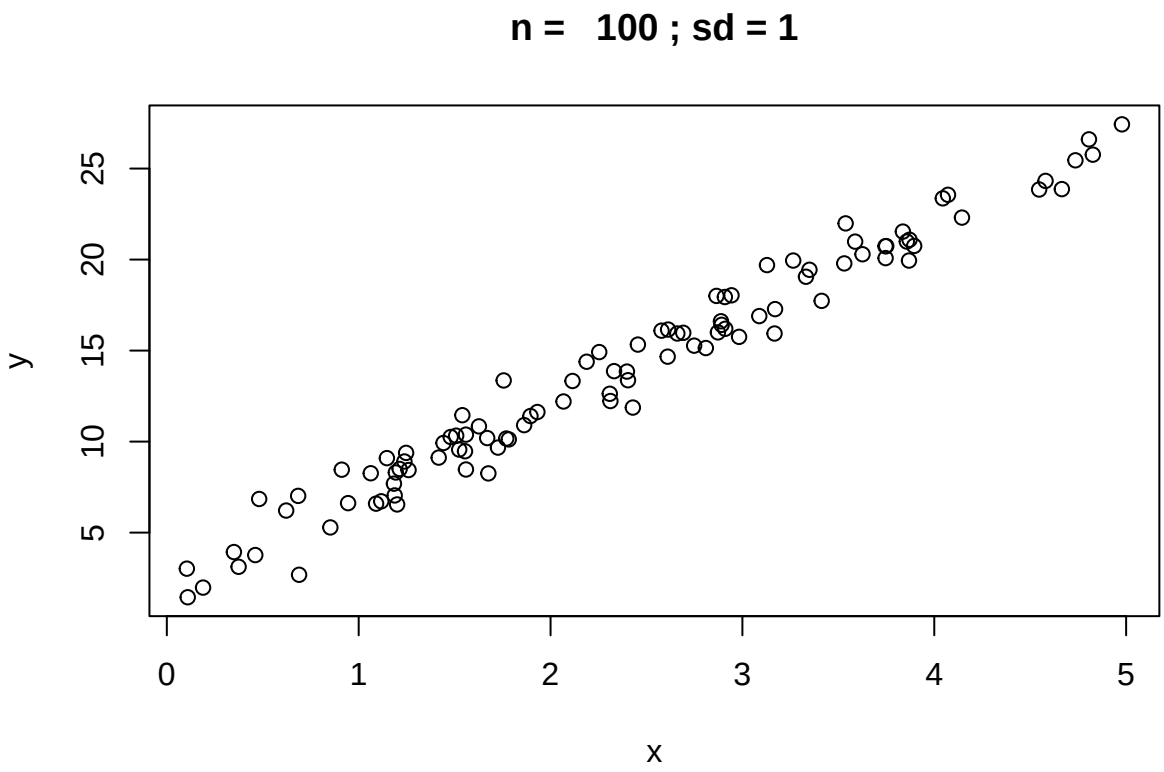
$$y_i = \beta_0 + \beta_1 x_i + \epsilon_i$$

En R, por lo tanto, empezamos generando nuestra muestra de 100 observaciones:

```
n <- 100

# x_i: n puntos aleatorios en el itervalo [min,max]
x <- runif(n, min = 0, max = 5)

# Parámetro beta0 del modelo
beta0 <- 2

 # Parámetro beta1 del modelo
beta1 <- 5

# Error (con desviación típica sd=1)
epsilon <- rnorm(n, sd = 1)

# Calculamos y
y <- beta0 + beta1 * x + epsilon
```

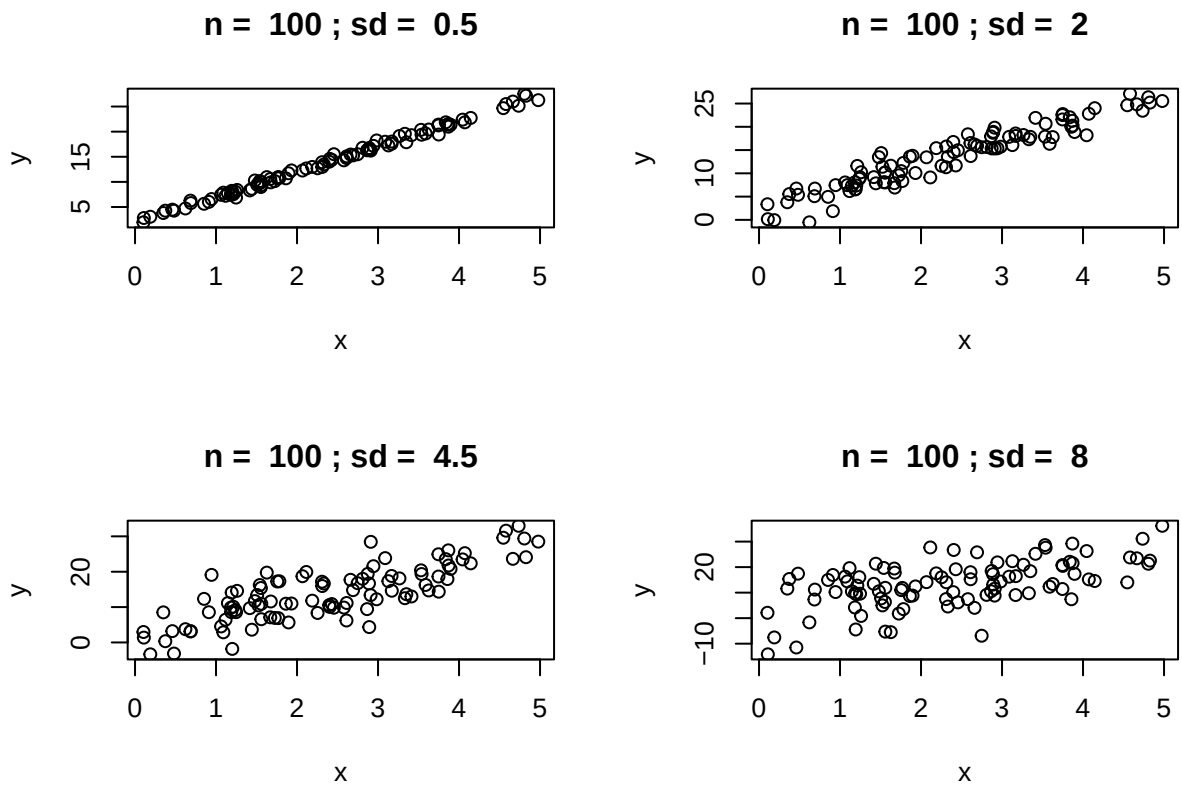
En primer lugar haremos un diagrama de dispersión de la muestra de entrenamiento generada:

```
plot(x, y, main = paste('n =  ', n, '; sd = 1'))
```



Podemos representar este diagrama para diferentes valores del parámetro `sd`:

```
# Para plotear en un grid las diferentes gráficas
par(mfrow=c(2,2))

for (i in 1:4) {
  # Definimos sd
  sd = 0.5 * i^2

  # Error (con desviación típica sd=1)
  epsilon <- rnorm(n=n, sd=sd)

  # Calculamos y
  y <- beta0 + beta1 * x + epsilon

  # Ploteamos
  plot(x, y, main = paste('n = ', n, '; sd = ', sd))
}
```



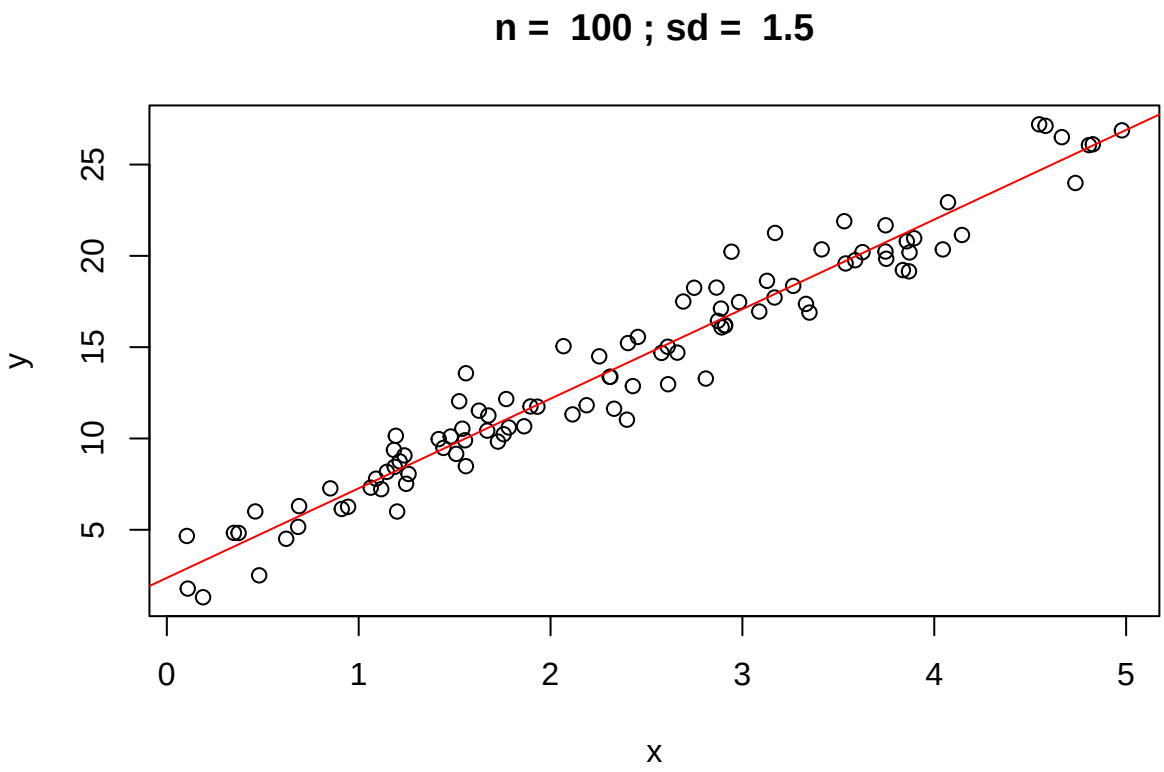
Como se puede observar, cuanto mayor es el valor de σ^2, `sd`, mayor es el valor de cada uno de los ϵ_i y por lo tanto también hay menos correlación entre la variable `y` y la variable `x`.

Ahora calcularemos el valor de los parámetros estimados (β_0 y β_1) mediante la función `lm`:

```
# Definimos sd
sd = 1.5

# Error (con desviación típica sd=1)
epsilon <- rnorm(n = n, sd = sd)

# Calculamos y
y <- beta0 + beta1 * x + epsilon

# Calculamos el modelo re regresión mediante `lm`
z <- lm(y ~ x)
```

Visualizamos el resultado obtenido mediante la función `lm`:

```
plot(x, y, main = paste('n = ', n, '; sd = ', sd))

# Añadimos la recta de regresión
abline(z, col = "red")
```



Resumimos los resultados del ajuste mediante la función `summary`:

```
summary(z)
```

```
##
## Call:
## lm(formula = y ~ x)
##
## Residuals:
##     Min      1Q  Median      3Q     Max
## -3.1010 -0.7685 -0.1132  0.7268  3.5570
##
## Coefficients:
##             Estimate Std. Error t value Pr(>|t|)
## (Intercept)   2.3670     0.2973   7.962 3.05e-12 ***
## x             4.9053     0.1117  43.908  < 2e-16 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 1.382 on 98 degrees of freedom
## Multiple R-squared:  0.9516, Adjusted R-squared:  0.9511
## F-statistic:  1928 on 1 and 98 DF,  p-value: < 2.2e-16
```

Obtenemos los parámetros estimados por el modelo mediante la función `coef`:

```
coef(z)
```

```
## (Intercept)           x
##    2.366953    4.905340
```

Ahora aproximaremos estos parámetros mediante la el método del descenso de gradiente. Para ello hay que minimizar la siguiente función:

$$J(\beta_0, \beta_1) = \frac{1}{2}\sum_{i=1}^{n}(y_i - \beta_0 + \beta_1 x_i)^2$$

Por lo tanto, programamos el método del descenso de gradiente para minimizar esta función. Lo primero será definir la función `f` que vamos a minimizar, la función `mse` que usaremos para calcular el error cuadrático medio y la función `coste` que usaremos para calcular el MSE para nuestro valor en cada iteración para β_0 y β_1.

```
# Función a modelar
f <- function(x, beta0, beta1) {
  return(beta0 + beta1 * x)
}

# RSE (Residual Standard Error)
rse <- function(reales, prediccion) {
  n <- length(reales)
  rss <- sum((prediccion - reales) ^ 2)
  rse <- sqrt((rss / (n - 2)))
  return(rse)
}

# Función de coste
coste <- function(x, y, beta0, beta1) {
  prediccion <- f(x, beta0, beta1)
  rse <- rse(y, prediccion)
  return(rse)
}
```

Ahora implementaremos el método del descenso de gradiente:

```
# Función del descenso de gradiente
desc_grad <- function(x, y, t, threshold,  max_iter) {
  # Inicializamos los coeficientes a 0
  beta0 <- 0
  beta1 <- 0

  converged = FALSE
  iterations = 0

  while (converged == FALSE) {
    # Obtenemos la prediccion
    prediccion <- f(x, beta0, beta1)

    # Calculamos el gradiente de f(x)
    grad_beta0 = sum(y - prediccion)
    grad_beta1 = sum(x * (y - prediccion))

    # Actualizamos los valores de los parámetros beta0 y beta1
    beta0 <- beta0 + t * grad_beta0
    beta1 <- beta1 + t * grad_beta1

    # Calculamos el rse cometido con la función de coste
```

```
    coste <- coste(x, y, beta0, beta1)

    # Obtenemos el criterio de parada
    stopping_criterion = grad_beta0^2 + grad_beta1^2

    # Comprobamos si se cumple el criterio de parada
    if (stopping_criterion <= threshold) {
      converged = T
      print(paste("Optimal beta0: ",beta0))
      print(paste("Optimal beta1: ",beta1))
      print(paste("RSE: ", coste))
      print(paste("Converged in", iterations, "iterations"))
    }

    # Comprobamos si se han ejecutado el número máximo de iteraciones
    iterations = iterations + 1
    if (iterations > max_iter) {
      converged = TRUE
      print(paste("Optimal beta0: ",beta0))
      print(paste("Optimal beta1: ",beta1))
      print(paste("RSE: ", coste))
      print(paste("Not Converged"))
    }
  }
  return(list(beta0, beta1, coste))
}
```

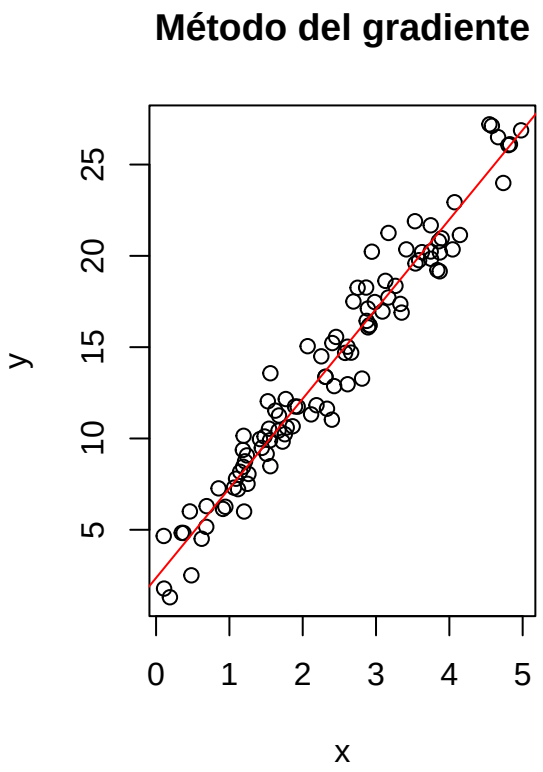
Ejecutamos nuestra implementación del descenso de gradiente, con un paso de 0,001 y un número de iteraciones máximas igual a 1000. Consideraremos que un valor para el criterio de parada, para el que usaremos $||\nabla f(x)||^2$, inferior a 0,0001 es suficientemente correcto:

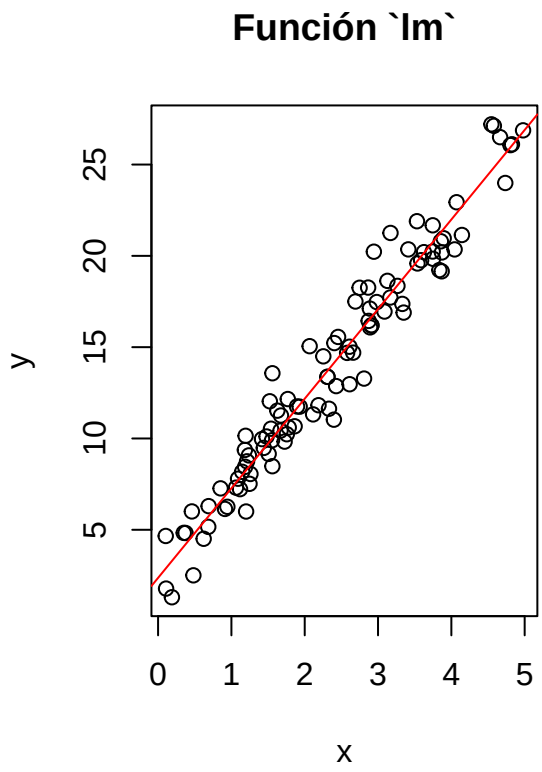
```
result <- desc_grad(x, y, t = 0.001, threshold = 0.0001, max_iter = 1000)
```

```
## [1] "Optimal beta0:  2.36647888789399"
## [1] "Optimal beta1:  4.90550212651836"
## [1] "RSE:  1.38201462785111"
## [1] "Converged in 365 iterations"
```

Compararemos mediante dos gráficas la recta de regresión obtenida por nuestro modelo y la obtenida mediante el uso de la función `lm`:

```
par(mfrow = c(1, 2))

plot(x, y, main = "Método del gradiente")
lines(c(-10:10), f(c(-10:10), result[[1]], result[[2]]), col = "red", lwd = 1)

plot(x, y, main = "Función `lm`")
abline(z, col = "red")
```





Como se puede observar la recta obtenida por ambos métodos es prácticamente la misma. Si comparamos los valores obtenidos por cada método obtenemos los siguiente.

Mediante el método del descenso del gradiente:

```
print(paste("beta0: ", result[[1]], "; beta1: ", result[[2]], "; RSE: ", result[[3]]))
```

```
## [1] "beta0:  2.36647888789399 ; beta1:  4.90550212651836 ; RSE:  1.38201462785111"
```

Mediante la función `lm`:

```
print(paste(
  "beta0: ",
  z$coefficients[[1]],
  "; beta1: ",
  z$coefficients[[2]],
  "; RSE: ",
  sigma(z)
))
```

```
## [1] "beta0:  2.36695251229913 ; beta1:  4.90534009887209 ; RSE:  1.38201460990146"
```