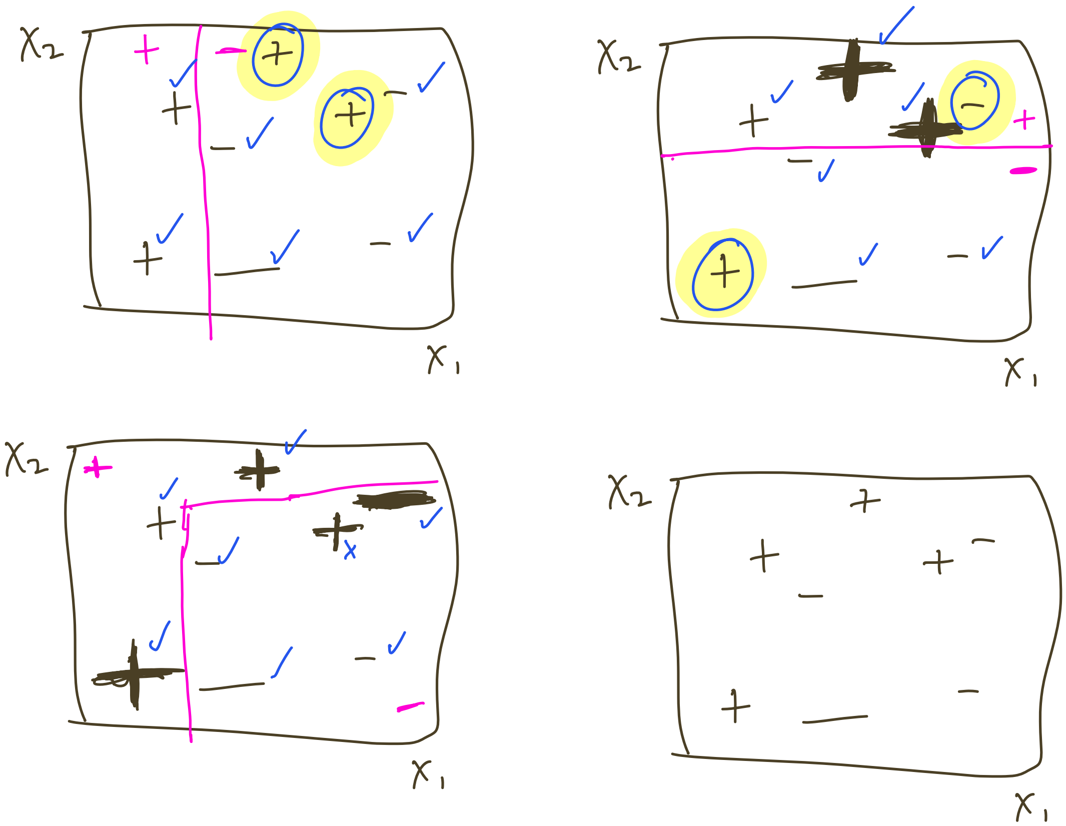
Adaboost: en cada iteración del algoritmo construimos un árbol de clasificación y le damos más peso a las observaciones donde el algoritmo se equivocó en la iteración pasada:



(x_1, y_1) (x_2, y_2) $\cdots$ (x_n, y_n) x_i: features, vector p-dimensional.

Primero, entrenamos un modelo $g^{[1]}(x)$ para predecir y.

Donde los pesos de cada observación son iguales.

Vemos $g^{[0]}$ en que puntos se equivocó y actualizamos los pesos de cada observación

Luego entrenamos $g^{[1]}$ para predecir y utilizando los pesos actualizados.

La predicción final:

$$\hat{f}_{Ada}(x_i) = \sum_{i=1}^{M} \alpha_m \underbrace{g^{[m]}(x_i)}_{\text{cada uno de los } 1,\dots,M \text{ modelos que entrenamos}}$$

pesos de acuerdo al desempeño del modelo.

M: # de modelos (árboles/iteraciones) del algoritmo.

En la iteración 1:

$w_i^{[1]} = \frac{1}{N}$, N es el número de observaciones.

Para $m = 2, \dots, M$:

Construimos un árbol utilizando los pesos $w_i^{[m]}$ para obtener $g^{[m]}(\cdot)$.

$$\text{error}^{[m]} = \frac{\sum_{i=1}^{N} w_i^{[m]} \mathbb{1}\{g_i^{[m]}(x_i) \neq y_i\}}{\sum_{i=1}^{N} w_i^{[m]}}$$

$$\alpha^{[m]} = \log\left(\frac{1 - \text{error}^{[m]}}{\text{error}^{[m]}}\right)$$

$$\tilde{w}_i^{[m+1]} = w_i^{[m]} \cdot \exp\left(\underbrace{\alpha^{[m]} \mathbb{1}\{g^{[m]}(x_i) \neq y_i\}}_{\text{esto es } =1 \text{ si el algoritmo predijo correctamente!}}\right)$$

Si el algoritmo $g^{[m]}$ predijo erróneamente para x_i, los pesos de la observación i van a aumentar.

$$w_i^{[m+1]} = \frac{\tilde{w}_i^{[m+1]}}{\sum_{i=1}^{N} \tilde{w}_i^{[m+1]}}$$

Gradient boosting (Xgboost):

$(x_1, y_1), \dots (x_N, y_N)$

Lo primero que hacemos es entrenar un modelo $f^{[1]}$ para predecir y.

error:

$y_1 = 0.9 \qquad f^{[1]}(x_1) = 0.8 \qquad 0.8 - 0.9 = -0.1$

$y_2 = 1.4 \qquad f^{[1]}(x_2) = 1.5 \longrightarrow 1.5 - 1.4 = 0.1$

$y_3 = 6.2 \qquad f^{[1]}(x_3) = 6.0 \qquad 6.0 - 6.2 = -0.2$

$\vdots \qquad \vdots$

Supongamos que entrenamos un modelo $f^{[2]}$ para predecir el negativo de los errores.

$\text{neg-error}_1 = 0.1$

$\text{neg-error}_2 = -0.1 \longrightarrow$ entreno $f^{[2]}(\cdot)$

$\text{neg-error}_3 = 0.2$

$\vdots$

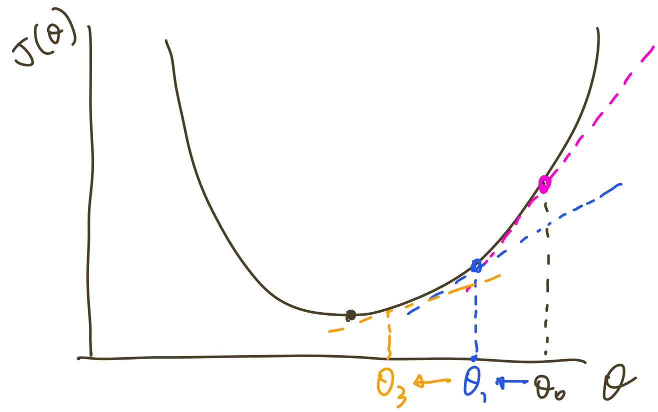
Supongamos que este modelo predijo perfectamente el negativo de los errores.

$$f^{[1]}(x_1) + f^{[2]}(x_1) = 0.8 + 0.1 = 0.9 = y_1$$

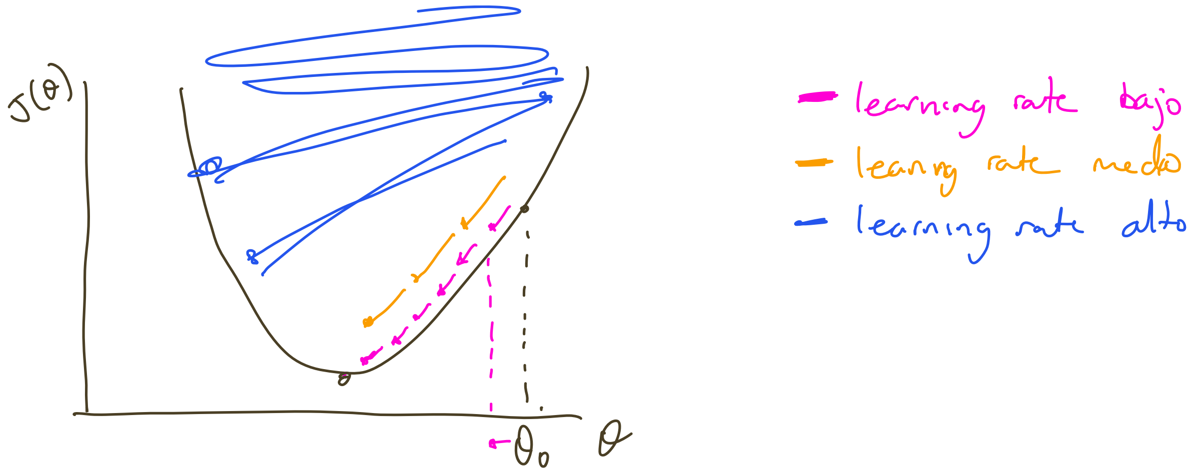
$$f^{[1]}(x_2) + f^{[2]}(x_2) = 1.5 + (-0.1) = 1.4 = y_2$$

$\vdots$

Algoritmos de gradient descent:



$$\theta_{i+1} = \theta_i - \overbrace{\rho}^{\text{learning rate}} \frac{\partial J(\theta)}{\partial \theta}$$



Función de pérdida cuadrática:

$$J(y, f(x)) = \sum_{i=1}^{N} \frac{(f(x_i) - y_i)^2}{2}$$

$$\frac{\partial J}{\partial f(x_i)} = \underbrace{f(x_i) - y_i}_{\text{error del modelo}}$$

En cada iteración, vamos a entrenar un modelo $g^{[m]}$ para predecir el negativo del error del algoritmo en la iteración anterior: $f^{[m-1]}(x_i) - y_i$

$$f^{[m]}(x_i) = f^{[m-1]}(x_i) + g^{[m]} \approx f^{[m-1]}(x_i) - y_i$$

$$f^{[m]}(x_i) = f^{[m-1]}(x_i) - 1 \cdot \left(f^{[m-1]}(x_i) - y_i\right)$$

$$\approx f^{[m-1]}(x_i) + \underbrace{\left(y_i - f^{[m-1]}(x_i)\right)}_{\text{error de la iteración anterior}}$$

$$f^{[m]}(x_i) = f^{[m-1]}(x_i) - \rho \frac{\partial J(y, f^{[m-1]}(x_i))}{\partial f^{[m-1]}(x_i)}$$