XGBoost For Regression

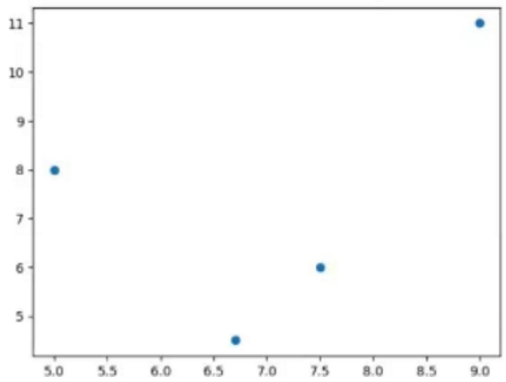
Problem Statement

Let's consider a simple regression problem. We have a student dataset with two columns:

- Input feature: CGPA (Cumulative Grade Point Average)
- Target output: Package (in LPA – Lakhs Per Annum)

We have data for four students, as shown below:

cgpa	package
6.7	4.5
9.0	11.0
7.5	6.0
5.0	8.0



We can visualize this data using a scatter plot:

- X-axis → CGPA
- Y-axis → Package

Our goal is to apply XGBoost Regression to this data. Once the model is trained, if a new student tells us their CGPA, we can predict their expected package.

This example will help us understand how XGBoost can be used to solve regression problems step-by-step.

Solution Overview

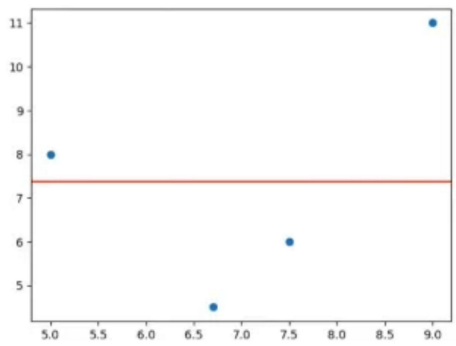
Let’s take a quick look at how XGBoost solves a regression problem. While XGBoost includes many optimizations for both performance and speed, at its core, it is still a form of Gradient Boosting. So, the way gradient boosting solves regression problems is essentially how XGBoost does it too — with a few key differences.

We need to predict the package using CGPA. In gradient boosting, we begin with a base estimator, which is usually the mean of the target variable.

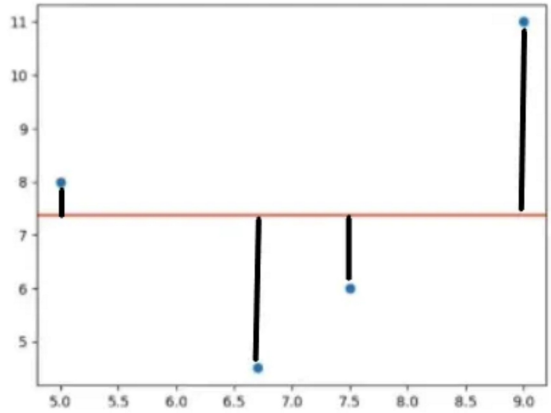
So, in Stage 1, if someone asks us the package of a new student, without even considering the CGPA, we would simply predict the mean of the packages of the four students.

Let’s say the mean is 7.375.

cgpa	package
6.7	4.5
9.0	11.0
7.5	6.0
5.0	8.0



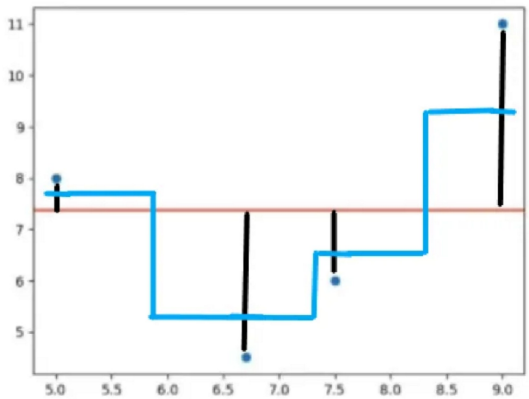
At this point, our prediction doesn't depend on the input feature (CGPA) at all — we just return the average package regardless of the CGPA.
This obviously causes errors, which are the differences between actual and predicted values. These are called residuals — or more specifically in this context, pseudo-residuals.



So the pseudo-residuals for our four students would be:

- 4.5 - 7.375 = -2.875
- 11 - 7.375 = 3.625
- 6 - 7.375 = -1.375
- 8 - 7.375 = 0.625

In Stage 2, we train a decision tree using the input feature (CGPA) as input and the pseudo-residuals as the target.
This tree tries to predict the error (pseudo-residuals) — i.e., we're trying to model what the first prediction missed.
This tree's output is added to the base model's predictions (scaled by a learning rate), creating a new, improved model.
This new combined model gives us a better prediction — represented by a new regression curve (e.g., the blue line).



Next, in Stage 3, we compute the new residuals (errors of the updated model), and again train another decision tree on these new residuals using the input feature.
This further improves the model.
We repeat this process, adding more trees — each one learning to correct the errors made by the model so far.
This is how Gradient Boosting solves regression problems — by adding weak learners (trees) stage by stage.
Now, XGBoost follows the same high-level idea:

- It starts with a base prediction (the mean).
- At each stage, it adds a new decision tree trained on the residuals.
- A learning rate is applied at each step.

The Key Difference in XGBoost

The overall process—starting with a base model and sequentially adding trees to correct errors—is similar in both Gradient Boosting and XGBoost. However, the major distinction lies in how the individual decision trees are constructed.
In a standard Gradient Boosting model, the regression trees are typically built to minimize a basic error metric, most commonly the Mean Squared Error (MSE). At each split, the tree selects the feature and threshold that most effectively reduces this error.
Note: Metrics like Gini impurity and entropy are used for classification, not for regression.
XGBoost, on the other hand, uses a more sophisticated approach. It constructs trees based on a custom objective function, which is a combination of:

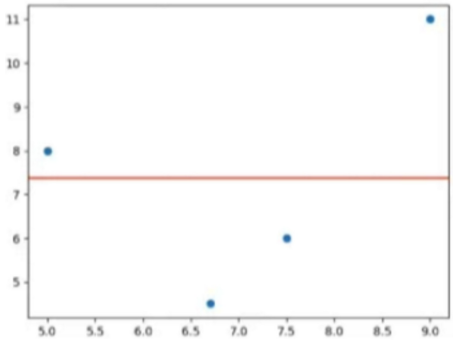
- A loss function (measuring prediction error), and
- A regularization term (penalizing model complexity, such as the number of leaves or leaf weights)

This regularization helps prevent overfitting, making the model more generalizable to unseen data.
At each split, XGBoost calculates the "gain" — a score that quantifies how much a split improves the model according to its regularized objective. The algorithm selects the split with the highest gain, resulting in a tree that balances accuracy and simplicity.

While both Gradient Boosting and XGBoost follow the same boosting architecture, the main advantage of XGBoost lies in its smarter, regularized, and mathematically optimized tree-building strategy. This is one of the key reasons behind XGBoost's superior performance and wide adoption in real-world machine learning tasks.

Step-by-Step Calculation — Stage 1

In Stage 1, we begin with a base estimator, which is the mean of the output variable (i.e., package values).



From our dataset:

Mean Package = $\frac{4.5+11+6+8}{4} = 7.375$

This base prediction (Model 1) outputs 7.3 for every input, regardless of CGPA. Now, we calculate the residuals (actual - predicted) from Model 1:

CGPA	Package	Model 1 Prediction	Residual 1
6.7	4.5	7.3	-2.8
9.0	11.0	7.3	3.7
7.5	6.0	7.3	-1.3
5.0	8.0	7.3	0.7

Stage 2 — First Tree (Before Splitting)

Now, in Stage 2, we build a decision tree using:

- Input feature → CGPA
- Target → Residual 1

Before splitting, we start with a single leaf node that contains all residuals: {−2.8, 3.7, −1.3, 0.7}

We calculate the similarity score for this node using the formula:

SS = $\frac{(\sum residuals)^2}{n+\lambda}$

Assuming λ = 0, we get :

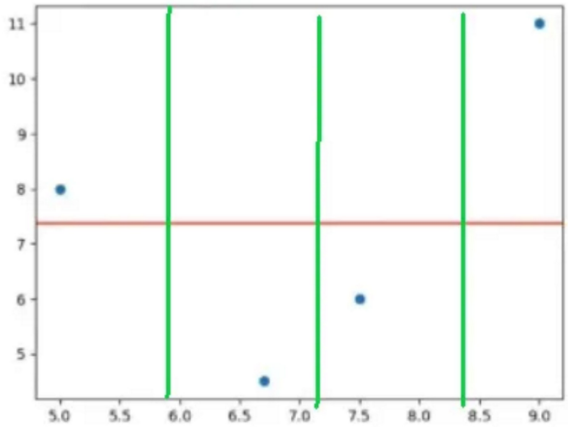
SS = $\frac{(-2.8+3.7-1.3+0.7)^2}{4+0} = \frac{(0.3)^2}{4} = 0.0225 \approx 0.02$

This score represents the "gain" if we keep all residuals in one leaf — this is our baseline.

Finding the Best Split

Now, we try to split the leaf node to see if we can improve the similarity score. We split based on CGPA values. First, we sort the CGPA column:
[5.0, 6.7, 7.5, 9.0]
We compute the midpoints between adjacent CGPA values as potential split thresholds:

- (5.0 + 6.7)/2 = 5.85
- (6.7 + 7.5)/2 = 7.1
- (7.5 + 9.0)/2 = 8.25



We will test these three thresholds (5.85, 7.1, 8.25) to split the dataset into two leaf nodes each time, and compute the total similarity score (left + right). The split with the highest gain (i.e., greatest increase in total similarity score) will be selected as the root node split of our first decision tree.
Here's the data we'll be splitting:

CGPA	Residual 1
6.7	-2.8
9.0	3.7
7.5	-1.3
5.0	0.7

Splitting Criteria Comparison

We'll now compare all three possible split points: CGPA < 5.85, CGPA < 7.1, and CGPA < 8.25 — to identify the best split for our first decision tree.

Splitting Criteria 1 → CGPA < 5.85

Left Node Residuals: 0.7

Right Node Residuals: -2.8, -1.3, 3.7

$$SS_L = \frac{(0.7)^2}{1+0} = 0.49$$

$$SS_R = \frac{(-2.8 - 1.3 + 3.7)^2}{3 + 0} = \frac{0.16}{3} \approx 0.05$$
$$Gain = SS_L + SS_R - SS_{Root} = 0.49 + 0.05 - 0.02 = 0.52$$

Splitting Criteria 2 → CGPA < 7.1

Left Node Residuals: 0.7, -2.8

Right Node Residuals: -1.3, 3.7

$$SS_L = \frac{(0.7 - 2.8)^2}{2} = \frac{4.41}{2} = 2.20$$
$$SS_R = \frac{(-1.3 + 3.7)^2}{2} = \frac{5.76}{2} = 2.88$$
$$Gain = SS_L + SS_R - SS_{Root} = 2.20 + 2.88 - 0.02 = 5.06$$

Splitting Criteria 3 → CGPA < 8.25

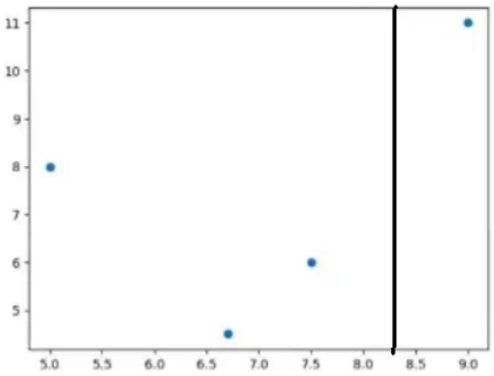
Left Node Residuals: 0.7, -2.8, -1.3

Right Node Residuals: 3.7

$$SS_L = \frac{(0.7 - 2.8 - 1.3)^2}{3} = \frac{11.56}{3} = 3.85$$
$$SS_R = \frac{(3.7)^2}{1} = 13.69$$
$$Gain = SS_L + SS_R - SS_{Root} = 3.85 + 13.69 - 0.02 = 17.52$$

Best Split

Among all three splits, the highest gain in similarity comes from Splitting Criteria 3 → CGPA < 8.25, with a gain of 17.52.
So, our first decision tree will split the data at CGPA = 8.25 — this becomes the root node of the tree.



Final Split: Depth = 2 Decision Tree

We now have two remaining splitting possibilities at CGPA < 5.85 and CGPA < 7.1 from the earlier best root split (CGPA < 8.25). Let's compare them by calculating their respective gains.

Splitting at CGPA < 5.85

Left node: [0.7]
Right node: [-2.8, -1.3]

$$SS_L = \frac{(0.7)^2}{1} = 0.49$$
$$SS_R = \frac{(-2.8 + -1.3)^2}{1} = \frac{(-4.1)^2}{2} = \frac{16.81}{2} = 8.40$$
$$Gain = SS_L + SS_R - SS_{Parent} = 0.49 + 8.40 - 3.85 = 5.04$$

Splitting at CGPA < 7.1

Left node: [0.7, -2.8]
Right node: [-1.3]

$$SS_L = \frac{(0.7 - 2.8)^2}{2} = \frac{(-2.1)^2}{2} = 2.20$$
$$SS_R = \frac{(-1.3)^2}{1} = 1.69$$
$$Gain = SS_L + SS_R - SS_{Parent} = 2.20 + 1.69 - 3.85 = 0.04$$

Best Split at Depth 2

Clearly, CGPA < 5.85 gives a much higher gain (5.04) compared to 0.04, so we choose it as the next best split.

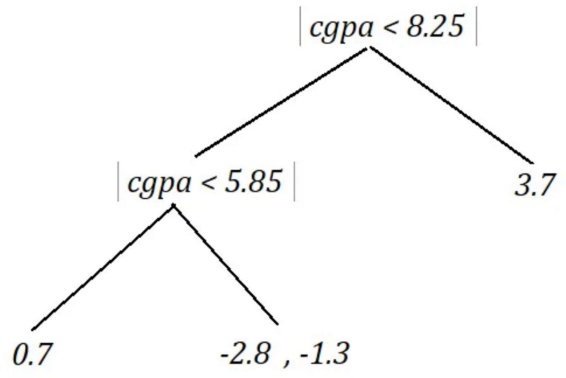
Stopping Criteria

We could go further and split again, but we'll stop here because:

- Our max depth is set to 2, for simplicity.
- XGBoost usually allows deeper trees (default max depth = 6), but with such a small dataset, further splits might lead to overfitting.

Final Decision Tree (Depth = 2)

Here is the resulting decision tree based on our calculated gains:



Leaf Node Output Value Calculation

In XGBoost, each leaf node of a decision tree outputs a value that contributes to the final prediction. This value is calculated using the following formula:

$$SS(Leaf\ Output) = \frac{\sum \text{Residuals}}{n + \lambda}$$

Where:

- ∑Residuals is the total of residuals within the leaf,
- n is the number of residuals (samples) in the leaf,
- λ is the regularization parameter.

For this example, the regularization parameter is assumed to be λ = 0 . The formula thus simplifies to:

$$SS = \frac{\sum \text{Residuals}}{n}$$

Example: Calculating Output Values for Three Leaf Nodes

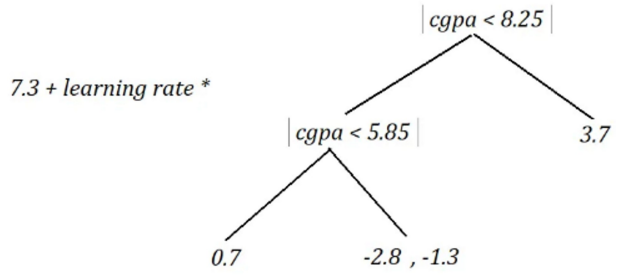
Given the residuals distributed across three leaves:

1. Leaf 1: Residuals = [0.7]
   $\frac{0.7}{1} = 0.7$
2. Leaf 2: Residuals = [-2.8, -1.3]
   $\frac{-2.8 + (-1.3)}{2} = -\frac{4.1}{2} = -2.05$
3. Leaf 3: Residuals = [3.7]
   $\frac{3.7}{1} = 3.7$

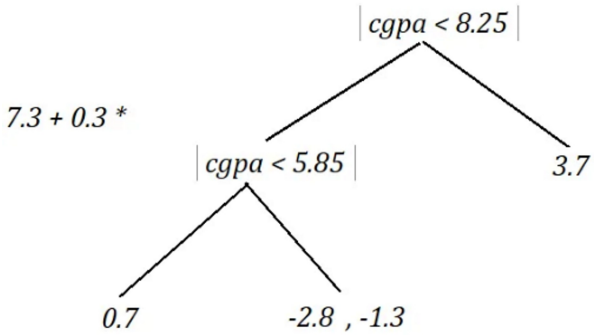
- Each leaf outputs the average residual, when λ=0 .
- These values are used to update the prediction at the corresponding nodes

during model training.

The combined model at Stage 2 is:



The default value of the learning rate is 0.3, so it becomes:



Now we will make predictions for each data point using the combined model from Stage 2 and compute the corresponding residuals:

CGPA	Actual Package (LPA)	Model 1 Prediction	Residual 1	Model 2 Prediction	Residual 2
6.7	4.5	7.3	-2.8	6.69	-2.19
9.0	11.0	7.3	3.7	8.41	2.59
7.5	6.0	7.3	-1.3	6.69	-0.69
5.0	8.0	7.3	0.7	7.51	0.49

If we compare the residuals side by side, we observe that the errors have been reduced in Stage 2. This indicates that the combined model at this stage performs better than the initial model.
Next, we proceed to Stage 3. In this stage, we train a new decision tree using the input feature (CGPA) and the residuals from Stage 2 as the target. The updated combined model will be:

Final Prediction = $\mu + \eta \cdot DT_1 + \eta \cdot DT_2 + \eta \cdot DT_3$

Where:

- μ is the mean of the target variable (initial prediction),
- η is the learning rate,
- DT_i represents the predictions from the decision tree at stage *i*.

The number of trees to be built is a hyperparameter that must be defined

beforehand. The process continues until either:

- A predefined number of trees is reached, or
- The residuals are sufficiently close to zero.

The overall objective of XGBoost regression is to minimize the residuals iteratively, which leads to a robust and high-performing model.

XGBoost builds a strong predictive model by combining multiple weak learners (decision trees) in a stage-wise manner. At each stage, it fits a new tree on the residuals of the previous model, gradually improving performance by minimizing the error. The final prediction is a weighted sum of all trees, scaled by the learning rate.

Key advantages of XGBoost:

- Handles bias and variance effectively.
- Supports regularization, preventing overfitting.
- Efficient and scalable for large datasets.
- Allows control over the number of trees, learning rate, and tree depth.

In our example, we demonstrated how each stage progressively reduced the residuals, showing how XGBoost refines predictions with each iteration. This makes it a powerful and popular algorithm for regression tasks in real-world applications.