

Gradient Boosting Machine Learning Algorithm

From Weak to Strong Learners

Sequential Decision Trees & Ensemble Magic

Topics Covered:

- Boosting Ensemble Technique
- Regression & Classification
- Sequential Tree Construction
- Learning Rate Optimization

December 31, 2025

Contents

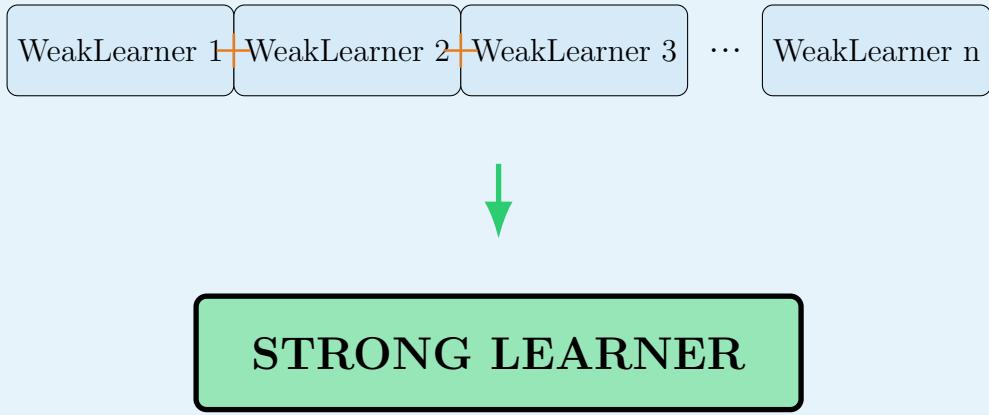
1	Introduction to Gradient Boosting	2
2	Understanding the Dataset	3
3	Step-by-Step Gradient Boosting Algorithm	4
4	Final Mathematical Formula	9
5	Decision Tree Construction Details	11
6	Complete Algorithm Workflow	12
7	Summary & Key Takeaways	13

Draft

1 Introduction to Gradient Boosting

What is Gradient Boosting?

Gradient Boosting is a powerful machine learning algorithm that belongs to the **ensemble learning** family. It creates a strong predictive model by combining multiple weak learners (typically decision trees) in a sequential manner.



Key Features

Versatile: Can solve both **Regression** and **Classification** problems

Sequential Learning: Trees are built one after another, each correcting errors of the previous

Boosting Technique: Combines weak learners to create a powerful strong learner

Error Minimization: Each tree focuses on reducing residual errors

Gradient Boosting vs AdaBoost

- **AdaBoost:** Creates **stumps** (decision trees with **only one split**)
- **Gradient Boosting:** Creates **full decision trees** (can grow to **complete depth**)

2 Understanding the Dataset

Regression Problem Dataset

Let's work with a practical example to understand Gradient Boosting from the ground up!

Record	Experience	Degree	Salary (K)
1	x_{11}	x_{12}	50
2	x_{21}	x_{22}	70
3	x_{31}	x_{32}	80
4	x_{41}	x_{42}	100

Independent Features
Experience & Degree

Predicts

Dependent Feature
Salary (Output)

*Since Salary is a continuous value,
this is a **Regression Problem***



3 Step-by-Step Gradient Boosting Algorithm

The Complete Process

Let's build our Gradient Boosting model step by step!

STEP 1: Create Base Model

The first step is to create a simple base model that is **unbiased** and provides a default value.

How? Compute the **average** of all output values (salaries)

Base Model Calculation

$$\begin{aligned}\text{Base Model Output} &= \frac{\sum \text{All Salaries}}{\text{Number of Records}} \\ &= \frac{50 + 70 + 80 + 100}{4} \\ &= \frac{300}{4} \\ &= \mathbf{75K}\end{aligned}$$



STEP 2: Compute Residuals (Errors)

Now we calculate the **difference** between actual values and predicted values from the base model.

Residual Formula

$$r_i = y_i - \hat{y}_i$$

r_i = Residual for record i

y_i = Actual (true) value

\hat{y}_i = Predicted value from base model

Residual Calculations:

Record	True Value (y)	Predicted (\hat{y})	Residual (r_1)
1	50K	75K	$50 - 75 = -25\text{K}$
2	70K	75K	$70 - 75 = -5\text{K}$
3	80K	75K	$80 - 75 = +5\text{K}$
4	100K	75K	$100 - 75 = +25\text{K}$



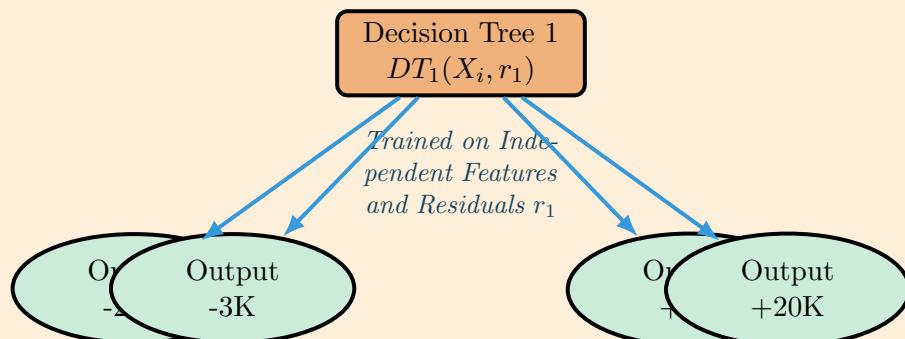
STEP 3: Build First Decision Tree

This is the **MOST IMPORTANT** step!

We construct a decision tree where:

- **Input Features:** X_i (Experience and Degree)
- **Output Feature:** r_1 (Residuals calculated in Step 2)

Tree builds using MSE, Variance Reduction, or Information Gain



After Training with the independent inputs and the residuals r_1 Let's say we got output r_2 . Here r_2 is new record. So after complete training The model will not train with 100 percent accuracy. So we assume these close values.

After Training, Decision Tree 1 produces outputs (r_2):

Record	Tree Output (r_2)
1	-23K
2	-3K
3	+3K
4	+20K

The Learning Rate Concept

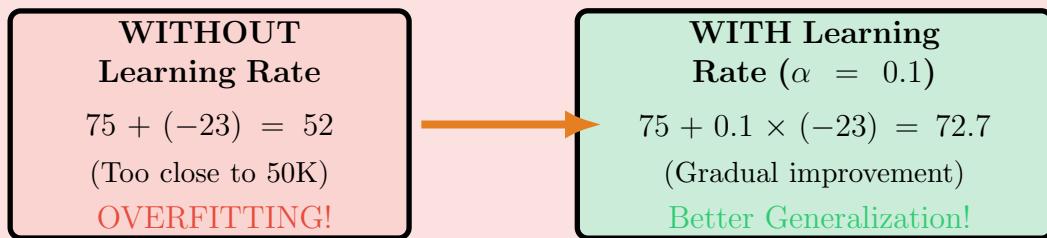
Problem: If we directly add the tree output to the base model, our model will **OVERFIT!**

Solution: Introduce a **Learning Rate (α)** to control the contribution of each tree.

Learning Rate Formula

$$\text{Updated Prediction} = H_0(x) + \alpha \cdot DT_1(x)$$

$H_0(x)$ = Base model output
 α = Learning rate (typically 0.1)
 $DT_1(x)$ = Decision Tree 1 output



Complete Prediction Calculation - Record 1

Given:

- Base Model Output: $H_0 = 75K$
- Decision Tree 1 Output: $DT_1 = -23K$
- Learning Rate: $\alpha = 0.1$

Step-by-step Calculation:

$$\begin{aligned}
 \hat{y}_1^{\text{updated}} &= H_0 + \alpha \cdot DT_1 \\
 &= 75 + (0.1 \times -23) \\
 &= 75 + (-2.3) \\
 &= \mathbf{72.7K}
 \end{aligned}$$

Error Analysis:

True Value: 50K — New Prediction: 72.7K — Error: 22.7K
 Still has error! Need more trees!

Updated Predictions After Tree 1

After applying the learning rate and getting outputs from Decision Tree 1:

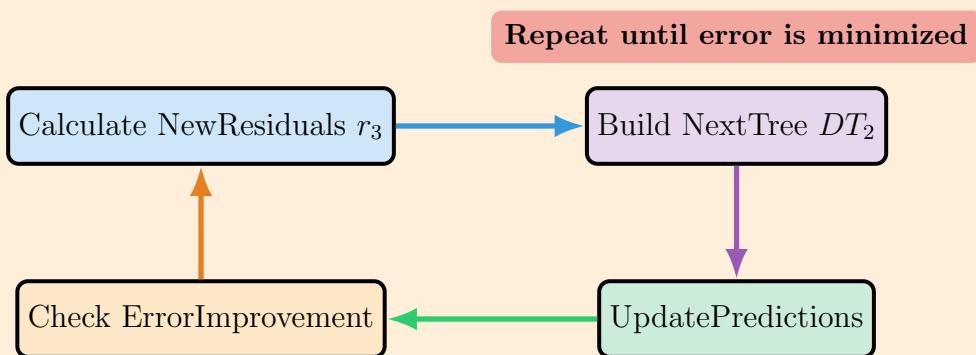
Record	True (y)	Base	Tree 1	Updated \hat{y}
1	50K	75	$0.1(-23) = -2.3$	$75 - 2.3 = \mathbf{72.7K}$
2	70K	75	$0.1(-3) = -0.3$	$75 - 0.3 = \mathbf{74.7K}$
3	80K	75	$0.1(3) = 0.3$	$75 + 0.3 = \mathbf{77.7K}$
4	100K	75	$0.1(20) = 2.0$	$75 + 2.0 = \mathbf{85K}$

Observations:

- Predictions are getting closer to true values
 - Errors still exist - we need **more trees!**
 - Each tree corrects the mistakes of the previous model

STEP 4: Repeat the Process (Iterations)

The Magic Loop:



For Decision Tree 2:

$$r_3 = y - \hat{y}_{\text{updated}}$$

For Record 1: $r_3 = 50 - 72.7 = -22.7K$

For Record 2: $r_3 = 70 - 74.7 = -4.7K$

Train DT_2 on (X_i, r_3) and repeat the process!

4 Final Mathematical Formula

The Complete Gradient Boosting Model

Final Prediction Function:

$$F(x) = H_0(x) + \alpha_1 H_1(x) + \alpha_2 H_2(x) + \alpha_3 H_3(x) + \dots + \alpha_n H_n(x)$$

Compact Notation:

$$F(x) = \sum_{i=0}^n \alpha_i H_i(x)$$

Where:

$F(x)$: Final prediction function

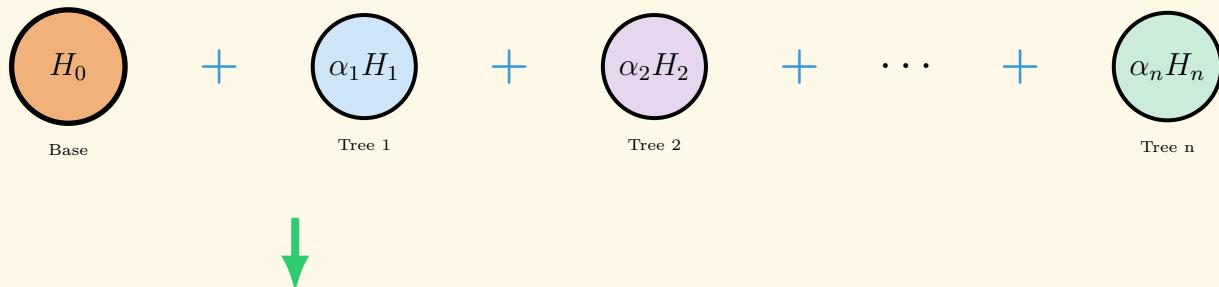
$H_0(x)$: Base model (average of all outputs)

$H_i(x)$: Decision tree i (where $i = 1, 2, 3, \dots, n$)

α_i : Learning rate for tree i

n : Total number of trees

Visual Representation of the Formula



Key Points About Learning Rates

1. **Range:** $\alpha \in (0, 1]$
2. **Common Value:** $\alpha = 0.1$ (as used in our example)
3. **Can Use:**
 - Same α for all trees: $\alpha_1 = \alpha_2 = \dots = \alpha_n = 0.1$
 - Different α for each tree (less common)
4. **Effect of Learning Rate:**

High α (e.g., 0.9)

Faster learning
Risk of overfitting
Fewer trees needed

Low α (e.g., 0.01)

Slower learning
Better generalization
More trees needed

Draft

5 Decision Tree Construction Details

How Are Trees Built in Gradient Boosting?

The decision trees in Gradient Boosting are **full regression trees**, not stumps!

Splitting Criteria:

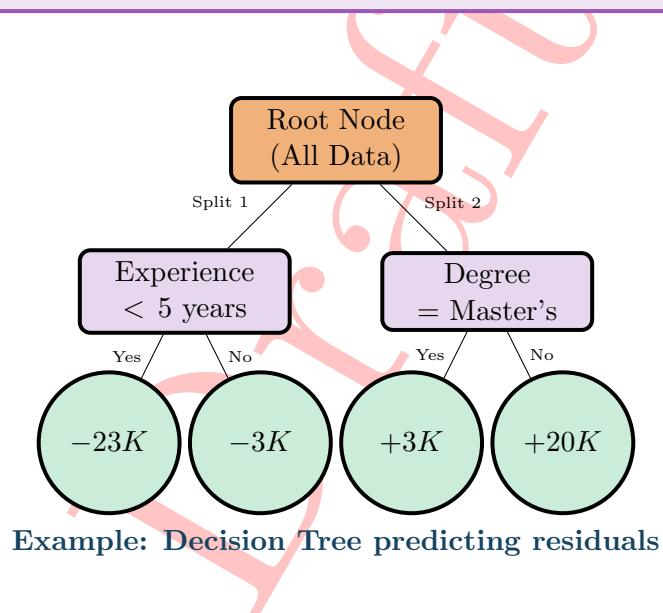
Mean Squared Error (MSE)

$$MSE = \frac{1}{n} \sum_{i=1}^n (y_i - \hat{y}_i)^2$$

Variance Reduction

$$\text{Variance} = \frac{1}{n} \sum_{i=1}^n (y_i - \bar{y})^2$$

Information Gain (for classification variant)



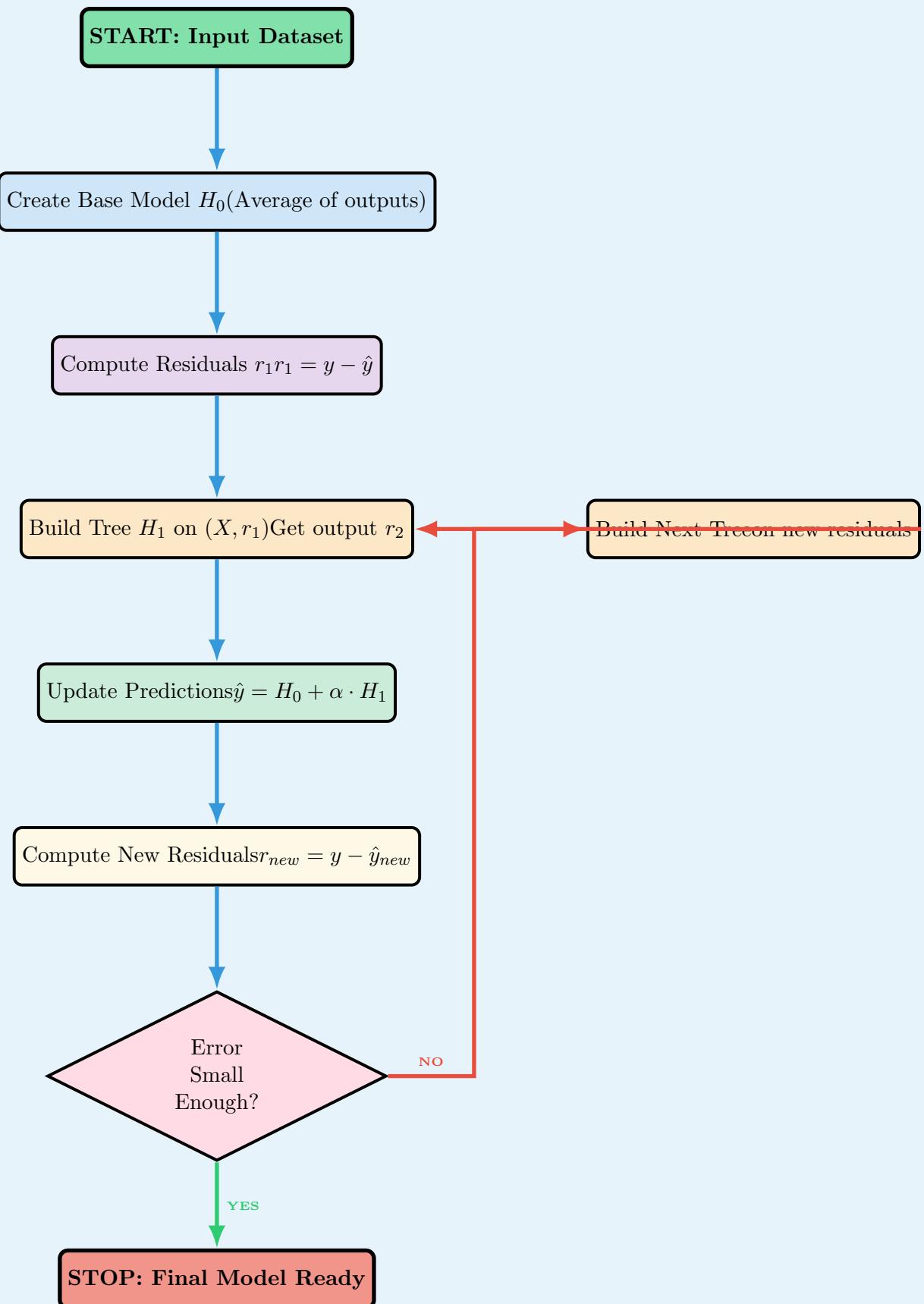
Pre-Pruning Parameters

To prevent overfitting, we can control tree growth:

- **max_depth:** Maximum depth of each tree
- **min_samples_split:** Minimum samples required to split a node
- **min_samples_leaf:** Minimum samples required at leaf node
- **max_features:** Maximum features to consider for splitting

6 Complete Algorithm Workflow

The Full Gradient Boosting Pipeline



7 Summary & Key Takeaways

Quick Summary

Gradient Boosting at a Glance:

What is it?

How does it work?

1. Create base model (average of outputs)
2. Calculate residuals (errors)
3. Build tree to predict residuals

Key Components:

- **Base Model:** $H_0(x)$ - Simple average
- **Residuals:** $r = y - \hat{y}$ - Errors to correct
- **Learning Rate:** $\alpha \in (0, 1]$ - Controls learning speed

Final Formula:

$$F(x) = H_0(x) + \alpha_1 H_1(x) + \alpha_2 H_2(x) + \alpha_3 H_3(x) + \dots + \alpha_n H_n(x)$$

Applications:

Regression: Predicting continuous values (salaries, prices, temperatures)

Classification: Predicting categories (spam detection, disease diagnosis)

Important Differences

Aspect	AdaBoost	Gradient Boosting
Tree Type	Stumps (1 split)	Full decision trees
Weighting	Sample weights	Learning rate
Training Focus	Misclassified samples	Residual errors
Flexibility	Less flexible	More flexible
Overfitting Risk	Lower	Higher (needs regularization)

Advantages

High Accuracy: Often wins Kaggle competitions

Handles Complex Relationships: Non-linear patterns

Feature Importance: Can identify important features

Versatile: Works for regression and classification

Robust: Handles missing values well

Disadvantages

Overfitting: Can overfit if not tuned properly

Computationally Expensive: Sequential nature = slow training

Sensitive to Outliers: Residuals can be affected

Requires Tuning: Many hyperparameters to tune

Not Parallelizable: Trees must be built sequentially

Quick Revision Checklist**Can you answer these?****1. What is the base model in Gradient Boosting?**

- Average of all output values

2. What are residuals?

- Difference between true values and predicted values: $r = y - \hat{y}$

3. Why do we use a learning rate?

- To prevent overfitting and control the contribution of each tree

4. What is the typical range of learning rate?

- Between 0 and 1, commonly 0.1

5. What is the final prediction formula?

- $F(x) = \sum_{i=0}^n \alpha_i H_i(x)$

6. What kind of trees does Gradient Boosting use?

- Full decision trees (not stumps like AdaBoost)

7. How are trees built sequentially?

- Each new tree is trained on the residuals of the previous model

8. Can Gradient Boosting solve both regression and classification?

- Yes! It's versatile for both types of problems

Final Thoughts

Congratulations!

You've just learned one of the most powerful machine learning algorithms used in industry today!

What's Next?

Practice implementing in Python/Scikit-learn

Apply to real datasets (Kaggle competitions!)

Experiment with hyperparameters (learning rate, max depth, etc.)

Learn about XGBoost, LightGBM (advanced variants)

Compare with Random Forest and other algorithms

Keep Learning! Keep Growing!

Practice Makes Perfect

Happy Learning!

Review these notes regularly for best retention

Star performers understand the math AND the intuition