

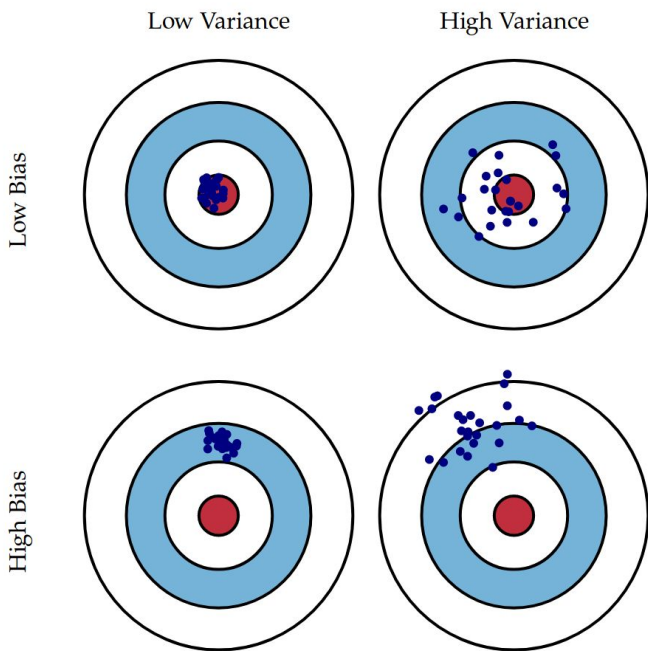
Gradient Boosting

A dark blue, abstract shape that starts as a thin line on the left and expands into a large, solid area on the right, resembling a stylized wave or a modern graphic element.

Índice

1. Equilibrio entre sesgo y varianza
2. Boosting
3. Repaso del descenso del gradiente
4. Gradient Boosting
5. Gradient Boosting para Regresión
6. Gradient Boosting para Clasificación
7. Hiperparámetros
8. CatBoost

1. Equilibrio entre sesgo y varianza



El **error** de un modelo se puede expandir en error por **sesgo**, **varianza** y **error irreducible**.

Los **árboles de decisión** tradicionales adolecen de tener mucha varianza, para estos casos se diseñaron las técnicas de ensamble.

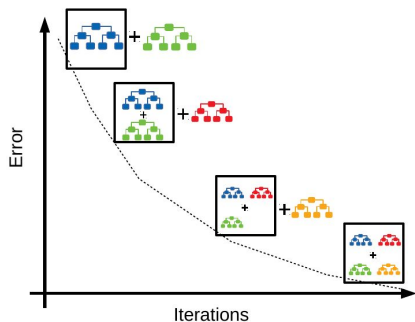
Algunas técnicas de ensamble:

- Familia **Bagging**
- Familia **Boosting**
- Familia **Stacking**
- ...

2. Boosting

Algunas ideas sobre Boosting:

- **No** es un **algoritmo** específico de Machine Learning.
- Es un concepto que puede **aplicarse** a **diferentes modelos** de ML, por lo que se le conoce como meta-algoritmo
- Es un **meta-algoritmo** de ensamble usado para **convertir** varios “**weak learners**” en un “**strong learner**”:
 - Weak learner: es un algoritmo de ML que tiene un poder predictivo levemente mejor que el azar.
 - Strong learner: es un algoritmo que puede ser “tuneado” para alcanzar un buen poder predictivo.



$$\hat{y}_i = F_M(x_i) = \sum_{m=1}^M h_m(x_i)$$

weak learner

Aplicado a árboles de decisión

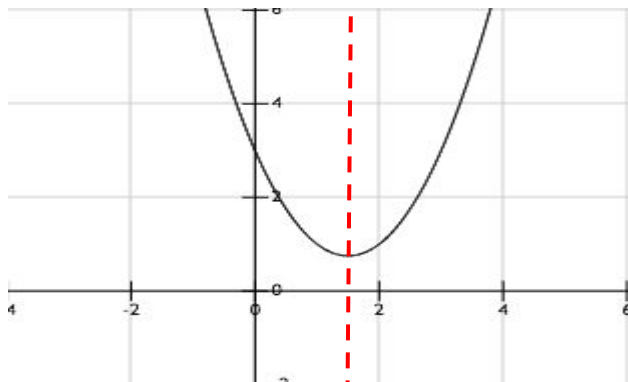
Pasos:

1. **Aprender iterativamente** una serie de “weak” **models** es muestras de los datos,
2. Darle un **peso** a cada “weak” predicción de acuerdo al **desempeño** de cada “weak learner”
3. **Combinar** todas las **predicciones ponderadas** para obtener una única predicción.

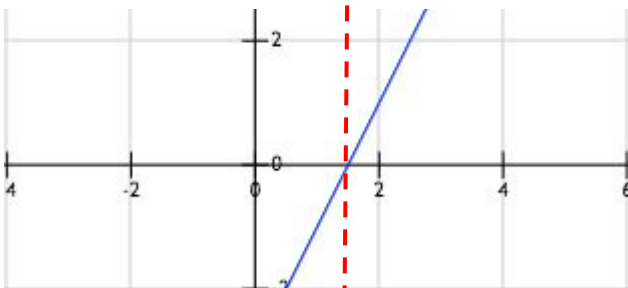
Por lo tanto:

- La idea de boosting es ir entrenando modelos de manera **secuencial** sobre los **errores** que van dejando los modelos, así cada **modelo** va **mejorando** donde el modelo anterior erró.
- En vez de entrenar un gran árbol de decisión profundo, que probablemente **sobreajuste**, boosting usa una manera de **aprender despacio**, sin prisa.

3. Repaso del descenso del gradiente



Función de
costo



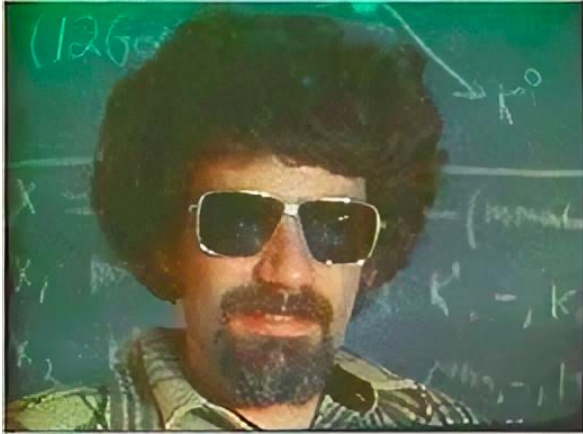
Función
derivada de la
función de
costo

Descenso del Gradiente

$$a_{n+1} = a_n - \gamma \nabla f(a_n)$$

$$f(a_n) = \frac{1}{n} \sum_{i=1}^n (y_i - (a_0 + a_1 x_i))^2$$

4. Jerome H. Friedman: el Walter White de GB



“La evidencia empírica muestra que tomar muchos pasos pequeños en la dirección correcta resulta en una mejor predicción sobre una muestra de validación”

La idea principal del algoritmo

En el **descenso** del **gradiente** que vimos anteriormente, teníamos una **función** de **pérdida** que **dependían** del **valor observado** y del valor **predicho**, a su vez el valor predicho dependía de una **función paramétrica**, por lo que el descenso del gradiente lo que hacía era ir **moviendo** los valores de los **parámetros** con el objetivo de que su **derivada** fuese igual a **0**.

En el caso de **gradient boosting** va a ser similar, simplemente en este caso la predicción **no depende** de una **función paramétrica**, sino de los **árboles de decisión**, por lo que lo que voy a ir modificando no son los parámetros sino la función (gracias al boosting). La idea, por lo tanto, es entrenar una función que minimice la pérdida esperada.

$$\hat{F} = \operatorname{argmin}_F E_{x,y}[L(y, F(x))]$$

Gradient Boosting Trees: algoritmo original

Este es el algoritmo original:

1. Initialize $f_0(x) = \arg \min_{\gamma} \sum_{i=1}^N L(y_i, \gamma)$.

2. For $m = 1$ to M :

(a) For $i = 1, 2, \dots, N$ compute

$$r_{im} = - \left[\frac{\partial L(y_i, f(x_i))}{\partial f(x_i)} \right]_{f=f_{m-1}}.$$

(b) Fit a regression tree to the targets r_{im} giving terminal regions R_{jm} , $j = 1, 2, \dots, J_m$.

(c) For $j = 1, 2, \dots, J_m$ compute

$$\gamma_{jm} = \arg \min_{\gamma} \sum_{x_i \in R_{jm}} L(y_i, f_{m-1}(x_i) + \gamma).$$

(d) Update $f_m(x) = f_{m-1}(x) + \sum_{j=1}^{J_m} \gamma_{jm} I(x \in R_{jm})$.

3. Output $\hat{f}(x) = f_M(x)$.

5. Gradient Boosting para regresión

Ejemplo con el que vamos a trabajar:

#	Altura (m)	Color favorito	Género	Peso (kg)
1	1.6	Azul	Hombre	88
2	1.6	Verde	Mujer	76
3	1.5	Azul	Mujer	56
4	1.8	Rojo	Hombre	73
5	1.5	Verde	Hombre	77
6	1.4	Azul	Mujer	57

Pasos para regresión (least squares)

Paso 0: Tomamos una función de pérdida diferenciable:

$$L = \frac{1}{2} \sum_{i=1}^n (y_i - \hat{y}_i)^2 \quad L' = \frac{\partial L(y_i, F(x_i))}{\partial F(x_i)} = - \sum_{i=1}^n (y_i - \hat{y}_i)$$

función más típica y cómoda

Paso 1: Partimos construyendo una sola hoja, que representa una predicción inicial para la variable objetivo en todas las observaciones. Esta primera predicción es el valor promedio de la variable objetivo.

$$F_0(x) = \operatorname{argmin}_{\gamma} \sum_{i=1}^n L(y_i, \gamma) = \bar{y}$$

#	Altura (m)	Color favorito	Género	Peso (kg)	Predicción $m=0$ ($F_0(x)$)
1	1.6	Azul	Hombre	88	71.2
2	1.6	Verde	Mujer	76	71.2
3	1.5	Azul	Mujer	56	71.2
4	1.8	Rojos	Hombre	73	71.2
5	1.5	Verde	Hombre	77	71.2
6	1.4	Azul	Mujer	57	71.2

Paso 2: Repetir $m = 1$ hasta M :


Paso 2.1: Calculamos los “pseudo-residuos” (*pseudo* porque los residuos son solo el valor observado menos el predicho, y aquí dependen de la función de pérdida que se defina):

$$r_{im} = - \left[\frac{\partial L(y_i, F(x_i))}{\partial F(x_i)} \right]_{F(x)=F_{m-1}(x)} \quad \text{para } i = 1, \dots, n$$

*Esta derivada es el gradiente
por el que se llama Gradient
Boosting*

*$F(x_i)$ es una función (árbol de
decisión) que nos da los valores
predichos.*

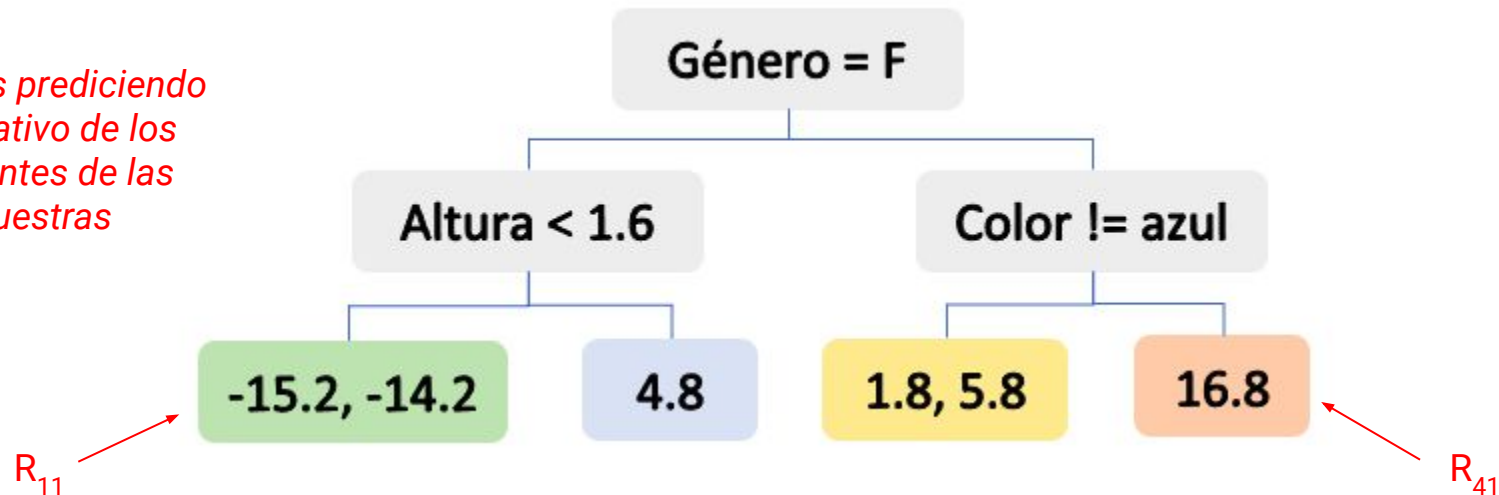
Residuos de la derivada de la función de pérdida



#	Altura (m)	Color favorito	Género	Peso (kg)	Predicción $m=0$ ($F_0(x)$)	Pseudo residuos $m=1$
1	1.6	Azul	Hombre	88	71.2	$88 - 71.2 = 16.8$
2	1.6	Verde	Mujer	76	71.2	$76 - 71.2 = 4.8$
3	1.5	Azul	Mujer	56	71.2	$56 - 71.2 = -15.2$
4	1.8	Rojo	Hombre	73	71.2	$73 - 71.2 = 1.8$
5	1.5	Verde	Hombre	77	71.2	$77 - 71.2 = 5.8$
6	1.4	Azul	Mujer	57	71.2	$57 - 71.2 = -14.2$

Paso 2.2: Entrenamos un “weak learner” (un árbol CART con poca profundidad) usando nuestras variables independientes y los pseudo-residuos (ya no usamos la variable Peso) del paso anterior, y creamos las hojas terminales (R_{jm}):

*Estamos prediciendo
el negativo de los
gradientes de las
muestras*



#	Altura (m)	Color favorito	Género	Peso (kg)	Pseudo residuos m=1
1	1.6	Azul	Hombre	88	16.8
2	1.6	Verde	Mujer	76	4.8
3	1.5	Azul	Mujer	56	-15.2
4	1.8	Rojo	Hombre	73	1.8
5	1.5	Verde	Hombre	77	5.8
6	1.4	Azul	Mujer	57	-14.2

Paso 2.3: Para cada hoja del árbol, calculamos “ γ ”, por lo que calculamos la salida de cada una de las hojas del árbol. Para calcular la salida de la hoja hay que realizar una optimización (minimizar la suma de las pérdidas L_m dado el previo ensamble F_{m-1}), cuyo resultado, dada la función de pérdida previamente escogida, es el promedio de las observaciones de la hoja

$$\gamma_m = \operatorname{argmin}_{\gamma} \sum_{x_i \in R_{j,m}} L(y_i, F_{m-1}(x_i) + \gamma)$$

-15.2, -14.2
= -14.7

4.8

1.8, 5.8
= 3.8

16.8

Paso 2.4: Calculamos la predicción para registro. Aquí es donde viene la parte de boosting

$$F_m(x) = F_{m-1}(x) + \nu \sum_{j=1}^{J_m} \gamma_{jm} I(x \in R_{jm})$$

predicción anterior

tasa de aprendizaje

árbol que acabamos de entrenar

En nuestro ejemplo, la tasa de aprendizaje es 0.1

#	Altura (m)	Color favorito	Género	Peso (kg)	Pseudo residuos m=1	Predicción m=1 ($F_1(x)$)
1	1.6	Azul	Hombre	88	16.8	$71.2 - 0.1 * 16.8 = 72.9$
2	1.6	Verde	Mujer	76	4.8	$71.2 - 0.1 * 4.8 = 71.7$
3	1.5	Azul	Mujer	56	-15.2	$71.2 - 0.1 * -14.7 = 69.7$
4	1.8	Rojo	Hombre	73	1.8	$71.2 - 0.1 * 3.8 = 71.6$
5	1.5	Verde	Hombre	77	5.8	$71.2 - 0.1 * 3.8 = 71.6$
6	1.4	Azul	Mujer	57	-14.2	$71.2 - 0.1 * -14.7 = 69.7$

6. Gradient Boosting para clasificación (log-loss)

Para un problema de clasificación, el procedimiento es **similar** pero un poco **más complejo**. **Cambia** la **función de pérdida** y el hecho de que los árboles **predicen** en **términos** de **log(odds)**.

La función de pérdida que comúnmente se utiliza es la '**negative binomial log-likelihood**'.

$$L = - \sum y_i \log(\hat{y}_i) + (1 - y_i) \log(1 - \hat{y}_i)$$

$$\hat{y}_i = \frac{1}{1 + e^{\log(odds)}}$$

7. Hiperparámetros básicos

Hiperparámetros de la versión **original** básica:

- **Número de árboles:** número de árboles que se van a construir.
- **Hiperparámetros típicos en la construcción de árboles:** máxima profundidad, máximo número de hojas, etc.

Hiperparámetros de “**regularización**” introducidos después por el autor:

- **Tasa de aprendizaje:** previene del sobreajuste, es decir que la varianza sea muy alta. Tiene un valor entre 0 y 1.
- **Subsampling (Stochastic Gradient Boosting):** al momento de generar los árboles se puede tomar una submuestra aleatoria sin reemplazo.

Los mejores resultados se obtienen utilizando una tasa de aprendizaje pequeña (<0.1) y una tasa de subsampling de entre 0.5 y 0.8. Obviamente todo depende del dataset objetivo.

8. CatBoost – Categorical Boosting



CatBoost es una librería desarrollada y mantenida por **Yandex** (principal buscador en Rusia). Es totalmente **open-source**, lo que permite su uso gratuito para cualquier propósito.

CatBoost presenta algunas ventajas y mejoras frente al algoritmo original de **Gradient Boosting**:

- Entrenamiento de **árboles simétricos**:
 - Menor impacto en el cambio de parámetros
 - menor posibilidad de overfitting
 - menor tiempo de entrenamiento y predicción



- **Tratamiento** automático de **variables categóricas**:
 - One-hot encoding automático para variables categóricas con pocos niveles,
 - Transformación a variables numéricas de variables categóricas con muchos niveles (Ordered TS),
 - Cálculo automático de interacciones entre variables categóricas que sean significantes.
- Implementación de '**ordered boosting**', que es una alternativa al clásico boosting. En muestra pequeñas gradient boosting sobreajusta muy rápido, sin embargo Catboost funciona bien.
- Se puede entrenar fácilmente en la **GPU**.
- Soporta **valores nulos**.
- Buenas **visualizaciones**, herramientas para graficar en Python
- Puedes escribir tu propia **función** de **pérdida**.
- Calcula la importancia de variables.

Principales hiperparámetros



Depth



Controla la **profundidad de los árboles de decisión** creado en cada iteración. A mayor profundidad, mayor capacidad de ajuste pero más probable el sobreajuste



L2 Leaf
Regularization



Controla la **regularización** que se aplica a los valores calculados en los árboles de decisión contruidos



Random
Strength



Controla la **intensidad de la aleatoriedad** aplicada en el cálculo del valor óptimo para realizar el **split en los árboles de decisión**. A mayor valor, más aleatoriedad y menos probabilidad de elegir el valor óptimo, pero menos probabilidad de sobreajuste



Bagging
Temperature



Controla la **distribución con la que se realiza *resampling*** de la muestra en el proceso de construcción de los árboles de decisión



Learning Rate



Parámetro que **multiplica el gradiente calculado para entrenar la siguiente iteración** del árbol de decisión. Influye en controlar el sobreajuste del modelo y en el tiempo requerido para el entrenamiento

GRACIAS