Método de descenso de gradiente en regresión

Alicia Jiajun Lorenzo, Abraham Trashorras, Mariel Chavez Rodriguez

2023-10-16

Para la práctica se consideró el siguiente modelo de regresión lineal simple:

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

En primera instancia se simula una muestra de tamaño n = 100 de valores de (xi, yi), i $\in \{1, \ldots, n\}$ para el modelo, generando dichas observaciones xi de la uniforme (runif).

```
n <- 100
xi <- runif(n)  # Genera 100 observaciones xi de la uniforme
```

Los errores del modelo se generan a partir de la distribución normal de media 0 y desviación típica $sigma$. Por ejemplo, si cogemos $\sigma = 1$, esto se puede hacer con (rnorm(100,0,1)).

```
sigma <- 1      # Variabilidad de los errores
epsilon <- rnorm(n, 0, sigma)  # Genera errores de la distribución normal
```

Los valores de y_i se calcularían como $y_i = \beta_0 + \beta_1 x_i + \epsilon_i$.

```
beta0<-5 # supongamos que betha0 = 5 para el modelo
beta1<-3 # supongamos que beta1 = 3 para el modelo
yi <- beta0 + beta1 * xi + epsilon  # Calculamos los valores de y
```

NOTA Se puede estudiar el comportamiento del modelo para distintos valores de la desviación típica.

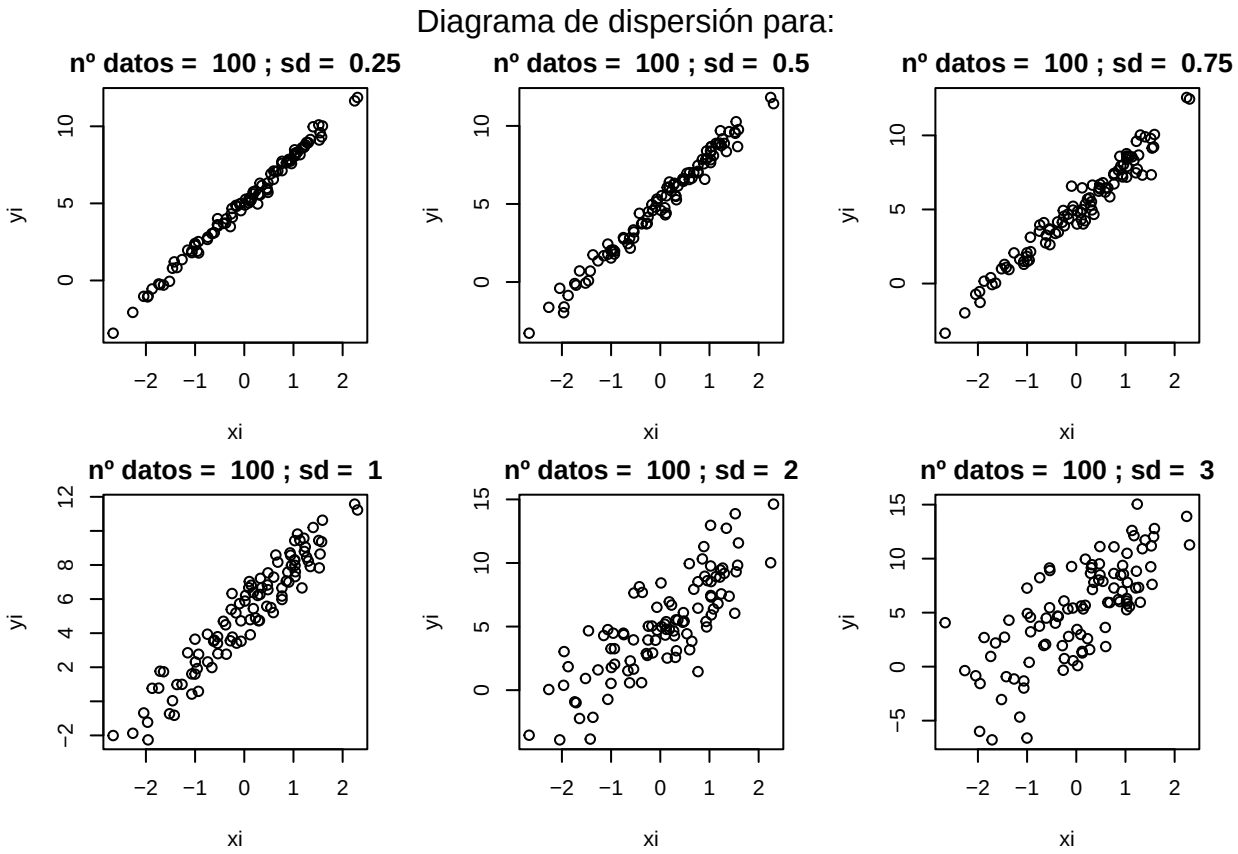
```
# Se crea el vector stds que contiene diferentes valores de desviación típica que se utilizarán para in
stds <- c(0.25, 0.5, 0.75, 1,2,3)
n <- 100  # Número de datos
beta0 <- 5
beta1 <- 3
xi <- rnorm(n)  # Datos xi generados aleatoriamente

# Para ordenar los gráficos se organizan en 2 filas y 2 columnas y se ajustan los márgenes
par(mfrow = c(2, 3), mar = c(4, 4, 2, 2),oma = c(0, 0, 2, 0))

# Se realiza este bucle for para poder iterar a través de los diferentes valores de desviación típica e
for (i in 1:length(stds)) {
  # En cada iteración se genera un vector de errores con la desviación estándar igual al valor correspo
  epsilon <- rnorm(n = n, sd = stds[i])
  # Calculamos y
  yi <- beta0 + beta1 * xi + epsilon
```

```r
  # Con los datos obtenidos de xi y yi, se generan las gráficas de dispersión correspondientes
  plot(xi, yi, main = paste('nº datos = ', n, '; sd = ', stds[i]))
}

# Agregamos un título general que englobe los subgráficos
mtext("Diagrama de dispersión para:", outer = TRUE, line = 0)
```



Nota Se puede observar como el aumento del valor de la desviación típica aumenta la dispersión de los puntos del modelo.

Para los siguientes apartados se usa una desviación típica de valor 2 ya que genera suficiente dispersión para llevar a cabo las pruebas que deseamos

```r
n <- 100  # Número de datos
beta0 <- 5
beta1 <- 3
xi <- rnorm(n)  # Datos xi generados aleatoriamente
epsilon <- rnorm(n = n, sd = 2)
# Calculamos y
yi <- beta0 + beta1 * xi + epsilon
```

Para aplicar el método de descenso de gradiente se escoge la función de la suma de los residuos al cuadrado:

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

```r
# Función a modelar, la cual calcula el valor de la variable dependiente para un valor de x dado y los
f <- function(x, beta0, beta1) {
return(beta0 + beta1 * x)
}

# Función de costo J(beta0, beta1)
grad_costo <- function(y,x,beta0, beta1) {
  # Obtenemos el error de valor y la prediccion
  error <- y - f(x,beta0,beta1)
  # Calculamos el gradiente de f(x)
  d_beta0 <- - sum(error)
  d_beta1 <- - sum(error * x)
  return(c(d_beta0, d_beta1))
}

# RSE (Residual Standard Error), para estimar la desviación típica de los residuos del modelo
rse <- function(y,x,beta0,beta1) {
prediccion<-f(x,beta0,beta1)
n <- length(y)
rss <- sum((prediccion - y)^2)
rse <- sqrt((rss / (n - 2)))
return(rse)
}
```

A partir de las funciones anteriormente definidas se crea la función para aplicar el algoritmo de método de descenso gradiente:

```r
# Función del descenso de gradiente
descenso_gradiente <- function(y, x, tasa_aprendizaje, num_iteraciones, tolerancia) {
  beta0_inicial <- 0
  beta1_inicial <- 0
  beta0_vector <- rep(0, times = num_iteraciones)
  converge=FALSE
  for (i in 1:num_iteraciones) {
    grad <- grad_costo(y,x,beta0_inicial, beta1_inicial)

    beta0_nuevo <- beta0_inicial - tasa_aprendizaje * grad[1]
    beta1_nuevo <- beta1_inicial - tasa_aprendizaje * grad[2]

    cambio_beta0 <- abs(beta0_nuevo - beta0_inicial)
    cambio_beta1 <- abs(beta1_nuevo - beta1_inicial)

    # Calculamos el rse cometido con la función de coste
    coste <- rse(y, x, beta0_nuevo, beta1_nuevo)


    beta0_inicial <- beta0_nuevo
    beta1_inicial <- beta1_nuevo
    beta0_vector[i]<-beta0_inicial

    # Comprobamos si se cumple el criterio de parada
    if (sqrt(sum(grad^2)) < tolerancia) {
      converge=TRUE
      break
```

```
    }

  }
  if (converge){
    print(paste("beta0: ",beta0_nuevo))
    print(paste("beta1: ",beta1_nuevo))
    print(paste("RSE: ", coste))
    print(paste("Iteracion en la que converge", i))
  }
  else{
    print(paste("beta0: ",beta0_nuevo))
    print(paste("beta1: ",beta1_nuevo))
    print(paste("RSE: ", coste))
    print(paste("NO CONVERGIO PARA EL NUMERO", i,"DE ITERACIONES"))
  }

  #print(paste("beta0: ",beta0_vector))
  resultado <- list(beta0_nuevo, beta1_nuevo,coste,converge,beta0_vector)
  return(resultado)
}

# Llamada a la función descenso_gradiente
tasa_aprendizaje <- 0.001
num_iteraciones <- 1000
tolerancia <- 0.0001

betas_estimados <- descenso_gradiente(yi, xi, tasa_aprendizaje, num_iteraciones, tolerancia)
```

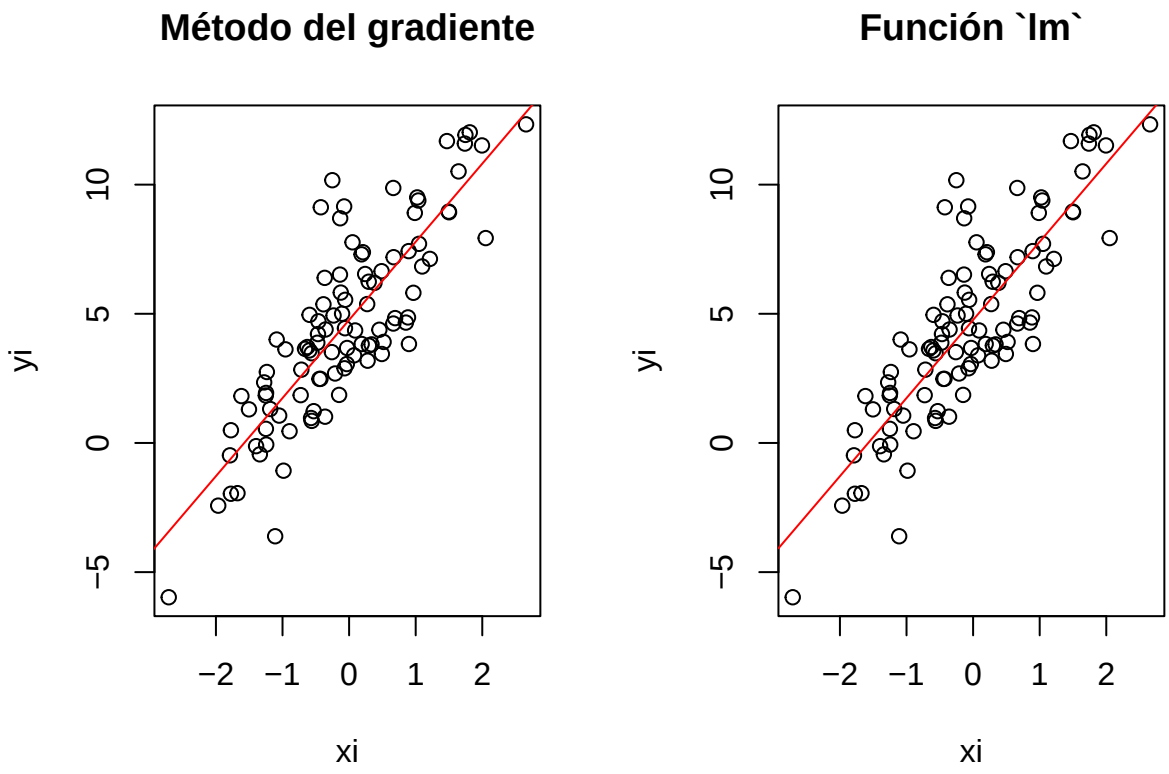
```
## [1] "beta0:  4.76328188591995"
## [1] "beta1:  3.0243317796544"
## [1] "RSE:  1.98956385705455"
## [1] "Iteracion en la que converge 159"
```

Se compara 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(xi, yi, main = "Método del gradiente")
lines(c(-10:10), f(c(-10:10), betas_estimados[[1]], betas_estimados[[2]]), col = "red", lwd = 1)

plot(xi, yi, main = "Función `lm`")
# Calculamos el modelo re regresión mediante `lm`
ml <- lm(yi ~ xi)
abline(ml, col = "red")
```



A simple vista parece que los resultados que aportan ambos metodos es similar así que se comprueban los resultados numéricos de cada método.

Descenso de gradiente:

```
print(paste("beta0: ",betas_estimados[[1]]))
```

```
## [1] "beta0:  4.76328188591995"
```

```
print(paste("beta1: ",betas_estimados[[2]]))
```

```
## [1] "beta1:  3.0243317796544"
```

```
print(paste("RSE: ", betas_estimados[[3]]))
```

```
## [1] "RSE:  1.98956385705455"
```

Usando el método lm para modelos lineales:

```
print(paste("beta0: ",ml$coefficients[[1]]))
```

```
## [1] "beta0:  4.76328261443232"
```

```r
print(paste("beta1: ",ml$coefficients[[2]]))
```

```
## [1] "beta1:  3.02433235993739"
```

```r
print(paste("RSE: ", sigma(ml)))
```

```
## [1] "RSE:  1.98956385705435"
```

Como se observa númericamente tambien son similares por lo que podemos deducir que ambos resultados se aproximan a la misma solución.