Gradient Descent

¿Por qué buscar otras formas de entrenar nuestro modelo si tenemos una fórmula?

La complejidad computacional de invertir $X^T X$ está entre $O(n^{2.4})$ y $O(n^3)$ dependiendo de la implementación. Esto puede ser muy lento para grandes datos.

Gradient Descent es un algoritmo que encuentra óptimos mediante iteraciones. Así, si nuestra función objetivo es:

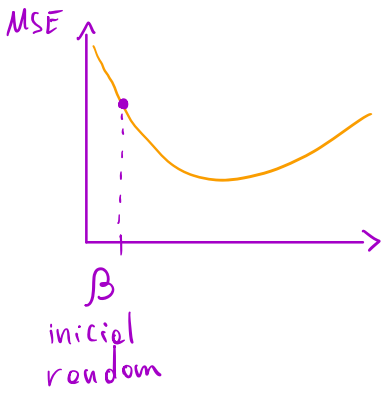
$$\min \sum_{i=1}^{n} (Y_i - \underbrace{\beta^T X_i}_{\hat{Y}_i})^2 \rightsquigarrow \min \underbrace{\frac{1}{n} \sum_{i=1}^{n} (Y_i - \beta^T X_i)^2}_{MSE}$$

* Recordemos que

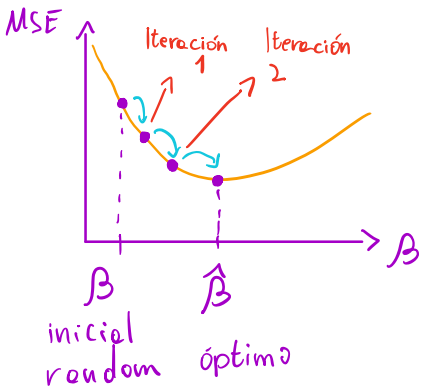
$$\beta \cdot X_i = \beta^T X_i$$

Nosotros queremos encontrar los valores de β que minimizan el MSE. Este algoritmo inicializa β aleatorio y los empieza a mover en el sentido del gradiente.

Ej. Supongamos que con el valor inicial tenemos esto:

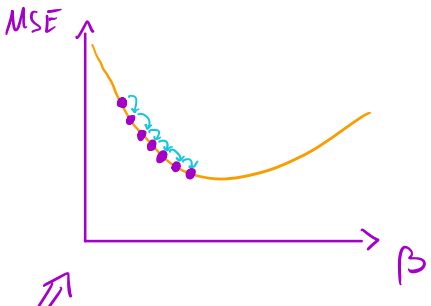


Así, nos moveremos en la dirección en la que decrece la función objetivo.

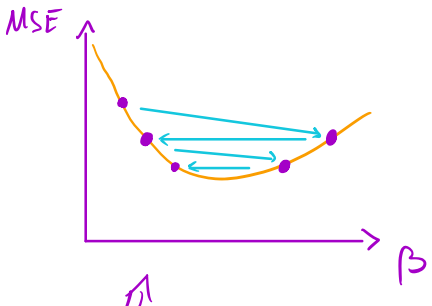


Esto es análogo a ir descendiendo un cerro por tanteo.

Además podemos controlar el tamaño de nuestros saltos. Esto se denomina **Learnin rate**.



Learning rate bajo



Learning rate alto

Ojo! este método se puede quedar "estancado" en mínimos locales, pero como la función MSE es convexa, sabemos que llegará a un mínimo global.

El método se llama Gradient Descent porque en cada iteración nos movemos en la dirección en la que va el gradiente. Recordemos que el MSE en este caso va de $\mathbb{R}^k \longmapsto \mathbb{R}$ con k número de features.

Observación: $$\frac{\delta\, MSE}{\delta\, \beta_j} = \frac{2}{n} \sum_{i=1}^{n} \left(\beta^T X_i - Y_i\right) X_{ij}$$

$$\nabla_\beta MSE = \frac{2}{n} X^T (X\beta - Y)$$