Ajuste de Polinomio de Tercer Grado

Armando Sánchez López

02/08/2024

Ajuste de Polinomio de Tercer Grado

Planteamiento

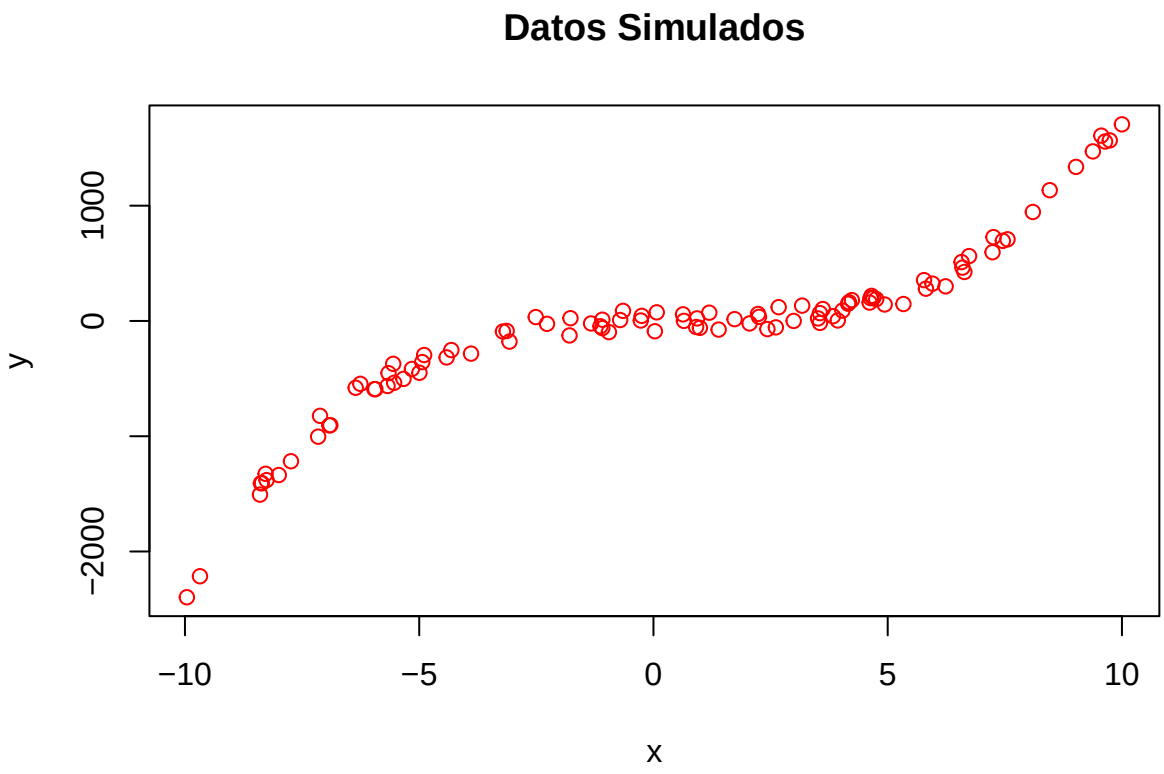
El objetivo de esta nota es ajustar un modelo de polinomio de tercer grado a un conjunto de datos simulados utilizando el método de gradiente descendente con mini-batches. Este proceso nos permitirá entender mejor cómo optimizar parámetros de un modelo polinomial y evaluar su rendimiento.

Generación de Datos Simulados

Para empezar, generaremos un conjunto de datos simulados para el análisis.

```
# Generar datos simulados
x <- sample(seq(-10, 10, length.out = 1000), size = 100, replace = TRUE)
a_true <- 2
b_true <- -3
c_true <- 5
d_true <- -1
y <- a_true * x^3 + b_true * x^2 + c_true * x + d_true + runif(length(x), -100, 100)

# Visualizar los datos simulados
plot(x, y, col = "red", main = "Datos Simulados", xlab = "x", ylab = "y")
```



División de los Datos

Dividiremos los datos en conjuntos de entrenamiento y prueba en una proporción 80/20 para evaluar el modelo posteriormente.

```r
# Dividir los datos en conjuntos de entrenamiento y prueba (80/20)
train_indices <- sample(1:length(x), size = 0.8 * length(x))
test_indices <- setdiff(1:length(x), train_indices)
x_train <- x[train_indices]
y_train <- y[train_indices]
x_test <- x[test_indices]
y_test <- y[test_indices]
```

Funciones Utilizadas en el Modelo

Inicialización de Parámetros

```r
# Función para inicializar parámetros
initialize_parameters <- function(seed = 42) {
  set.seed(seed)
  list(a = runif(1), b = runif(1), c = runif(1), d = runif(1))
}
```

Predicción

```
# Función para calcular predicciones
predict <- function(params, x) {
  params$a * x^3 + params$b * x^2 + params$c * x + params$d
}
```

Función de Costo

```
# Función para calcular la función de costo (MSE)
compute_cost <- function(y_true, y_pred) {
  mean((y_true - y_pred)^2)
}
```

Gradientes

```
# Función para calcular los gradientes
compute_gradients <- function(params, x, y_true) {
  y_pred <- predict(params, x)
  error <- y_pred - y_true
  grad_a <- mean(error * x^3)
  grad_b <- mean(error * x^2)
  grad_c <- mean(error * x)
  grad_d <- mean(error)
  list(grad_a = grad_a, grad_b = grad_b, grad_c = grad_c, grad_d = grad_d)
}
```

Actualización de Parámetros

```
# Función para actualizar los parámetros
update_parameters <- function(params, grads, learning_rate) {
  params$a <- params$a - (learning_rate * grads$grad_a)
  params$b <- params$b - (learning_rate * grads$grad_b)
  params$c <- params$c - (learning_rate * grads$grad_c)
  params$d <- params$d - (learning_rate * grads$grad_d)
  params
}
```

Creación de Mini-Batches

```
# Función para crear mini-batches
create_batches <- function(x, y, batch_size) {
  n <- length(y)
  indices <- sample(1:n)
  batches <- split(indices, ceiling(seq_along(indices)/batch_size))
  lapply(batches, function(batch_indices) {
    list(x = x[batch_indices], y = y[batch_indices])
  })
}
```

Entrenamiento del Modelo

```
# Función para entrenar el modelo usando gradiente descendente con mini-batches
train_model <- function(x, y, learning_rate, epochs, batch_size, seed) {
  params <- initialize_parameters(seed)
  for (epoch in 1:epochs) {
    batches <- create_batches(x, y, batch_size)
    for (batch in batches) {
      grads <- compute_gradients(params, batch$x, batch$y)
      params <- update_parameters(params, grads, learning_rate)
    }
    if (epoch %% 1000 == 0) {
      y_pred <- predict(params, x)
      cost <- compute_cost(y, y_pred)
      cat("Epoch:", epoch, "Cost:", cost, "\n")
    }
  }
  params
}
```

Entrenamiento del Modelo

Entrenaremos el modelo con los datos de entrenamiento.

```
# Entrenar el modelo
params <- train_model(x_train, y_train, learning_rate = 0.00001, epochs = 10000, batch_size = 20, seed =
```

```
## Epoch: 1000 Cost: 3693.28
## Epoch: 2000 Cost: 3432.053
## Epoch: 3000 Cost: 3380.853
## Epoch: 4000 Cost: 3910.791
## Epoch: 5000 Cost: 3575.926
## Epoch: 6000 Cost: 3400.453
## Epoch: 7000 Cost: 3749.472
## Epoch: 8000 Cost: 3431.854
## Epoch: 9000 Cost: 3324.09
## Epoch: 10000 Cost: 3333.515
```

```
# Imprimir los parámetros encontrados
cat("Parámetros encontrados:\n")
```

```
## Parámetros encontrados:
```

```
print(params)
```

```
## $a
## [1] 2.035105
##
## $b
## [1] -3.290275
##
## $c
## [1] 5.139012
##
## $d
## [1] 1.58516
```

Evaluación del Modelo

Evaluaremos el modelo en el conjunto de prueba y visualizaremos los resultados.

```
# Evaluar el modelo en el conjunto de prueba
y_pred_test <- predict(params, x_test)
test_cost <- compute_cost(y_test, y_pred_test)
cat("Error de prueba (MSE):", test_cost, "\n")
```

```
## Error de prueba (MSE): 3817.95
```

```
plot(x, y, col = "red")
points(x, predict(params, x), col = "blue")
```