Proyecto Final - MeIA 2025

Optimización de Regresión Lineal con Descenso del Gradiente

Alumno: Jesus Antonio Torres Contreras

Fecha: 7 de Junio de 2025

1. Introducción

La regresión lineal es uno de los modelos más fundamentales en el aprendizaje automático. Su objetivo es encontrar una relación lineal entre una o más variables independientes y una variable dependiente. En este proyecto se implementó desde cero el algoritmo de descenso del gradiente para resolver una tarea de regresión lineal, utilizando el conjunto de datos Iris.

Se evitó el uso de modelos ya implementados como LinearRegression de scikit-learn, y se trabajó directamente con NumPy para llevar a cabo las operaciones vectorizadas.

2. Objetivo

Implementar el método del descenso del gradiente para resolver una tarea de regresión lineal sobre el conjunto de datos Iris.

3. Explicación de Algoritmo

El algoritmo de descenso del gradiente se basa en actualizar los parámetros del modelo (pesos y bias) de manera iterativa en la dirección opuesta al gradiente de la función de costo.

```
4. from sklearn.datasets import load_iris
5. import pandas as pd
6. import numpy as np
7.
8. # Cargar los datos como DataFrame
9. iris = load_iris(as_frame=True)
10.      df = iris.frame
11.
12.      # Renombrar columnas por claridad si lo deseas
13.      df.columns = [col.lower().replace(" (cm)", "").replace(" ",
   "_") for col in df.columns]
14.
15.      # Extraer las variables necesarias
16.      X = df[['sepal_width', 'petal_width',
   'sepal_length']].values  # Variables independientes
17.      y = df['petal_length'].values.reshape(-1, 1)  # Variable
   dependiente y le aplique un reshape para manejar mas facil los datos
```

En esta primera parte mandamos a llamar a las librerías, la primera es para importar la base de datos de iris directamente (que es la base de datos que vamos a usar), la segunda para llamar a pandas que nos ayudara a poder manejar los datos de esta base de datos y por ultimo la de numpy que nos ayudara hacer operaciones con estos datos.

Primero cargamos los datos de iris como un data frame, luego renombramos las columnas y por último vamos a extraer tanto variables independientes como la dependiente (que es la que vamos a predecir).

```
np.random.seed(123)
m, n = X.shape
W = np.zeros((n, 1))
b = 0.0
alpha = 0.01
epocas = 1000

def costo_predicciones(X, y, W, b):
    N = len(y)
    y_pred = X @ W + b
    cost = (1 / (2 * N)) * np.sum((y_pred - y) ** 2)
    return cost

def gradiente(X, y, W, b):
    N = len(y)
    y_pred = X @ W + b
    error = y_pred - y
    dW = (1 / N) * (X.T @ error)
    db = (1 / N) * np.sum(error)
    return dW, db
```

Aquí declaramos primero con valores randoms para nuestra base de datos y el resto son variables que definimos desde un principio para que estas mismas se vayan actualizando en cada época. También tenemos a nuestras dos funciones, estas nos permiten calcular el error del modelo actual (costo) y determinar cómo actualizar los parámetros (gradientes).

```
cost_history = []

for epoch in range(epocas):
    dW, db = gradiente(X, y, W, b)
    W -= alpha * dW
    b -= alpha * db
    cost = predicciones(X, y, W, b)
    cost_history.append(cost)

    if epoch % 100 == 0:
        print(f"Epoca {epoch}: Costo = {cost:.4f}")
```

En este bloque realizamos el ciclo de entrenamiento del modelo. En cada iteración actualizamos los pesos y sesgo, y registramos el costo para observar su evolución.

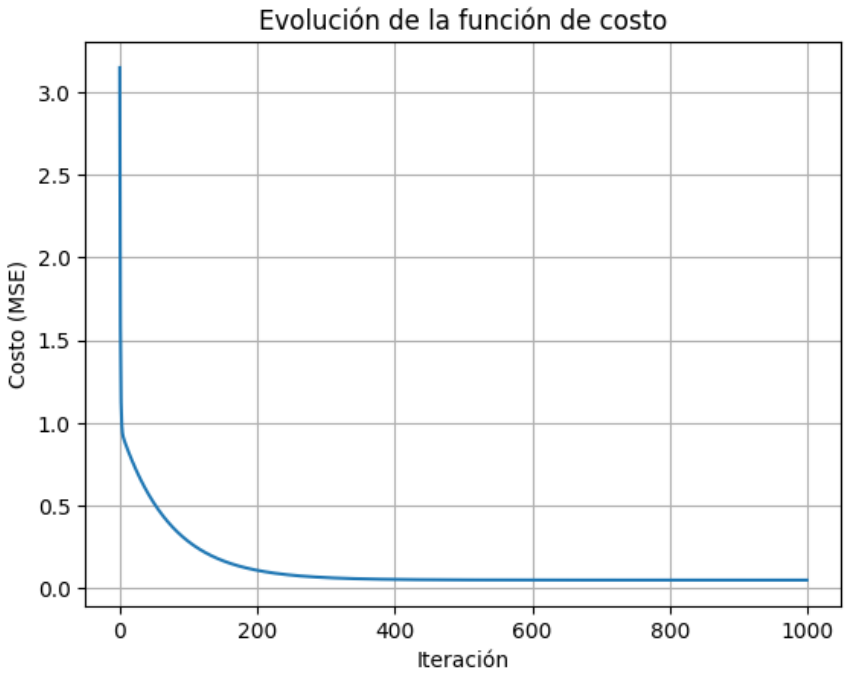
```
Epoca 0: Costo = 3.1475
Epoca 100: Costo = 0.2828
Epoca 200: Costo = 0.1089
Epoca 300: Costo = 0.0651
Epoca 400: Costo = 0.0540
Epoca 500: Costo = 0.0511
Epoca 600: Costo = 0.0504
Epoca 700: Costo = 0.0501
Epoca 800: Costo = 0.0500
Epoca 900: Costo = 0.0499
```

Estos son los resultados del código anterior y vemos que al paso de las épocas mejoro bastante su costo.

```
import matplotlib.pyplot as plt

plt.plot(cost_history)
plt.xlabel("Iteración")
plt.ylabel("Costo (MSE)")
plt.title("Evolución de la función de costo")
plt.grid(True)
plt.show()
```

En esta parte mandamos llamar a la función para graficar de matplotlib. La gráfica muestra cómo disminuye el error a lo largo del entrenamiento.



La grafica también nos muestra la evolución que tiene la función de costo con respecto a cada iteración y se nota como esta baja de manera suave y a un punto bastante bajo.

```
from sklearn.linear_model import LinearRegression
model = LinearRegression()
model.fit(X, y)

print("\nModelo propio:")
print("W:", W.ravel())
print("b:", b)

print("\nModelo con scikit-learn:")
print("W:", model.coef_.ravel())
print("b:", model.intercept_[0])
```

Ahora si mandamos llamar a LinearRegression para hacer una regresión lineal con nuestros datos de manera mas directa y sencilla. Ya con este entrenamiento con regresión lineal hacemos una comparación:

Modelo propio:

W: [-0.71339682 1.40626823 0.74551931]

b: -0.10219937214494418

Modelo con scikit-learn:

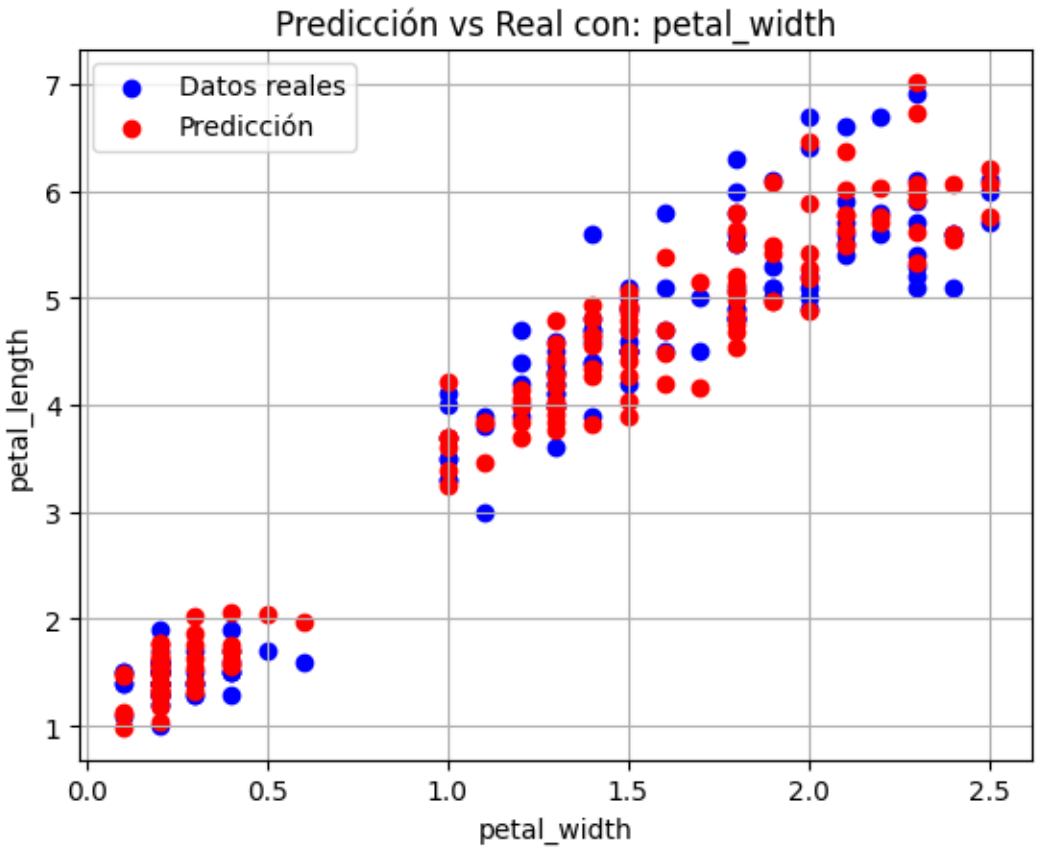
W: [-0.64601244 1.44679341 0.72913845]

b: -0.2627111975741898

Y como vemos ambos resultados son bastante parecidos, por lo que el modelo que creamos resulta ser bueno.

```
plt.scatter(X[:, 1], y, color='blue', label="Datos reales")
plt.scatter(X[:, 1], X @ W + b, color='red', label="Predicción")
plt.xlabel("petal_width")
plt.ylabel("petal_length")
plt.title("Predicción vs Real con: petal_width")
plt.legend()
plt.grid(True)
plt.show()
```

Ya por ultimo, observamos la comparación con grafica de los datos reales con respecto a nuestra predicción con la ayuda de matplotlib.



Y ahora podemos confirmar que nuestro modelo resulta ser acertado ya que vemos bastante similitudes entre ambos puntos.

4. Conclusiones

Durante el desarrollo de este proyecto pude comprender a profundidad cómo funciona el algoritmo de descenso del gradiente y cómo se utiliza para ajustar un modelo de regresión lineal. Al implementar el algoritmo desde cero, reforcé mi entendimiento sobre derivadas parciales, optimización y la importancia del aprendizaje iterativo.

Los resultados obtenidos fueron satisfactorios, ya que el modelo convergió adecuadamente y las predicciones fueron comparables con las obtenidas usando LinearRegression de scikit-learn. Esto valida que la lógica de actualización de parámetros fue implementada correctamente.

Además, observé que la tasa de aprendizaje y el número de iteraciones juegan un papel clave en la convergencia del modelo. Una tasa de aprendizaje muy alta puede causar divergencia, mientras que una muy baja puede hacer que el modelo aprenda lentamente.

En general, este proyecto me ayudó a afianzar habilidades prácticas en programación con NumPy, análisis de datos con pandas, y visualización con matplotlib, además de fortalecer los conceptos teóricos de machine learning.

5. Referencias bibliográficas

- Géron, A. (2019). Hands-On Machine Learning with Scikit-Learn, Keras, and TensorFlow: Concepts, Tools, and Techniques to Build Intelligent Systems (2nd ed.). O'Reilly Media.
- Scikit-learn. (s.f.). Plot Iris dataset. https://scikit-learn.org/stable/auto_examples/datasets/plot_iris_dataset.html
- NumPy. (s.f.). NumPy documentation. https://numpy.org/doc/

Link a COLAB:
https://colab.research.google.com/drive/14vaOVvHtva1Gk_8lJ8Oqeje4GgTimOGW?usp=sharing