Proyecto Final: Optimización de Regresión Lineal con Descenso del Gradiente

Guillermo Jair Montiel Juárez
Macro Entrenamiento de Inteligencia Artificial (MeIA) 2025

Junio 2025

Objetivo

Implementar desde cero el método del descenso del gradiente para resolver una tarea de regresión lineal sobre el conjunto de datos Iris. Se busca comprender y aplicar los fundamentos matemáticos y computacionales detrás de la optimización iterativa de modelos.

1. Introducción

El descenso del gradiente es un algoritmo de optimización iterativo usado para encontrar el mínimo de una función objetivo. En el contexto de aprendizaje automático, se utiliza para ajustar los parámetros de modelos como la regresión lineal, minimizando el error cuadrático medio (MSE) entre predicciones y valores reales.

En este proyecto, trabajamos con el conjunto de datos **Iris**, uno de los datasets clásicos en la ciencia de datos y el aprendizaje automático. El dataset Iris contiene 150 muestras de flores de tres especies diferentes (*Iris setosa*, *Iris versicolor*, *Iris virginica*), y para cada flor se registran cuatro características: longitud y ancho del sépalo, y longitud y ancho del pétalo (todas en centímetros).

Utilizaremos la regresión lineal para modelar la relación entre tres características independientes (*sepal width*, *petal width*, *sepal length*) y la variable objetivo (*petal length*). Es decir, construiremos un modelo capaz de predecir la longitud del pétalo de una flor basada en las otras tres medidas, ajustando los parámetros del modelo mediante el descenso del gradiente.

Modelo de regresión lineal:

$$\hat{y} = \sum_{j=1}^{m} w_j x_j + b \tag{1}$$

donde:

- m: número de características,

- x_j: variable independiente,
- w_j: coeficiente de la característica j,
- b: término independiente,
- y: variable objetivo.

2. Función de costo

Se utiliza el error cuadrático medio (MSE) como función de costo:

$$f(W, b) = \frac{1}{2N} \sum_{i=1}^{N} (\hat{y}^{(i)} - y^{(i)})^2 \tag{2}$$

donde N es el número de muestras.

3. Gradientes parciales

Los gradientes para actualizar los parámetros son:

$$\frac{\partial f}{\partial w_j} = \frac{1}{N} \sum_{i=1}^{N} (\hat{y}^{(i)} - y^{(i)}) x_j^{(i)} \tag{3}$$

$$\frac{\partial f}{\partial b} = \frac{1}{N} \sum_{i=1}^{N} (\hat{y}^{(i)} - y^{(i)}) \tag{4}$$

4. Algoritmo del Descenso del Gradiente

1. Inicializar w_j y b (por ejemplo en cero).
2. Repetir para $k = 0, \ldots, K$:
   a) Calcular gradientes de w_j y b.
   b) Actualizar los parámetros:

$$w_j^{k+1} = w_j^k - \alpha \frac{\partial f}{\partial w_j}$$

$$b^{k+1} = b^k - \alpha \frac{\partial f}{\partial b}$$

3. Verificar convergencia.

5. Implementación en Python

A continuación se explica cada bloque de código utilizado, junto con su función dentro del proyecto.

Carga y preprocesamiento del dataset

Se importan las librerías necesarias para manipulación de datos (numpy, pandas), visualización (matplotlib), para cargar el dataset Iris. Luego, se renombran las columnas para mayor claridad y se seleccionan las variables independientes y la variable dependiente según las instrucciones del proyecto.

```
import numpy as np
import pandas as pd
from sklearn.datasets import load_iris

iris = load_iris(as_frame=True)
df = iris.frame
df.columns = [col.lower().replace(" (cm)", "").replace(" ", "_")
    for col in df.columns]

X = df[['sepal_width', 'petal_width', 'sepal_length']].values  #
    Variables independientes
y = df['petal_length'].values.reshape(-1, 1)                   #
    Variable dependiente
```

Código 1: Carga y preprocesamiento del dataset Iris

Función de costo

Se define la función de costo, que calcula el error cuadrático medio (MSE) entre los valores predichos por el modelo y los valores reales. Este valor será el que el descenso del gradiente intentará minimizar.

```
def compute_cost(X, y, W, b):
    m = X.shape[0]
    y_pred = X @ W + b
    cost = (1/(2*m)) * np.sum((y_pred - y)**2)
    return cost
```

Código 2: Función de costo (MSE)

Descenso del Gradiente

Se implementa el descenso del gradiente desde cero. Se inicializan los parámetros, se calcula el error y los gradientes en cada época, se actualizan los parámetros y se almacena el historial del costo para graficar su evolución.

```
def gradient_descent(X, y, alpha=0.01, epochs=1000):
    m, n = X.shape
    W = np.zeros((n, 1))
    b = 0
    costs = []
    for epoch in range(epochs):
        y_pred = X @ W + b
        error = y_pred - y
        dW = (1/m) * (X.T @ error)
```

```
        db = (1/m) * np.sum(error)
        W -= alpha * dW
        b -= alpha * db
        costs.append(compute_cost(X, y, W, b))
    return W, b, costs
```

Código 3: Algoritmo de descenso del gradiente

Entrenamiento del modelo

Se definen la tasa de aprendizaje y el número de épocas (ajustado a 1000), y se entrena el modelo usando la función de descenso del gradiente para obtener los parámetros optimizados y el historial de la función de costo.

```
alpha = 0.01
epochs = 1000
W, b, costs = gradient_descent(X, y, alpha, epochs)
```

Código 4: Entrenamiento del modelo

Visualización de la función de costo

Se grafica la evolución del costo (MSE) durante el entrenamiento, lo que permite verificar visualmente el proceso de convergencia del modelo.

```
import matplotlib.pyplot as plt
plt.figure(figsize=(8,4))
plt.plot(costs)
plt.xlabel("Época")
plt.ylabel("Costo (MSE)")
plt.title("Evolución del costo durante el entrenamiento")
plt.show()
```

Código 5: Gráfica de la evolución del costo

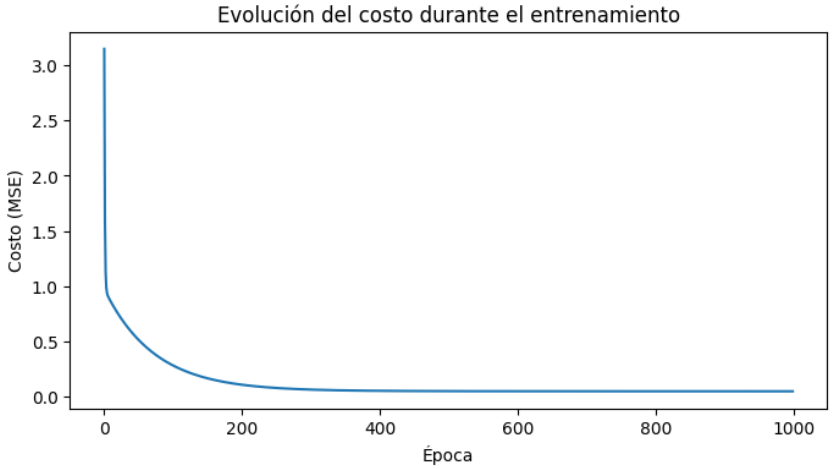

Figura 1: Evolución del costo (MSE) a lo largo de las épocas. Una tendencia decreciente indica que el modelo está aprendiendo y convergiendo.

Gráfica de valores predichos vs. reales

Se compara visualmente las predicciones del modelo con los valores reales. Los puntos cercanos a la diagonal indican un buen ajuste del modelo.

```
y_pred = X @ W + b
plt.figure(figsize=(6,6))
plt.scatter(y, y_pred, alpha=0.6)
plt.plot([y.min(), y.max()], [y.min(), y.max()], '--r',
   label="Ideal")
plt.xlabel("Longitud de pétalo real")
plt.ylabel("Longitud de pétalo predicha")
plt.title("Predicción vs. Real (Descenso del Gradiente)")
plt.legend()
plt.show()
```

Código 6: Gráfica de valores predichos vs. reales

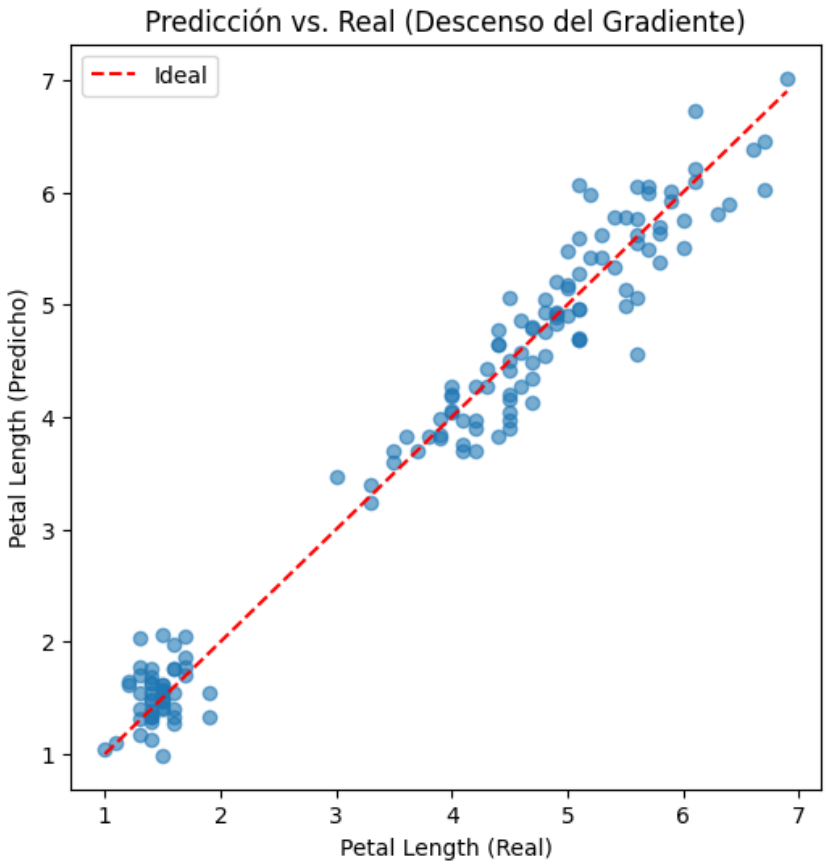

Figura 2: Valores predichos contra valores reales. Los puntos cercanos a la línea roja indican buenas predicciones.

Comparación con LinearRegression de scikit-learn

Se entrena un modelo de regresión lineal con la clase `LinearRegression` de scikit-learn para comparar los resultados con la implementación del descenso del gradiente.

```
from sklearn.linear_model import LinearRegression

lr = LinearRegression()
lr.fit(X, y)
y_pred_skl = lr.predict(X)

plt.figure(figsize=(6,6))
plt.scatter(y, y_pred, alpha=0.5, label="Gradiente")
plt.scatter(y, y_pred_skl, alpha=0.5, label="sklearn", marker='x')
plt.plot([y.min(), y.max()], [y.min(), y.max()], '--r')
plt.xlabel("Longitud de pétalo real")
plt.ylabel("Longitud de pétalo predicha")
plt.title("Comparación: Gradiente vs. sklearn")
plt.legend()
plt.show()
```

Código 7: Comparación con LinearRegression de scikit-learn

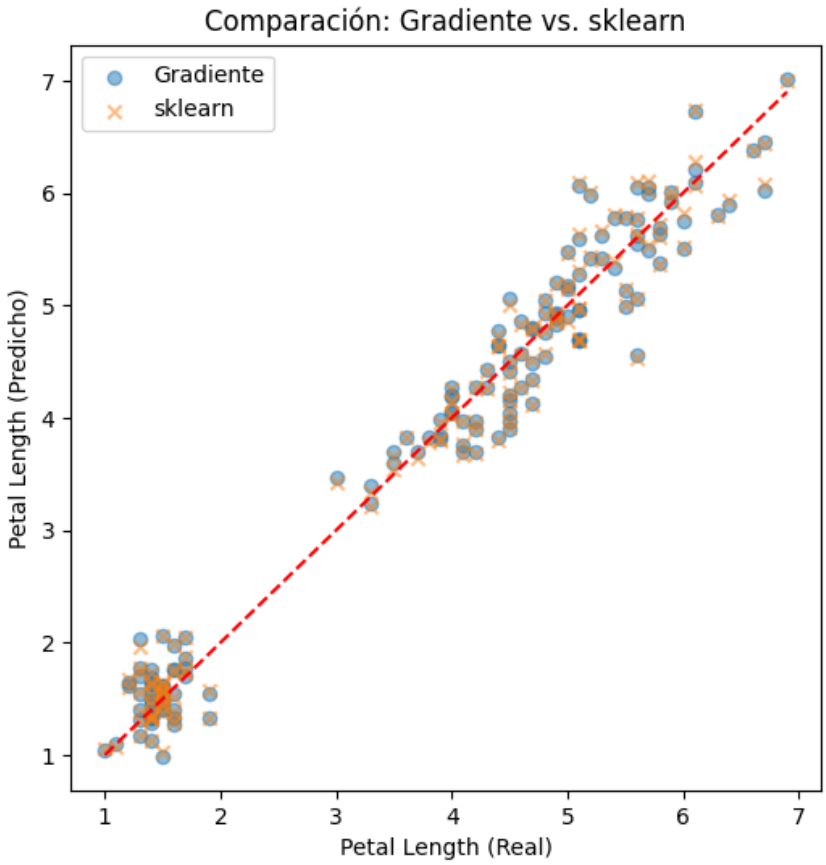

Figura 3: Comparación entre el modelo entrenado con descenso del gradiente y la solución analítica de scikit-learn. La similitud indica una correcta implementación.

6. Resultados e interpretación

La implementación desde cero del descenso del gradiente logra resultados comparables a los obtenidos mediante la regresión lineal clásica (solución cerrada) de scikit-learn. La evolución de la función de costo muestra una convergencia adecuada, y la gráfica de predicciones indica que el modelo ajusta bien los datos. Cualquier diferencia entre ambas soluciones puede deberse al número de épocas, la tasa de aprendizaje o la inicialización de los parámetros. Al intentar applicar el algoritmo durante menos épocas, los resultados empiezan a diferir con más notoriedad, por lo que se considera que 1000 es adecuado para este caso.

7. Conclusiones

Se implementó exitosamente el descenso del gradiente para regresión lineal usando el dataset Iris, replicando los resultados del método analítico de scikit-learn. Esto valida tanto la comprensión del algoritmo como su codificación.

8. Referencias

- Géron, A. (2019). *Hands-On Machine Learning with Scikit-Learn, Keras, and TensorFlow*. O'Reilly.
- Documentación oficial de scikit-learn: `https://scikit-learn.org/stable/modules/generated/sklearn.linear_model.LinearRegression.html`
- Documentación oficial de NumPy: `https://numpy.org/doc/`
- Iris dataset en scikit-learn: `https://scikit-learn.org/stable/auto_examples/decomposition/plot_pca_iris.html`
- Apuntes y especificaciones del curso MeIA 2025.