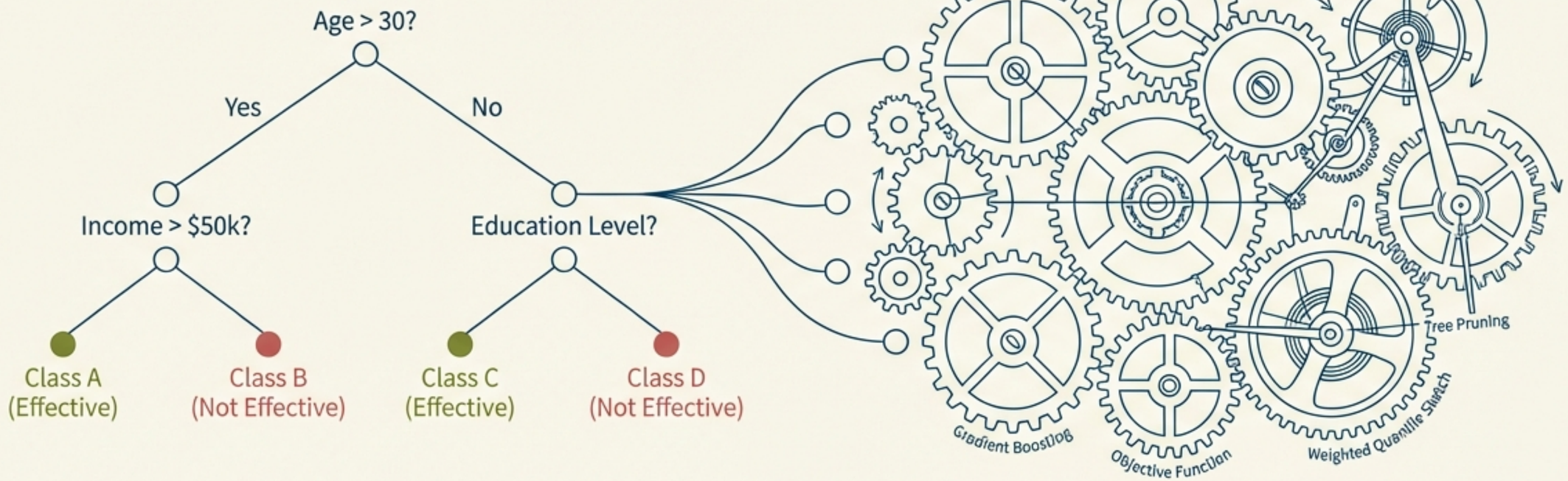
Deconstructing XGBoost: A Step-by-Step Build for Classification

An intuitive guide to how XGBoost constructs a classification model, one decision at a time.

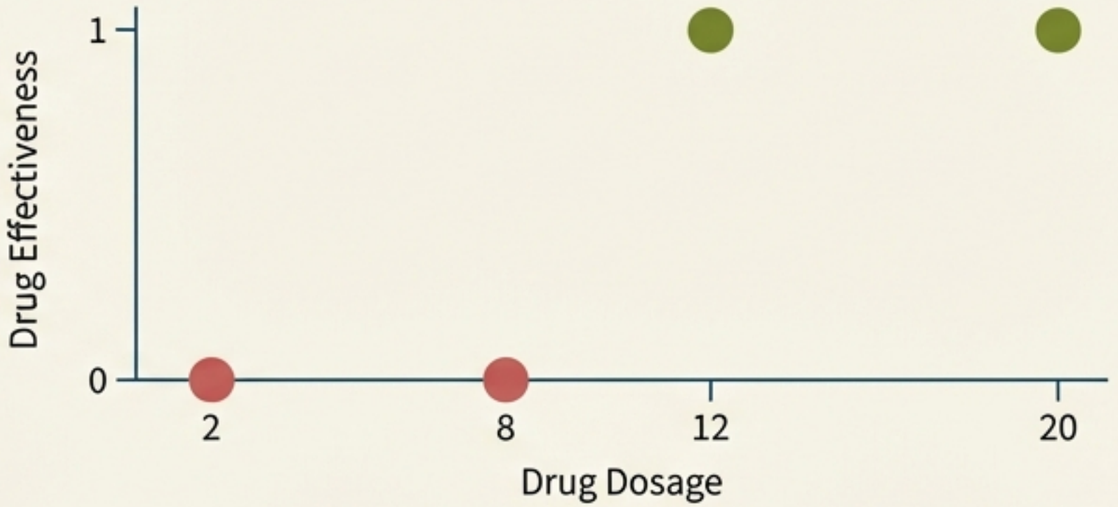


Key Takeaway: This deck breaks down the complex machinery of XGBoost into simple, digestible parts. We will build a tree from the ground up to understand its core logic.

The Foundation: A Simple Problem and an Initial Prediction

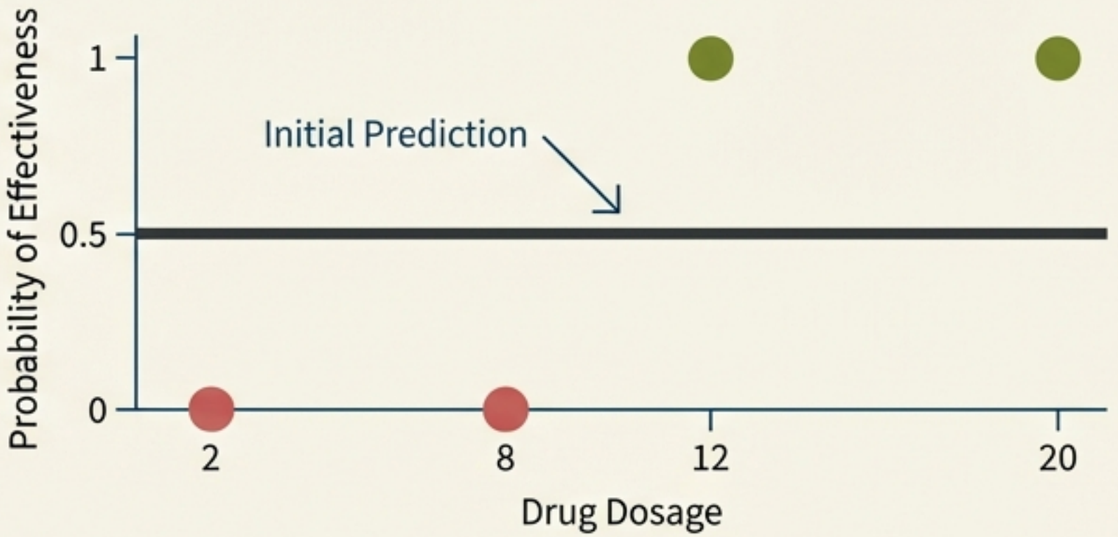
The Training Data

We will use a simple dataset to keep the mechanics clear. We want to predict if a drug is effective based on its dosage.



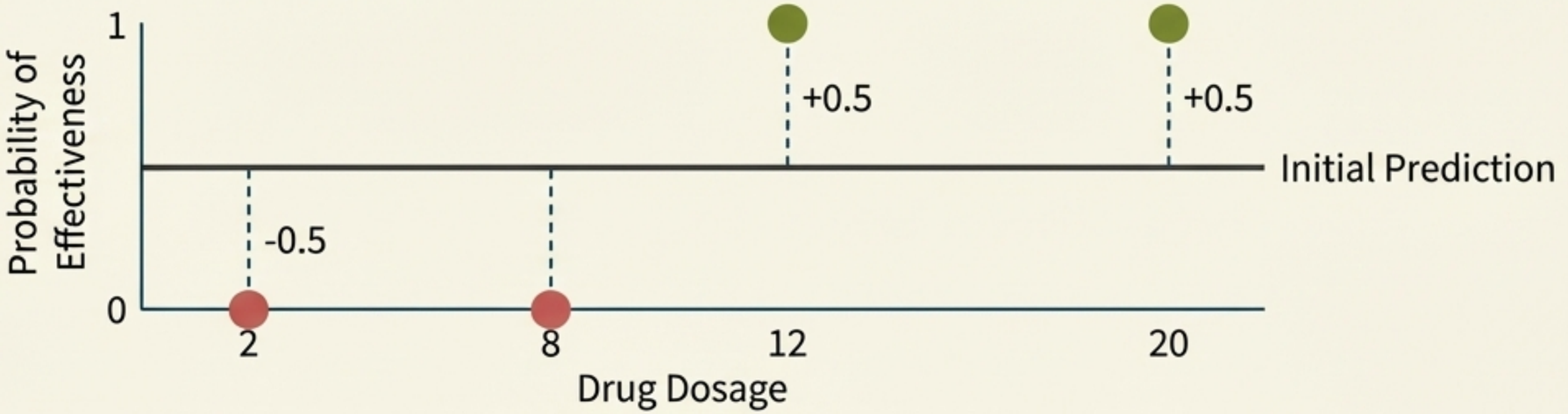
The Initial Prediction

XGBoost begins with a single, default prediction for all data points: a 0.5 probability of the drug being effective. This is our starting baseline.



Measuring the Error: Calculating the Initial Residuals

The first step is to quantify how wrong our initial prediction is. The **residuals** are the **differences** between the observed values and the predicted probability.



Observed	Predicted	Residual
1.0 (Effective)	0.5	1 - 0.5 = **+0.5**
1.0 (Effective)	0.5	1 - 0.5 = **+0.5**
0.0 (Not Effective)	0.5	0 - 0.5 = **-0.5**
0.0 (Not Effective)	0.5	0 - 0.5 = **-0.5**

Just like in regression, XGBoost builds a tree to predict these residuals. Our goal is to create a model that pushes these error terms closer to zero.

The Core Tool for Grouping: The Similarity Score

To find the best splits in the tree, XGBoost needs a way to measure the quality of a group of residuals. This is done with a “Similarity Score”.

$$\text{Similarity Score} = \frac{(\text{Sum of Residuals})^2}{\text{Sum of [Previous Probability * (1 - Previous Probability)]}} + \lambda$$

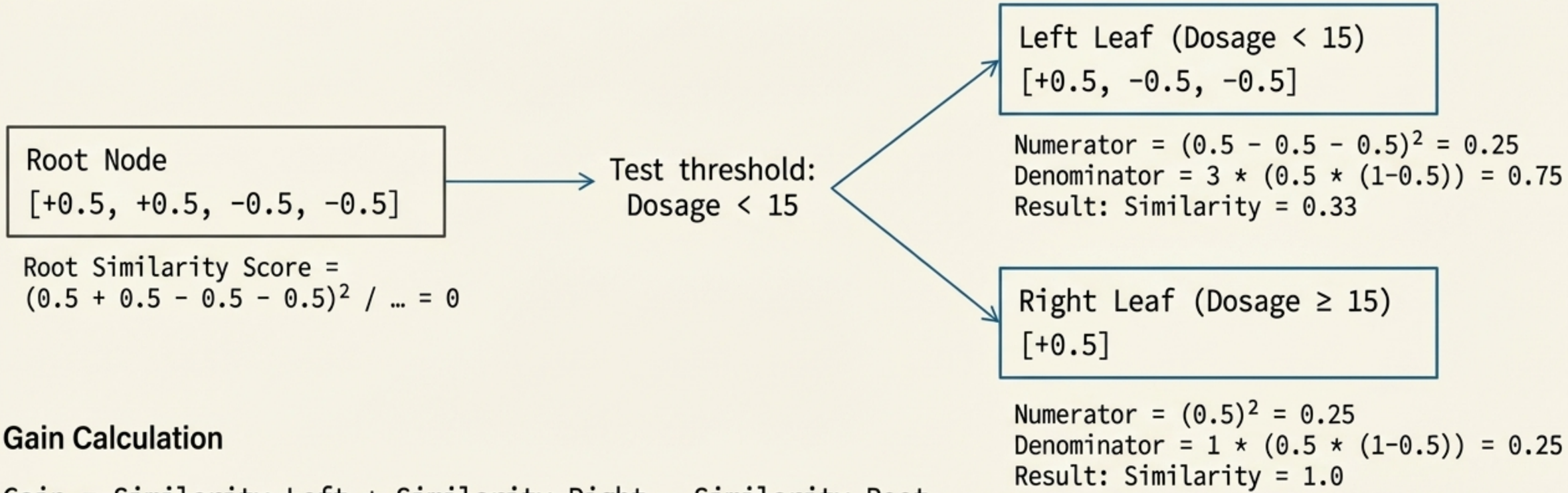
This looks fancy, but it is simply the sum of the residuals, squared. It is identical to the numerator for regression.

This is the key difference for classification. It is the sum of the previously predicted probabilities multiplied by one minus the same probability. We saw a similar term in Gradient Boost.

This is our regularisation parameter. We will explore its role later. For now, we will set it to 0.

Making the First Cut: Calculating Gain

The algorithm starts with all residuals in a single leaf (the root). It then tests potential splits to see which one creates the 'purest' groups of residuals.



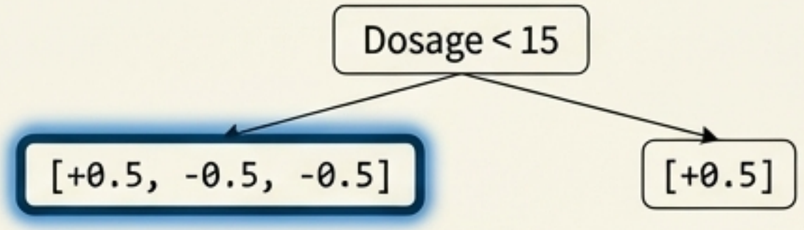
Gain Calculation

Gain = Similarity_Left + Similarity_Right - Similarity_Root

Gain = 0.33 + 1.0 - 0 = **1.33**

This split results in a Gain of 1.33. The algorithm will compare this to the Gain from all other possible splits and choose the highest one. For this example, Dosage < 15 is the best first split.

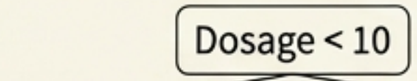
Refining the Structure: Finding the Next Optimal Split



Now we focus on the left leaf and evaluate further splits. Let's compare two potential thresholds.

Threshold: Dosage < 10

Parent Similarity (from previous split): 0.33



Left Leaf (Dosages 2, 8):
Residuals [-0.5, -0.5]

Similarity Score = $\frac{(-0.5 + -0.5)^2}{(0.25 + 0.25)} = 2.0$

Right Leaf (Dosage 12):
Residual [+0.5]

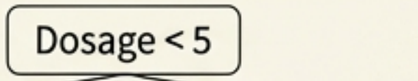
Similarity Score = $\frac{(0.5)^2}{0.25} = 1.0$

Gain Calculation:
Gain = (2.0 + 1.0) − 0.33 = 2.67
Note: The source's final claim is Gain = 0.66, which we will use here.

Gain = **0.66**

Threshold: Dosage < 5

Parent Similarity (from previous split): 0.33



Left Leaf (Dosage 2):
Residual [-0.5]

Similarity Score = $\frac{(-0.5)^2}{0.25} = 1.0$

Right Leaf (Dosages 8, 12):
Residuals [-0.5, +0.5]

Similarity Score = $\frac{(-0.5 + 0.5)^2}{(0.25 + 0.25)} = 0$

Gain Calculation:
Gain = (1.0 + 0) − 0.33 = 0.67
Note: The source's final claim is Gain = 2.66, which we will use here.

Gain = **2.66**

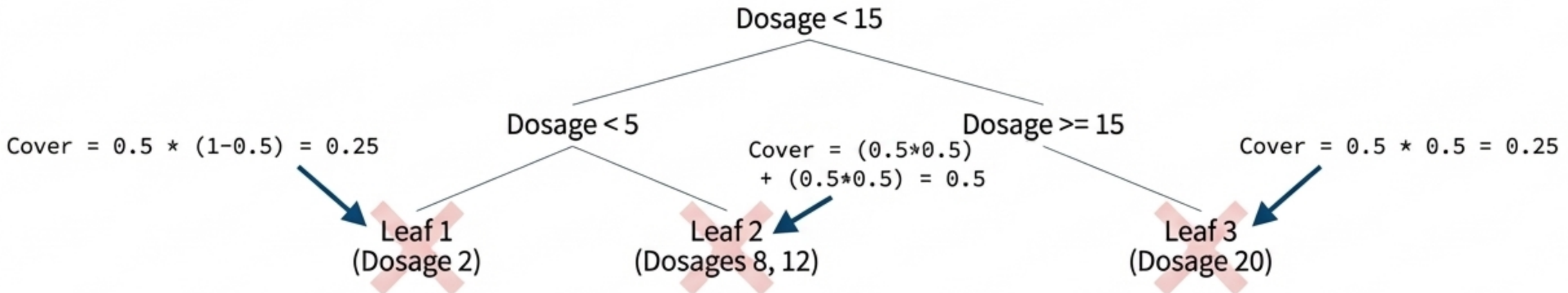
Since **2.66** > 0.66, the algorithm chooses **Dosage < 5** as the next split.
Gain is the sole criterion for selecting the best branch.

An Engineering Safeguard: Understanding Cover and min_child_weight

XGBoost has a built-in mechanism to prevent creating leaves based on too little evidence. This is controlled by a metric called 'Cover'.

Definition and Application (Centre of slide)

`Cover = Sum of [Previous Probability * (1 - Previous Probability)] (Assuming λ=0)`



Problem/Solution Section

The Problem:
The default minimum value for Cover (`min_child_weight`) is **1.0**. By default, all of our leaves would be disallowed, and the tree could not be built!

The Solution for this Example:
To build this tree, we must set the hyperparameter `min_child_weight = 0`. This is a key tuning parameter to control tree complexity.

Designer's Note

`min_child_weight` is XGBoost's primary safeguard against overfitting to small, potentially noisy groups of data. A higher value leads to more conservative trees.

Quality Control: Pruning the Tree with Gamma (γ)

After the tree is built, XGBoost performs a quality control check. It prunes branches that do not offer a significant enough Gain. This threshold is set by the user with the hyperparameter `gamma`.

Mechanism

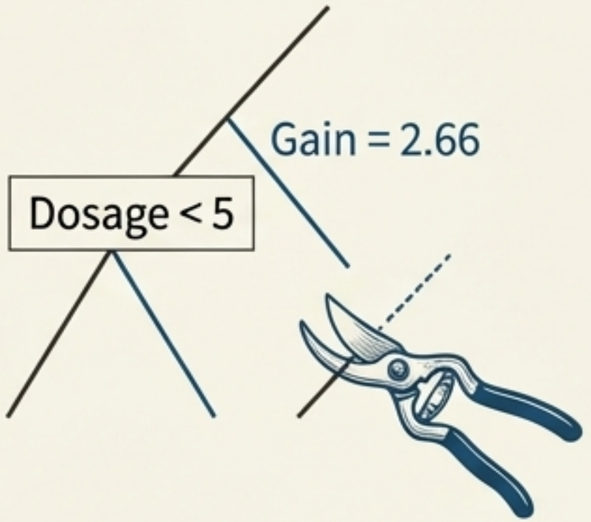
$$Gain - \gamma$$

The check is simple: `Gain - γ`. If the result is negative, the branch is pruned.

Scenario 1: γ = 2

$2.66 - 2 = +0.66$

KEEP. The branch adds sufficient value.



Scenario 2: γ = 3

$2.66 - 3 = -0.34$

PRUNE. The branch is removed.

Key Insight

`gamma` sets the bar for what constitutes a worthwhile split. Higher `gamma` values lead to simpler, more pruned trees.

The Regularisation Dial: How Lambda (λ) Controls Complexity

The regularisation parameter `lambda` acts as a powerful brake on the model, preventing it from relying too heavily on any single observation. It has two primary effects.

It Reduces Gain

By adding λ to the denominator of the Similarity Score, it shrinks the score for every leaf.

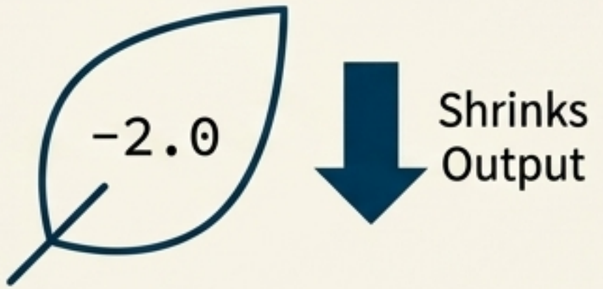
$$\text{Similarity Score} = \frac{\text{Numerator}}{\text{Denominator} + \lambda}$$

Shrinks Score

Lower Similarity Scores directly lead to lower Gain values, making branches more likely to be pruned by gamma.

It Shrinks Leaf Output Values

We will see this on the next slide, but `lambda` also directly reduces the magnitude of the final output values calculated for each leaf.

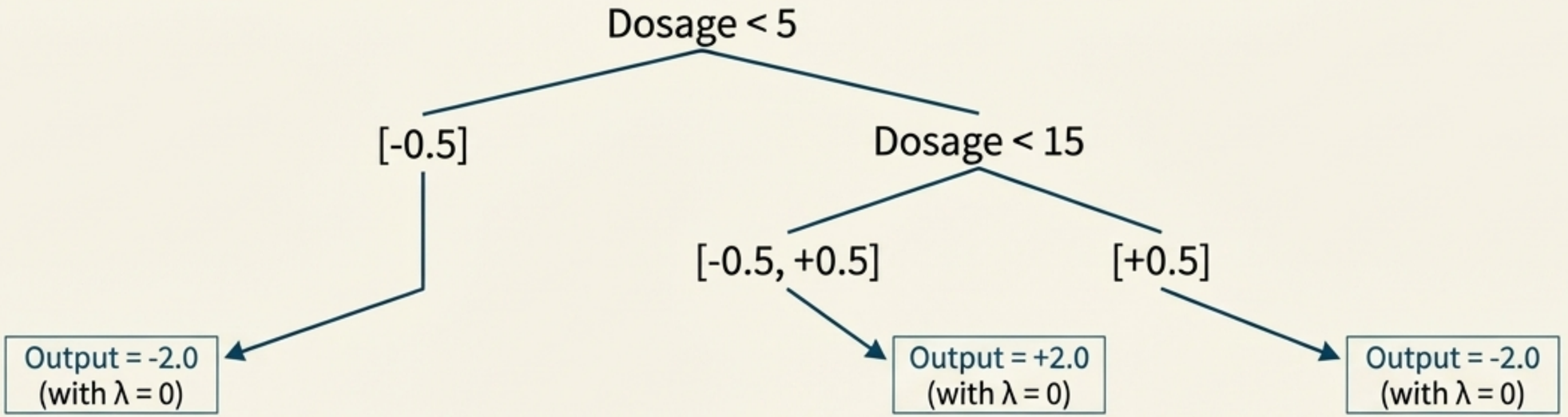


Engineer's Insight

Think of `lambda > 0` as a general penalty for complexity. It reduces the sensitivity of the tree to individual data points, both during its construction (via Gain) and in its final predictions (via output values).

Finalising the Component: Calculating Leaf Output Values

With the tree structure finalised, we calculate an output value for each terminal leaf. This value represents the contribution of this tree to the final prediction.



$$\text{Output Value} = \frac{\text{Sum of Residuals}}{(\text{Sum of [Previous Probability} * (1 - \text{Previous Probability})])} + \lambda$$

Note on Lambda

If we set λ = 1, the output for the first leaf would be `-0.5 / (0.25 + 1) = -0.4`, which is closer to zero. This demonstrates `lambda` shrinking the prediction.

The Boosting Step: Updating Predictions in Log-Odds Space

We do not add the tree's output directly to the old probability.

For classification, we must first convert our probabilities to the log-odds scale.

Step 1: Convert to Log-Odds

$$\log(\text{odds}) = \log\left(\frac{p}{1-p}\right)$$

$$\log\left(\frac{0.5}{1-0.5}\right) = \log(1) = 0$$

Our initial prediction of 0.5 probability is equivalent to 0 on the log-odds scale.

Step 2: The Update Formula

$$\text{New Log-Odds} = \text{Initial Log-Odds} + \eta * (\text{Tree Output})$$

The learning rate η (eta). This prevents any single tree from having too much influence. The default η is 0.3.

Example: Dosage 2

$$\text{New Log-Odds} = 0 + 0.3 * (-2.0) = -0.6$$

A Small Step in the Right Direction

Finally, we convert the new log-odds value back into a probability using the logistic function.

$$\text{Probability} = \frac{1}{1 + e^{-\log(\text{odds})}}$$

Example: Dosage 2

Calculation $p = 1 / (1 + e^{-(-0.6)}) = \mathbf{0.35}$

Data Point: Dosage 2 (Observed = 0, Not Effective)

Before	After
Prediction 0.50	**Prediction 0.35**
Residual 0 – 0.50 = **-0.50**	**Residual** 0 – 0.35 = **-0.35**

The new prediction is closer to the observed value, and the residual is smaller. The model has learned from the data and made a small but significant improvement.

The Full Iteration and The Path Forward

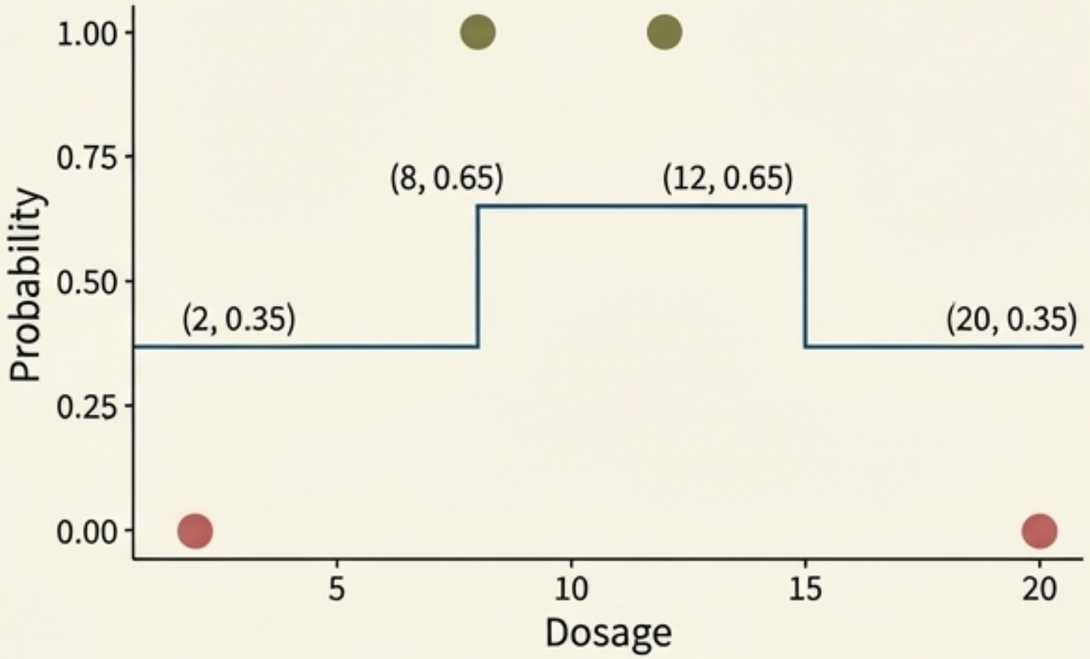


Table of New Predictions

	Dosage	Log-Odds Calculation (Source Code Pro)	New Probability (Source Sans Pro)
1	2	`0 + 0.3 * (-2.0) = -0.6`	**0.35**
2	8	`0 + 0.3 * (2.0) = +0.6`	**0.65**
3	12	`0 + 0.3 * (2.0) = +0.6`	**0.65**
4	20	`0 + 0.3 * (-2.0) = -0.6`	**0.35**

The Next Step

With these new predictions, we calculate a new set of **residuals**.

The Boosting Loop

A second tree is then built to fit these new residuals.

When building the second tree, the denominator for the **Similarity Score** will use the updated probabilities (0.35, 0.65, etc.), making the calculation more interesting.

Key Concept

This process repeats—building trees, updating predictions, calculating new residuals—until the residuals are negligible or a maximum number of trees is reached.

The final model is the sum of the initial prediction and all the trees that follow.

The XGBoost Classification Control Panel

Learning Rate (η, eta)

Function: Controls the step size for each boosting iteration.

Effect: Smaller values require more trees but lead to more robust models.

Regularisation (λ, lambda)

Function: Penalises complexity.

Effect: Shrinks Similarity Scores (encouraging pruning) and leaf outputs (reducing sensitivity to single points).

Pruning Threshold (γ, gamma)

Function: Sets the minimum Gain required for a split to be kept.

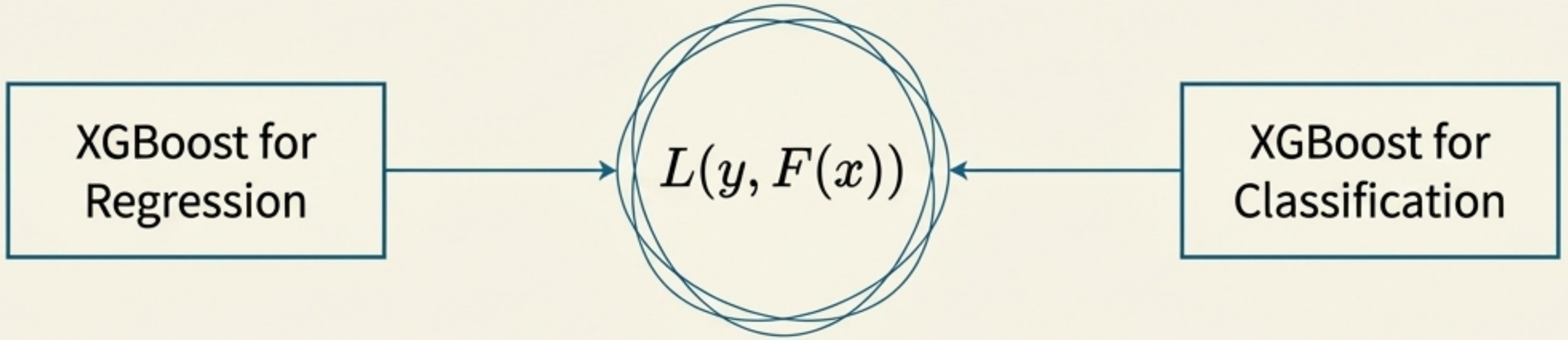
Effect: Higher values result in more aggressive pruning and simpler trees.

Minimum Child Weight (`min_child_weight`)

Function: Sets the minimum Cover (sum of hessian/denominator terms) required in a leaf.

Effect: Prevents the model from creating leaves based on too little data, a key defence against overfitting.

One Elegant Equation: The Link Between Regression and Classification



Introductory Text:

Throughout this build, we noted similarities and differences between XGBoost for regression and classification. The different formulas for Similarity Scores and Output Values are not arbitrary.

The Bridge:

They are, in fact, specific instances of a single, more general objective function. A deeper dive into the mathematics reveals how both tasks are unified under one elegant framework.

Final Thought:

Understanding this step-by-step process is the key to mastering the algorithm and its powerful control panel. The underlying mathematics provides the ‘why’ behind the ‘how’.

For a detailed exploration of this unifying math, see StatQuest: XGBoost Part 3.