Laboratorio 3: Regresión Lineal con Descenso del Gradiente

1. Introducción

El objetivo principal de este laboratorio ha sido implementar un sistema de Regresión Lineal utilizando el método del Descenso del Gradiente (Gradient Descent). El propósito del programa es que el ordenador "aprenda" la relación lineal entre unas variables de entrada y una variable de salida, minimizando iterativamente el error entre las predicciones y los valores reales.

Para lograr esto, hemos tenido que implementar tres clases nuevas que interactúan con las clases que ya teníamos:

Model: Esta clase representa la hipótesis. Almacena los parámetros (pesos) de la ecuación lineal, incluyendo el término independiente (bias). Sus responsabilidades principales son realizar predicciones (predict) usando el producto escalar y actualizar sus propios parámetros (update) basándose en el gradiente calculado.

Algorithm: Esta clase encapsula la lógica matemática del Descenso del Gradiente. Se encarga de calcular el gradiente de la función de coste (Ecuación 1) y ejecuta el bucle principal para minimizar el error hasta que se cumple el criterio de parada.

SupervisedLearner: Actúa como una clase "gestora" o envoltorio. Combina un Dataset y un Algorithm para entrenar un Model. Simplifica la interacción con el usuario, permitiéndonos llamar a solve() para entrenar y predict() para usar el modelo una vez entrenado.

2. Descripción de la Solución

Dado que la estructura de clases venía definida por el diagrama UML del enunciado, nuestras decisiones de diseño se centraron en la lógica interna del algoritmo, específicamente en el criterio de parada del entrenamiento:

Alternativa A (Número fijo de iteraciones): Valoramos implementar el método solve() usando un bucle for simple que se ejecutase un número fijo de veces (por ejemplo, 1000 iteraciones) independientemente del resultado.

Ventaja: Es muy fácil de programar.

Desventaja: Es ineficiente. Si el modelo aprende rápido, seguimos perdiendo tiempo. Si el modelo necesita más tiempo, cortamos el entrenamiento antes de acabar.

Alternativa B (Solución Elegida - Convergencia basada en la Norma): Decidimos implementar un bucle while que continúa hasta que la norma del gradiente es menor que un criterio de parada (stoppingCriteria).

Razón de la elección: Esta solución es mucho más precisa. Garantiza que el algoritmo solo se detiene cuando el error es realmente bajo (cuando hemos llegado al "fondo" de la función de coste).

Nota de implementación: Para evitar bucles infinitos si el modelo no converge (por ejemplo, si el learning rate es malo), añadimos una variable de seguridad (maxIterations) para forzar la parada si se tarda demasiado.

Conceptos teóricos de POO aplicados:

Encapsulamiento: Hemos protegido los atributos críticos (como los pesos del modelo y la tasa de aprendizaje) haciéndolos private.

Composición: La clase SupervisedLearner utiliza composición. En lugar de heredar código, esta clase contiene instancias de Model y Algorithm, coordinando el trabajo entre ambas piezas.

3. Conclusión

La solución funcionó de manera efectiva en la práctica. Para verificar la implementación, creamos una clase MainTest robusta siguiendo las directrices del profesor.

En lugar de probar con números aleatorios, utilizamos una estrategia de "Vector Ideal":
-Definimos una fórmula objetivo conocida (por ejemplo, y=2x+3y+4).
-Generamos un dataset pequeño (N=4,D=2) basado exactamente en esa fórmula.
-Ejecutamos el algoritmo y verificamos que los pesos aprendidos convergían a [2.0,3.0,4.0].

Resultados: Las pruebas mostraron que la norma del gradiente disminuía en cada iteración, confirmando que las ecuaciones matemáticas (1-3) estaban correctamente implementadas. El error final de predicción fue insignificante (<0.1), demostrando que el modelo aprendió el patrón correctamente.

Dificultades: El principal desafío fue manejar el riesgo de bucle infinito en la clase Algorithm. Si la tasa de aprendizaje era demasiado alta, el modelo no convergía nunca. Solucionamos esto implementando un contador de seguridad local (maxIterations) para forzar la parada del bucle si tarda demasiado. También tuvimos que asegurarnos de manejar correctamente el bias, recordando siempre usar el método augment() para añadir el 1.0 extra a los vectores de entrada.