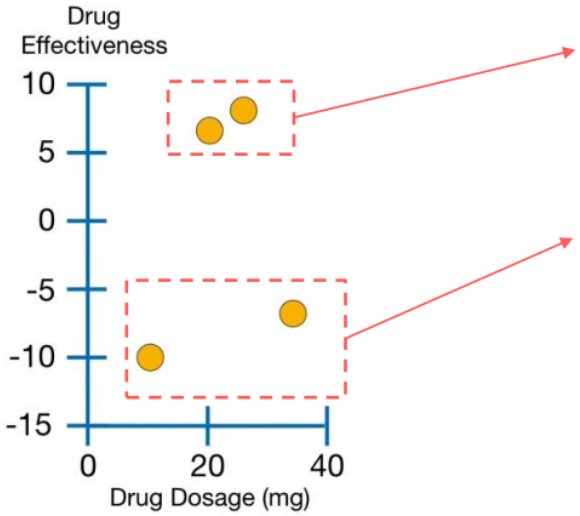
XGBoost

XGBoost: eXtreme Gradient Boost

XGBoost fue diseñado para Big Data, es decir para conjuntos de datos grandes y complejos.

Sin embargo a fines de entender el algoritmo principal lo usaremos con un conjunto de datos simple (y para el caso de regresión)

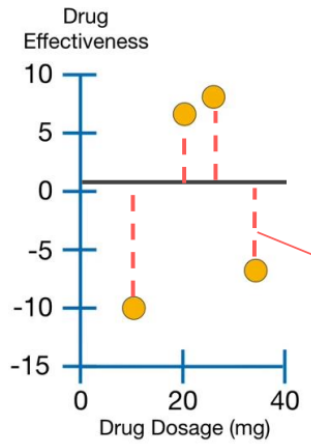
Se puede usar para clasificacion y regresion



Estas dosis, han tenido un efecto **positivo** en la salud de los pacientes. Por eso están sobre el cero.

Estas dosis, han tenido un efecto **negativo** en la salud de los pacientes. Por eso están por debajo de cero.

Queremos predecir, para una dosis dada en miligramos, que tan efectiva es en los pacientes



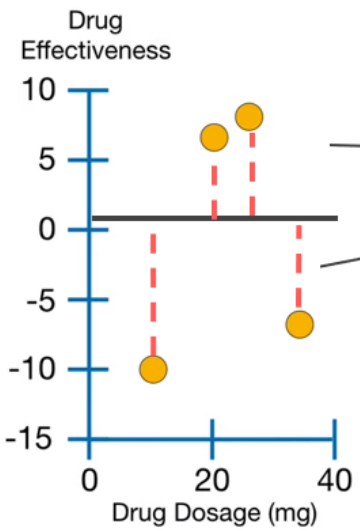
Primer paso: Hacer una predicción inicial.
Esta predicción puede ser cualquier valor.
Pero por defecto se toma 0.5

0.5

Esta predicción inicial es igual, ya sea que esté haciendo regresión o clasificación

Esta observación inicial se corresponde a esta linea negra

Estas líneas muestran los residuos, es decir el error que comete el estimador para cada medición.



Segundo paso: Construir un árbol para los residuos.
Este árbol es diferente a los usados por Gradient Boost.
Primero se crea un nodo hoja.
Y se ponen allí todos los residuos

-10.5, 6.5, 7.5, -7.5

NOTA: Hay varias formas de construir el árbol utilizado por **XGBoost**. Esta es la más común.

Tercer paso: Calcular el ***Similarity Score***, para los residuos

-10.5, 6.5, 7.5, -7.5

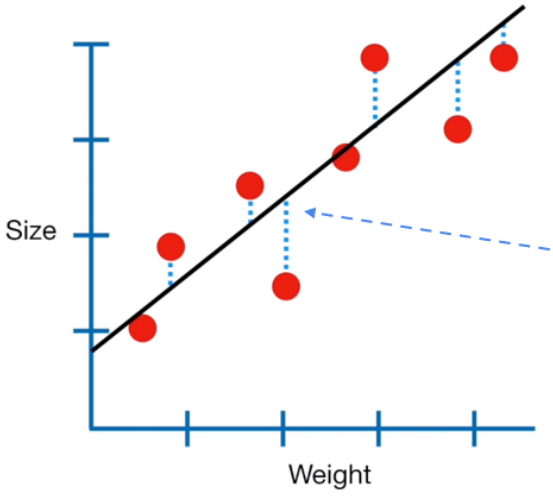
$$\text{Similarity Score} = \frac{(\text{Suma de residuos})^2}{\text{Cantidad de residuos} + \lambda}$$

λ (lambda): es un parámetro de regularización.

Regularización

Imaginemos que tenemos una serie de datos y queremos encontrar una forma de predecir valores futuros.

Utilizaremos en este caso, regresión lineal.



Finalmente obtendremos una ecuación como esta:

Size = 0.9 + 0.75 **Weight**

Punto de intersección del eje-y

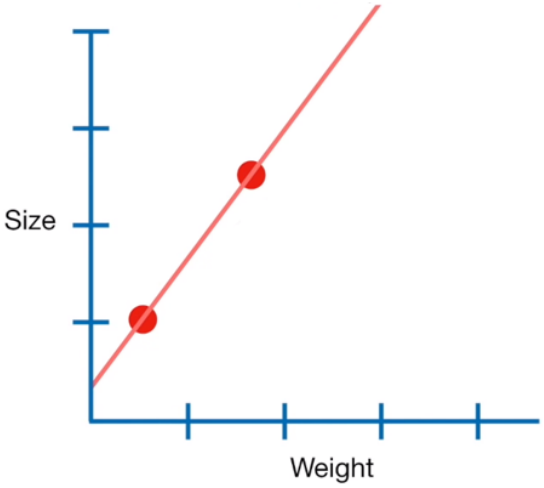
pendiente

Estos son los residuos o errores.

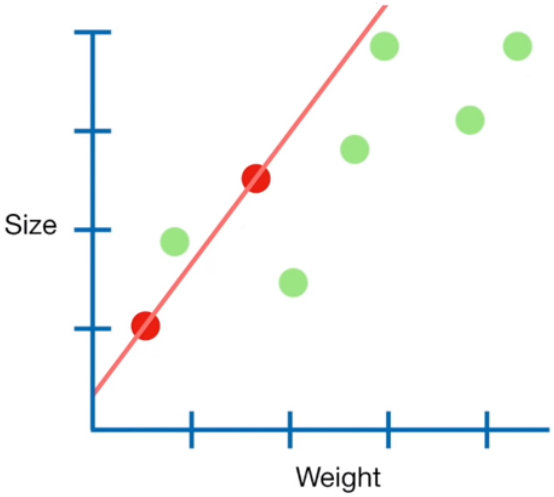
Si hay muchos puntos, podemos estar seguros de que la línea calculada será un buen estimador del tamaño.

¿Qué pasa si solo tenemos 2 puntos?

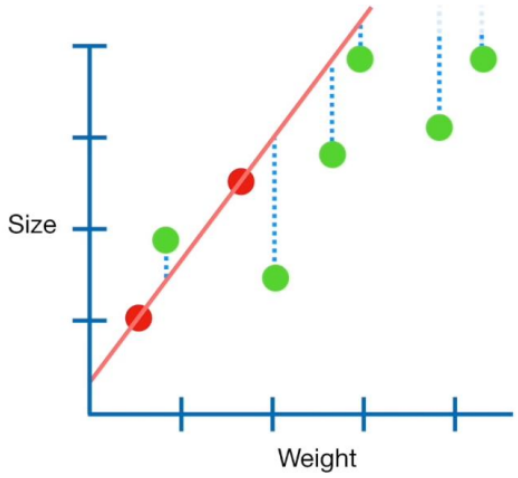
La línea calculada ajusta perfecto. No hay residuos. No hay errores.



Ahora mostramos en verde todos los puntos. Utilicemos al resto de los puntos, **los verdes**, como conjunto de prueba.

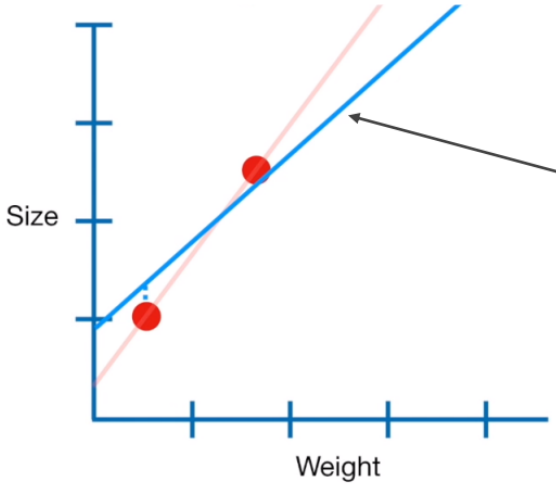


Este estimador no es bueno porque los residuos son grandes.



En el conjunto de pruebas, si hay residuos. Y estos son mucho mayores que en el primer caso, cuando entrenamos con todos los datos!!

Esto quiere decir que está nueva línea, este estimador tiene una varianza alta (***High Variance***).
Una **varianza alta** indica que los puntos de datos están muy separados de la media y entre sí.
Y el modelo está **sobreentrenado** (***overfit***).



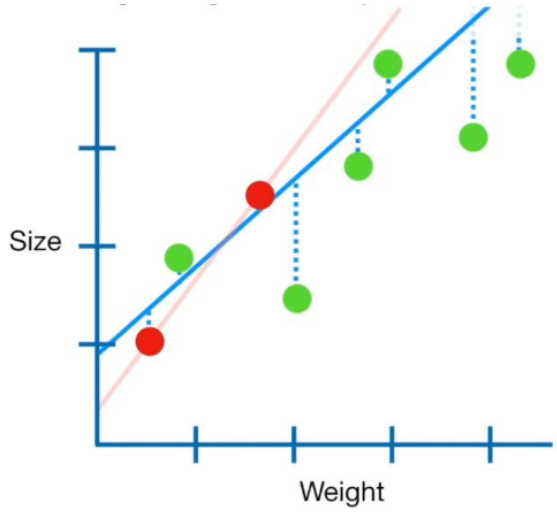
Una forma de solucionar este problema, es usando una variación de la regresión lineal llamada: **Ridge Regression**

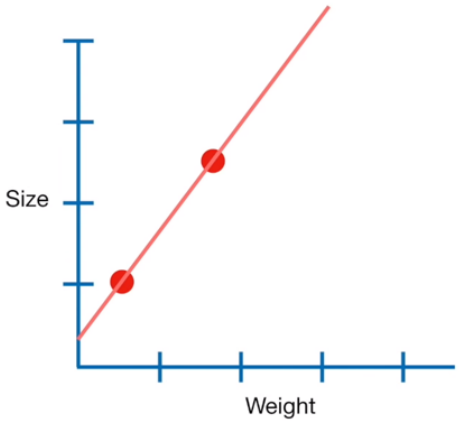
La idea es encontrar una línea que no ajuste tan bien

Para ello se va a introducir un pequeño sesgo o ***bias***, en los datos de entrenamiento.

El error de sesgo introducido en el estimador, hará que caiga el error por la varianza de forma mucho más abrupta en el conjunto de pruebas.

Aumentara el error de bias.





Cuando usamos **Regresión Lineal**, es decir cuadrados mínimos, lo que estamos haciendo es minimizando **la suma del cuadrado de los residuos.**
En cambio en **Ridge Regression** estamos minimizando:

la suma del cuadrado de los residuos + λ * (pendiente)^2

Y λ determina qué tan severa es esta penalización

Esto representa una penalización al método tradicional.

Según mínimos cuadrados, tenemos que:

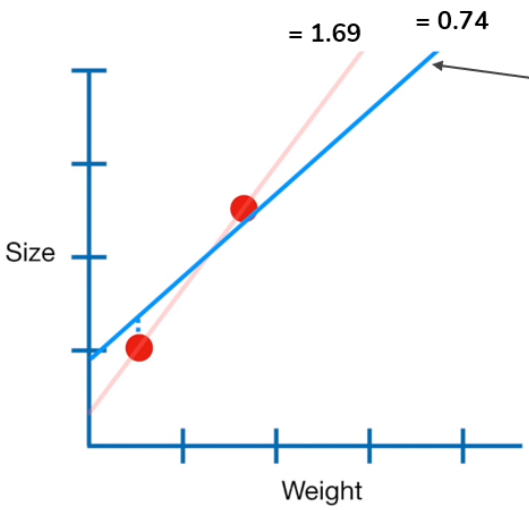
Size = 0.4 + 1.3 * **Weight**

Suma de cuadrados de residuos $= 0^2 + 0^2$

Pero usando Ridge Regression:

Suma de cuadrados de residuos + λ * (pendiente)^2 = $0 + 1 * (1.3)^2$ **= 1.69**

Asumimos $\lambda = 1$



Ridge Regression, seguirá ajustando hasta lograr una recta como la **azul**:

Size = 0.9 + 0.8 * **Weight**

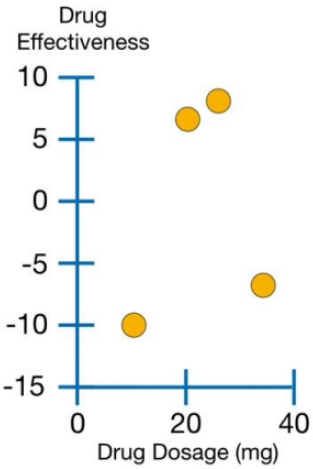
Suma de cuadrados de residuos = $0.3^2 + 0.1^2$

Suma de cuadrados de residuos + λ * (pendiente)^2

$0.3^2 + 0.1^2 + 1 * (0.8)^2$

$0.09 + 0.01 + 0.064$ = **0.74**

Volvamos a XGBoost y al calculo del Similarity Score



Tercer paso: Calcular el ***Similarity Score***, para los residuos

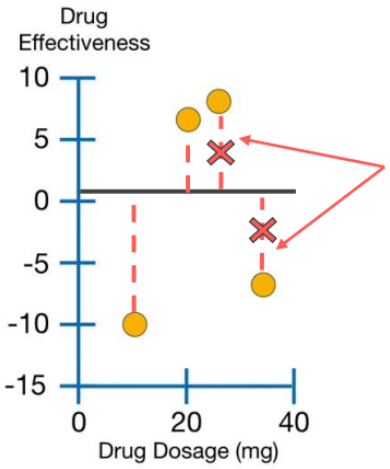
-10.5, 6.5, 7.5, -7.5

$$\text{Similarity Score} = \frac{(\text{Suma de residuos})^2}{\text{Cantidad de residuos} + \lambda}$$

λ (lambda): es un parámetro de regularización.

Con λ reducimos el error por varianza.
Pero por ahora dejemos $\lambda = 0$

Tercer paso: Calcular el ***Similarity Score***, para los residuos



Se cancelan mutuamente

-10.5, 6.5, 7.5, -7.5

$$\text{Similarity Score} = \frac{(-10.5 + 6.5 + 7.5 - 7.5)^2}{4 + 0}$$

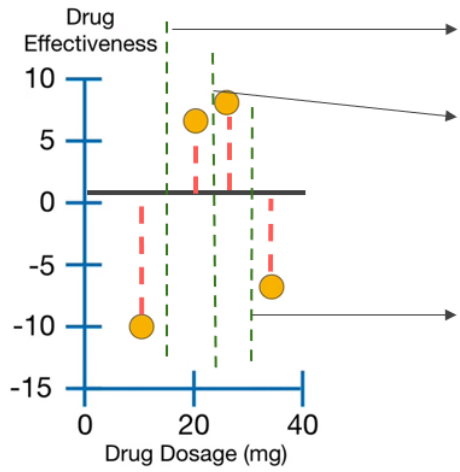
$$\text{Similarity Score} = \frac{(-4)^2}{4}$$

El ***Similarity Score*** para los residuos del nodo raíz es = **4**

Tenemos este nodo raiz:

-10.5, 6.5, 7.5, -7.5 ***Similarity Score = 4***

Cuarto paso: Tenemos que ver cuál será el siguiente nodo. Para ello vamos a calcular la **ganancia total**, según escojamos una opción u otra para partir el árbol.

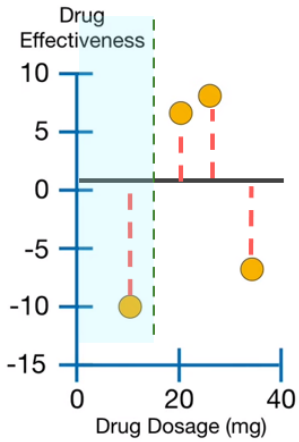


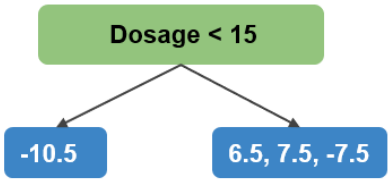
Opción 1, tomar como umbral la distancia intermedia entre las primeras 2 observaciones

Opción 2, tomar como umbral la distancia intermedia entre las segundas 2 observaciones

Opción 3, tomar como umbral la distancia intermedia entre últimas 2 observaciones

Opcion 1





Similarity Score = 4

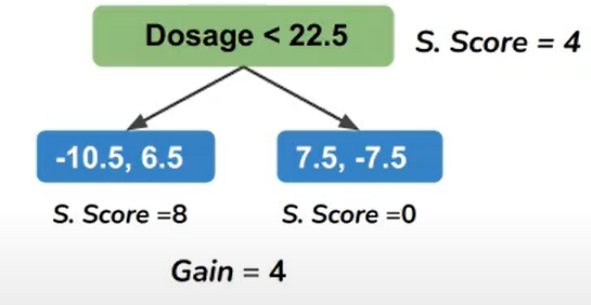
Similarity Score = $(-10.5)^2 / 1 =$ **110,25**

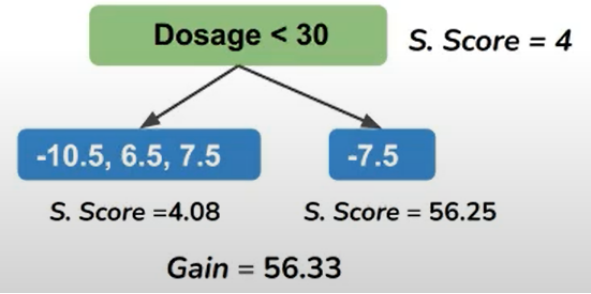
Similarity Score = $(6.5)^2 / 3 =$ **14.08**

Gain = $\text{Nodo\_Izquierda}_{\text{Similarity Score}} + \text{Nodo\_Derecha}_{\text{Similarity Score}} - \text{Raiz}_{\text{Similarity Score}}$

Gain = 110.25 + 14.08 - 4 = **120.33**

Exploramos todas las opciones



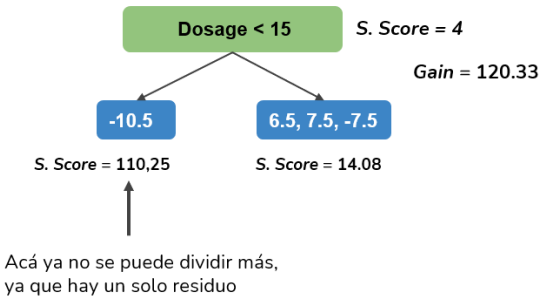


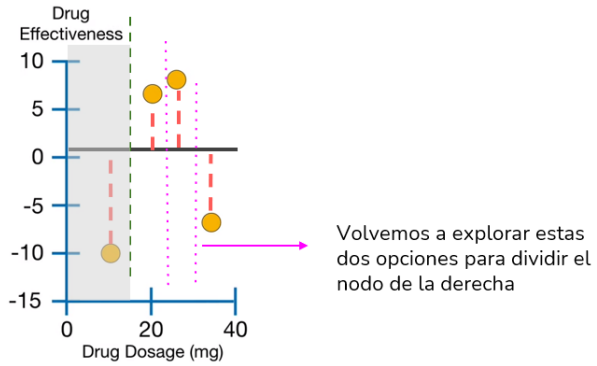
El primero es el que tiene una ganancia mayor

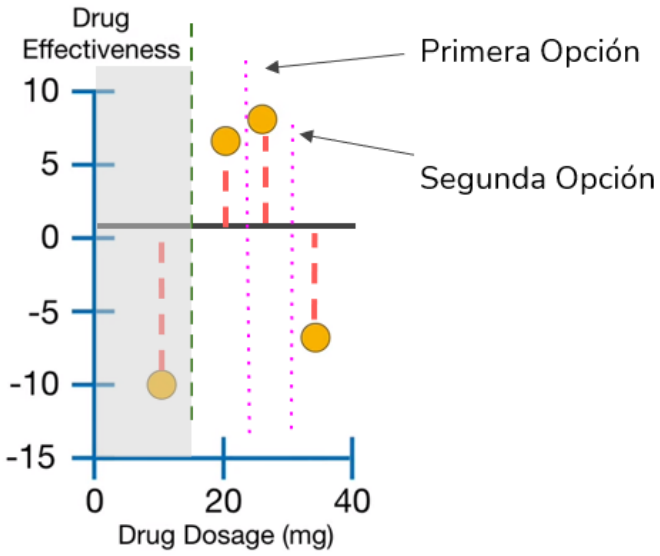
“Dosage < 15” es el umbral que mejor divide los residuos del árbol

Quinto paso:

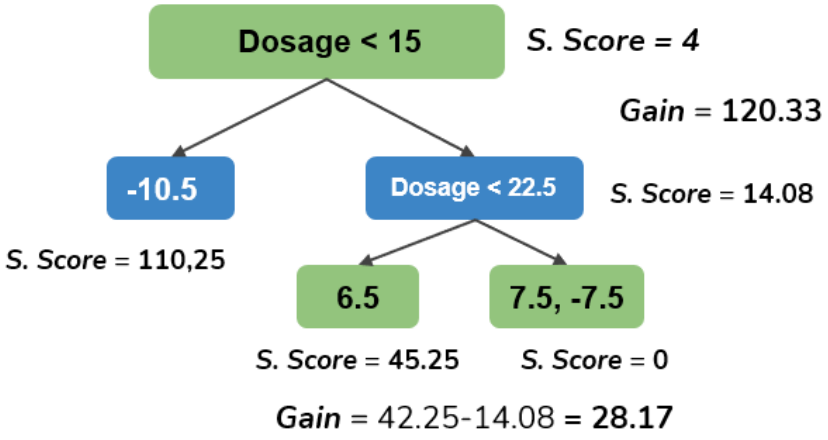
Repetimos el anterior hasta alcanzar la profundidad del árbol estipulada


Dosage < 15
S. Score = 4
Gain = 120.33
-10.5
6.5, 7.5, -7.5
S. Score = 110,25
S. Score = 14.08
Acá ya no se puede dividir más,
ya que hay un solo residuo

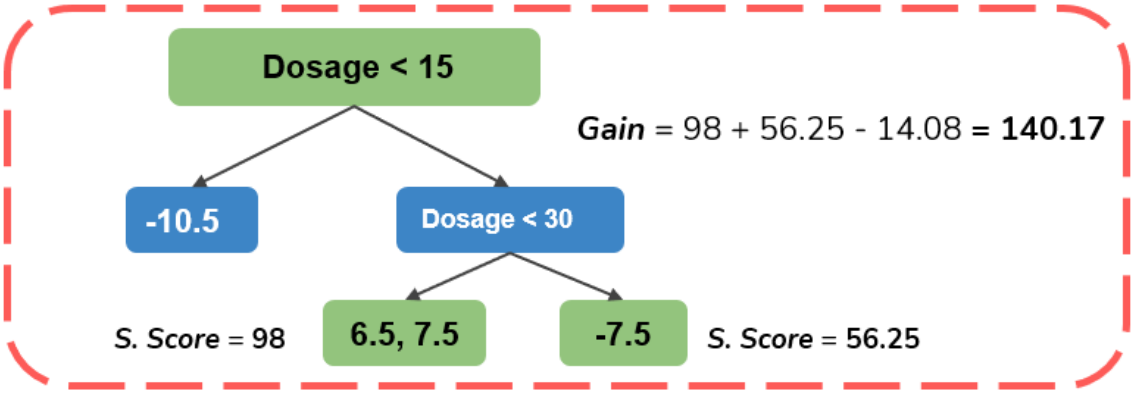

Drug
Effectiveness
10
5
0
-5
-10
-15
0
20
40
Drug Dosage (mg)
Volvemos a explorar estas
dos opciones para dividir el
nodo de la derecha


Drug
Effectiveness
10
5
0
-5
-10
-15
0
20
40
Drug Dosage (mg)
Primera Opción
Segunda Opción

1era Opción



2da Opción



Entonces este es el arbol

Cómo nuestra restricción era de 2 niveles de profundidad terminamos de generar el árbol.

Aunque por defecto XGBoost trabaja con 6 niveles

Sexto paso: poda

● Elegimos un número al azar llamado gamma (γ)
Por ejemplo: 130

● Calculamos la diferencia entre el Gain, del nodo más bajo y gamma

Gain - γ = 140.17 - 130 = 10.17

- Sí la diferencia es < 0 ⇒ removemos el nodo
- Sino, el nodo se queda y se terminó la poda

En este caso no se poda.

¿Qué hubiera pasado si elegíamos un valor más alto?

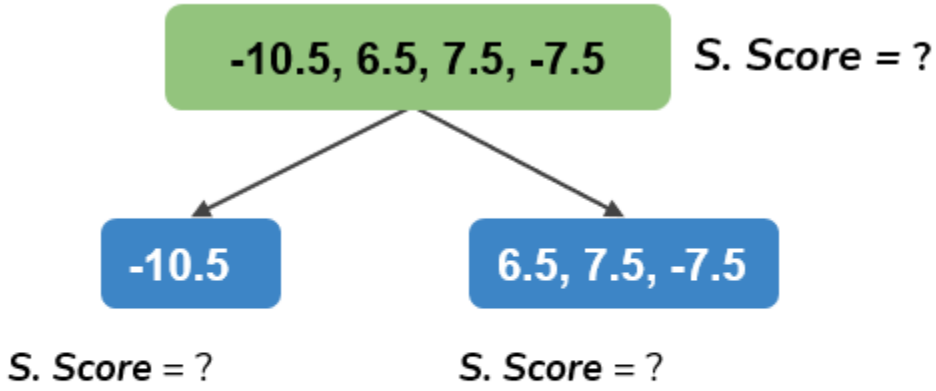
- Elegimos gamma (γ) igual a 150
- Calculamos la diferencia entre el Gain, del nodo más bajo y gamma
  - Gain - γ = 140.17 - 150 = -9,83
- Como la diferencia es < 0 ⇒ removemos el nodo
- La poda continua con gamma (γ) igual a 150
- Calculamos la diferencia de nuevo
  - Gain - γ = 120.33 - 150 = -29,67
- Como la diferencia es < 0 ⇒ removemos el nodo

⇒ Solo queda la estimación inicial 0,5

Por defecto gamma es igual a 0.
Cuanto más alto más conservador es el algoritmo

Séptimo paso:

Volvemos a calcular el árbol (repite paso 3), solo que esta vez usamos lambda λ igual a 1 al calcular el Similarity Score

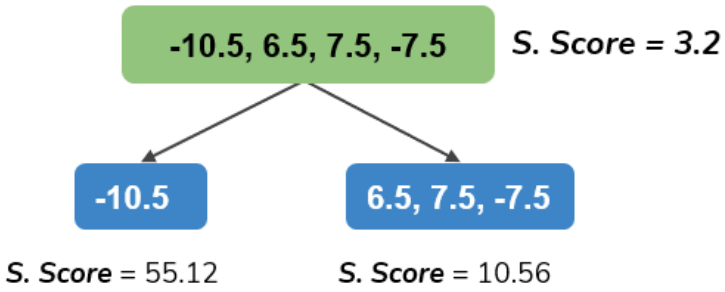


$$\text{Similarity Score} = \frac{(\text{Suma de residuos})^2}{\text{Cantidad de residuos} + \lambda}$$

$$S1 = \frac{(-10.5+6.5)^2}{4+1} = \frac{4^2}{5} = 3.2$$

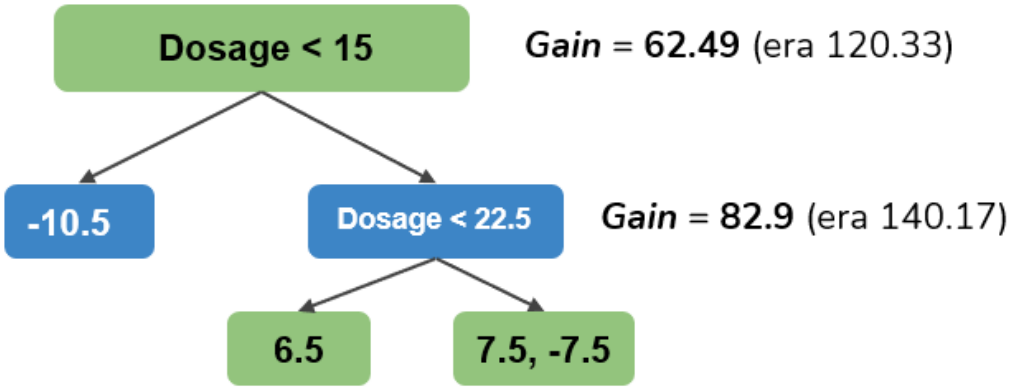
$$S2 = \frac{(-10.5)^2}{1+1} = \frac{110.25}{2} = 55.125$$

$$S3 = \frac{6.5^2}{3+1} = \frac{42.25}{4} = 10.5625$$



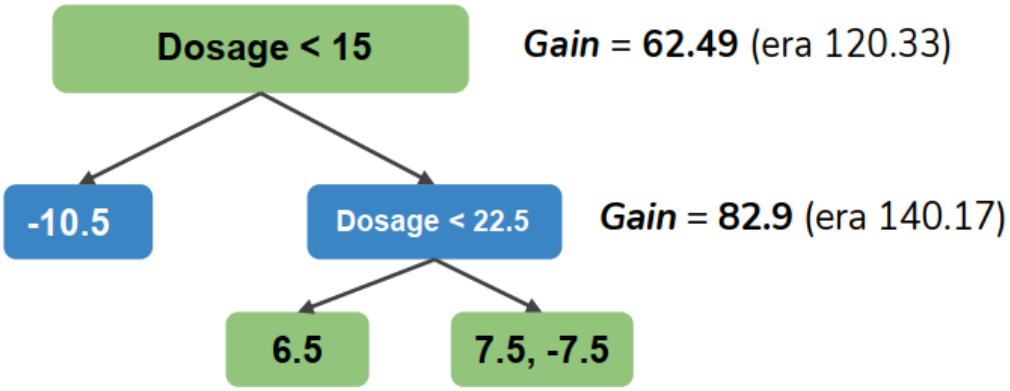
- Con lambda > 0, los Similarity Scores son mucho más chicos
- Esta disminución es proporcional a la cantidad de residuos en el nodo
  - El nodo raíz pasó de 4 a 3.2 (se redujo un 20%)
  - El nodo hoja izquierdo pasó de 110 a 55, un 50%

Calculamos ahora la ganancia de cada nodo. Es decir, el Gain



Tener lambda > 0 hace que el **Gain** también sea menor

Podamos (paso quinto nuevamente)



Cómo habíamos elegido gamma (γ) igual a 130, se poda todo el árbol.

La ganancia es mas pequeña, entonces se poda y este arbol se va.

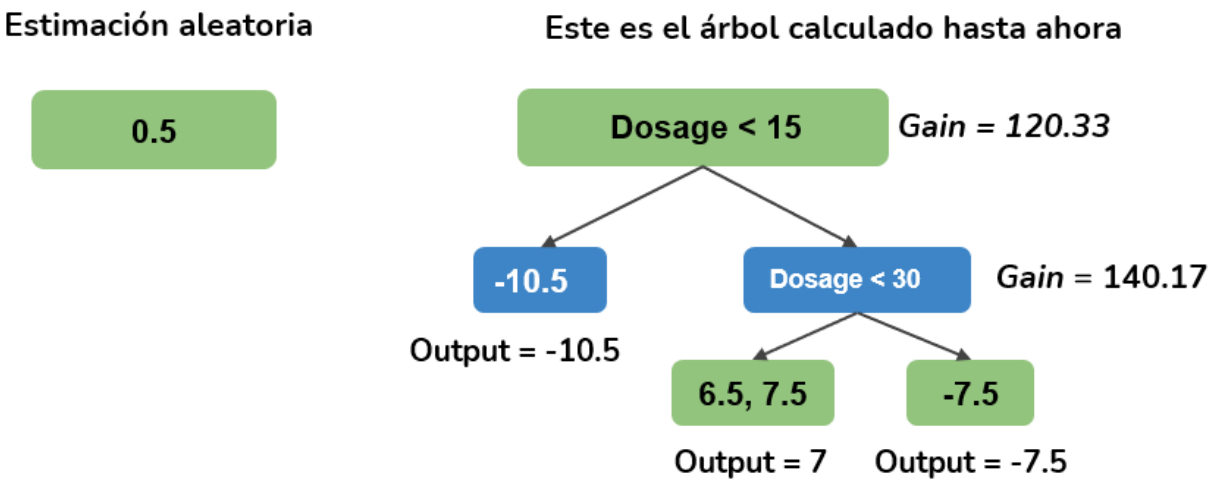
Como ya podes ese arbol, no sigo avanzando con arboles con λ mas grandes

● Cuando $\lambda > 0$ es más probable tener que podar un árbol, ya que los Gain calculados son menores.

● Aún eligiendo γ = 0 podemos tener que podar nodos, ya que la ganancia (Gain) puede ser negativa.

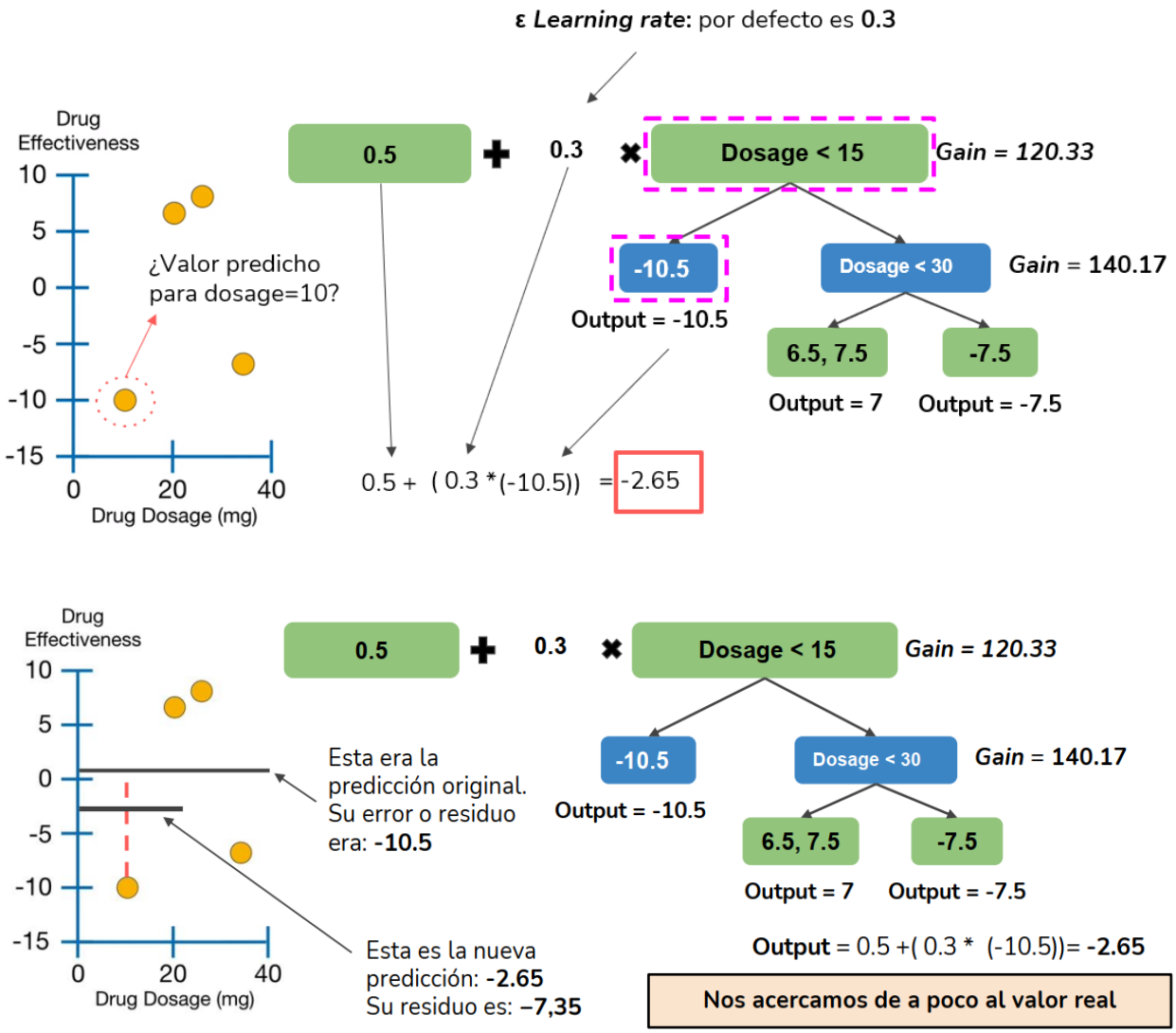
● λ=1, previene el sobreentrenamiento o sobreajuste del modelo en el conjunto de entrenamiento

Calcular la respuesta de XGBoost



$$\textbf{Output} = \frac{\text{Suma de residuos}}{\text{Número de residuos} + \boldsymbol{\lambda}}$$

Tomamos $\boldsymbol{\lambda}$ =0 ya que es el valor por defecto

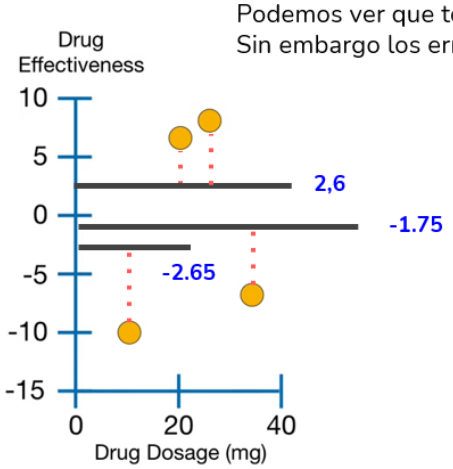


Deja un residuo mas chico.

La idea es avanzar poco a poco al valor real.

Nuevo paso: Calculamos todos los residuos utilizando el árbol hasta acá

Podemos ver que todas las estimaciones mejoran respecto de la original (0.5).
Sin embargo los errores (residuos) siguen siendo altos.



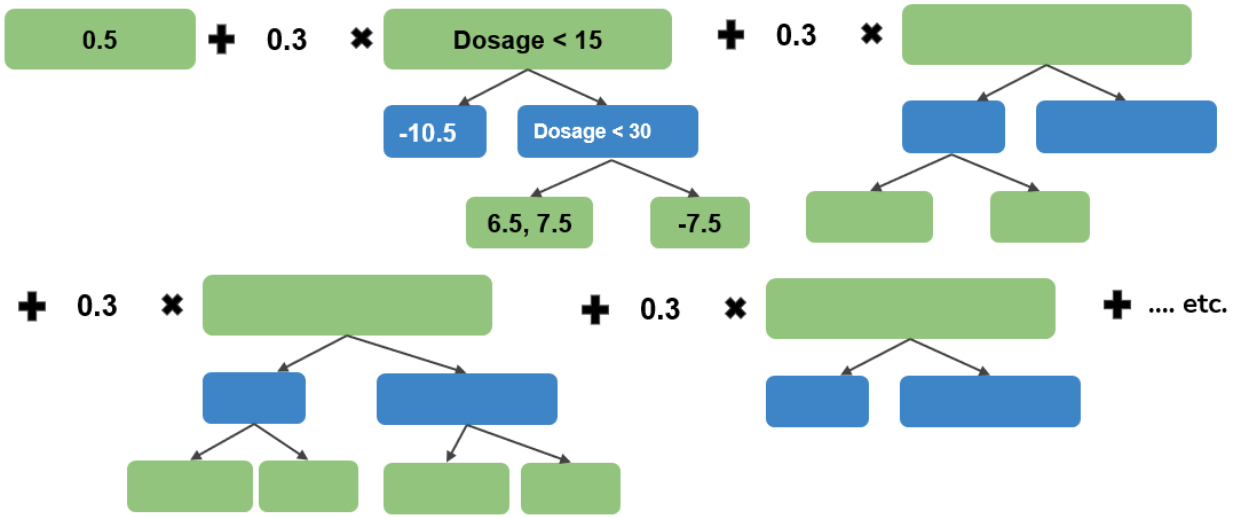
Con estos nuevos residuos, construimos un nuevo árbol.
Repetimos todo, desde el paso 2.

Con el nuevo árbol, calculamos la salida de cada elemento y luego los residuos.

Construimos otro árbol.

Seguimos hasta que los residuos son prácticamente cero o bien alcanzamos el número máximo de árboles predefinido

XGBoost: estructura final



Cada arbol va a ser diferente y cada uno va a trabajar sobre los residuos (igual que en Gradient Boost). No trabajamos sobre los valores que nosotros queremos calcular, sino sobre el error que estamos cometiendo.

💡 **Siempre hay que podar!**
Aunque sea con gamma 0 (ej si queda ganancia negativa, hay que podar)