Resumen del capítulo: Descenso de gradiente

Minimización de la función de pérdida

Antes de plantear una tarea de descenso de gradiente, veamos el problema de entrenamiento desde un nuevo ángulo. Ya conoces la función de pérdida. Esta devuelve el número de pérdidas de las respuestas incorrectas del modelo y suele utilizarse para su entrenamiento.

En el curso anterior, equiparamos la función de pérdida con el ECM. En este nuevo ejercicio, vamos a expresar la función de pérdida como *L(y, a),* donde el vector *y* representa las respuestas correctas y el vector *a* representa las predicciones.

Escribe la tarea de entrenamiento del modelo mediante la siguiente función de pérdida:

$$w = \arg\min_{w} L(y, a)$$

Para calcular el ECM, o la **función de pérdida cuadrática**, eleva al cuadrado la diferencia entre las respuestas correctas y las predicciones:

$$L(y, a) = \sum_{i=1}^{n} (a_i - y_i)^2$$

¿Qué otras funciones de pérdida existen para una tarea de regresión? Si queremos que la función de la tarea sea menos sensible a los valores atípicos, entonces en lugar del ECM, utilizaremos la **función de pérdida absoluta** (EAM), que también conoces

ya. Al igual que con el *ECM,* la utilizaremos como una función de pérdida, no como una métrica de evaluación.

$$L(y, a) = \sum_{i=1}^{n} |a_i - y_i|$$

Cuando se trata de problemas de clasificación, a menudo se utiliza la métrica de *exactitud*. Pero rara vez se puede utilizar como una función de pérdida. Esto se debe a que la función de cálculo de la exactitud no tiene ninguna **derivada** que muestre el cambio en la función ante pequeños cambios en el argumento.

Sustituyamos la *exactitud* por la **verosimilitud logarítmica negativa** o la **función de pérdida logística**. La función de pérdida suma los logaritmos de las probabilidades en función de la observación. Si la respuesta correcta es 0, se suma $log_{10}\, a_i$. Si es cero, se suma $log_{10}\,(1 - a_i)$. El **logaritmo** es la potencia a la que se eleva la base (en nuestro caso, 10) para extraer el argumento.

Aquí tienes la fórmula:

$$L(y, a) = -\sum_{i=1}^{n} \begin{cases} \log_{10} a_i & \text{if } y_i = 1 \\ \log_{10}(1 - a_i) & \text{if } y_i = 0 \end{cases}$$

donde a_i es la probabilidad de la clase 1 para la observación con índice *i*. Es decir, el valor de a_i debe ser lo más alto posible para una observación de clase positiva y lo más bajo posible para una observación de clase negativa.

El nombre de la verosimilitud logarítmica negativa proviene de la **función de verosimilitud**, que calcula la probabilidad de que el modelo dé respuestas correctas para todas las observaciones si los valores a_i se toman como respuesta:

$$\text{Probabilidad}(y, a) = \prod_{i=1}^{n} \begin{cases} a_i & \text{if } y_i = 1 \\ 1 - a_i & \text{if } y_i = 0 \end{cases}$$

Al tomar el logaritmo, los valores "se expanden" a un rango más amplio. Por ejemplo, el rango de 0 a 1 se convierte en el rango de -∞ a 0. Para este rango, los errores de cálculo no son tan importantes. Multiplicamos por -1 porque la función de pérdida final debe minimizarse.

A diferencia de la *exactitud*, la función de pérdida logarítmica tiene una derivada.

Gradiente de una función

No siempre podemos encontrar manualmente el mínimo de la función de pérdida. El **gradiente de una función** puede ayudar a encontrar la dirección.

La función de pérdida depende de los parámetros del algoritmo. Esta función es una **función de valor vectorial**, es decir, toma un vector y devuelve un escalar.

El gradiente de una función de valor vectorial es un vector que está formado por las derivadas de la respuesta para cada argumento. Se expresa como ∇ (nabla - un arpa hebrea de figura similar a la del símbolo). El gradiente de la función *f* de un vector n-dimensional x se calcula de la siguiente manera:

$$\nabla f(x) = \left(\frac{\partial f}{\partial x_1}, \frac{\partial f}{\partial x_2}, \dots, \frac{\partial f}{\partial x_n} \right)$$

donde $\partial f/\partial xi$ es la derivada parcial de la función *f* para el argumento x_i. La función gradiente para un argumento es la derivada.

El gradiente indica la dirección en la que la función crece más rápido. Sin embargo, no sirve para resolver el problema de minimización. Necesitamos el **gradiente** negativo, que es un vector opuesto que muestra el decrecimiento más rápido. Lo podemos determinar de la siguiente manera:

$$-\nabla f(x)$$

Descenso de gradiente

El **descenso de gradiente** es un algoritmo iterativo para encontrar el mínimo de la función de pérdida. Sigue la dirección del gradiente negativo y se aproxima gradualmente al mínimo.

Es difícil llegar al mínimo en una sola iteración, porque el vector de gradiente negativo no indica el punto mínimo de la función de pérdida, sino la dirección del decrecimiento.

Para comenzar el descenso, vamos a elegir el **valor inicial** del argumento (vector *x*). Se expresa como x^0. A partir de ahí, se realizará el primer **paso de descenso de gradiente**. El siguiente punto, x^1, se calcula de la siguiente manera: al punto x^0 se le suma el gradiente negativo multiplicado por el tamaño del paso de descenso del gradiente (μ).

$$x^1 = x^0 - \mu \times \nabla f(x)$$

El valor μ determina el tamaño del paso de descenso de gradiente. Si el paso es pequeño, el descenso pasará por muchas iteraciones. No obstante, cada una de ellas nos acercará al mínimo de la función de pérdida. Si el paso es pequeño, el descenso pasará por muchas iteraciones. No obstante, cada una de ellas nos acercará al mínimo de la función de pérdida. Si el paso es demasiado grande, podemos pasar por alto el mínimo (chocaremos contra la roca como un buceador).

Repite la operación para obtener los valores de los argumentos en las siguientes iteraciones. El número de iteraciones se expresa como *t*. Para obtener los nuevos valores de x^t, hay que multiplicar el gradiente negativo por el tamaño del paso y sumar este producto al valor anterior:

$$x^t = x^{t-1} - \mu \times \nabla f(x^{t-1})$$

El descenso de gradiente está completado cuando:

- el algoritmo complete el número necesario de iteraciones

o

- el valor de *x* deje de cambiar.

Descenso de gradiente en Python

Primero, vamos a hacer un resumen de los pasos necesarios para ejecutar un algoritmo de descenso de gradiente:

1. En los argumentos del algoritmo, establece el valor inicial, x^0.
2. Calcula el gradiente de la función de pérdida (es el vector de derivadas parciales respecto a cada argumento que toma el vector *x* como entrada).
3. Encuentra un nuevo valor mediante la fórmula:

   $$x^t = x^{t-1} - \mu \times \nabla f(x^{t-1})$$

   donde *μ* es el tamaño de paso que se establece en el argumento del algoritmo.
4. Realiza el número de iteraciones especificado en los argumentos.

```
import numpy as np

def func(x):
    # función que se minimizará

def gradient(x):
    # gradiente de función func

def gradient_descent(initialization, step_size, iterations):
    x = initialization
    for i in range(iterations):
        x = x - step_size * gradient(x)
    return x
```