Gradient Boosting Tree

Luu Minh Sao Khue

December 7, 2025

1. Introduction

Gradient Boosting Trees (GBT) are a powerful ensemble learning method that builds a strong predictive model by combining many weak decision trees. The main ideas are:

- Instead of training all trees independently (as in Random Forest), GBT builds trees *sequentially*.
- Each new tree tries to correct the errors made by the previous trees.
- The corrections are not done randomly: they follow the *gradient* of a chosen loss function.
- By repeatedly adding these small improvements, the final model becomes highly accurate and flexible.

Gradient boosting is general: it works for regression, binary classification, and multi-class classification simply by choosing an appropriate loss function.

2. Intuition

Suppose we want a model $F(x)$ that minimizes some loss function $L(y, F(x))$. In standard boosting (e.g., AdaBoost), each new tree focuses on the "hard" samples. In contrast, *gradient boosting* uses a more principled view:

> *Each new tree is trained to predict the* ***negative gradient*** *of the loss function with respect to the current predictions.*

Why the gradient? Because the gradient tells us the direction in which the loss decreases the fastest. If our model $F_{m-1}(x)$ is not perfect, then the gradient of the loss at each training sample tells us how we should adjust the prediction at that point. A decision tree is then trained to approximate these gradient values.

Thus, the gradient appears as:

$$g_{im} = - \left. \frac{\partial L(y_i, F(x_i))}{\partial F(x_i)} \right|_{F(x)=F_{m-1}(x)} .$$

This g_{im} becomes the pseudo-target that the m-th decision tree tries to fit.

3. Loss Functions and Derivative Derivations

In gradient boosting, each new tree is trained to approximate the *negative gradient* of the loss with respect to the current model prediction $F(x)$. Therefore, we must be able to **compute the derivative of the loss with respect to** $F(x)$. This derivative tells us:

- how much the loss would change if we slightly changed the prediction $F(x)$,
- in which direction we should move $F(x)$ to reduce the loss.

The negative gradient at each training point becomes the *pseudo-target* that the next tree tries to fit.

Below we derive these gradients step by step.

3.1 Regression Loss and Its Derivative

For standard regression, we often use the squared error loss:

$$L(y, F(x)) = \frac{1}{2}\big(y - F(x)\big)^2.$$

Let us denote $F(x)$ by F for simplicity. Then

$$L(y, F) = \frac{1}{2}\big(y - F\big)^2.$$

We want $\dfrac{\partial L}{\partial F}$.

Step 1: Expand the expression conceptually.

$$L = \frac{1}{2}(y - F)^2.$$

Step 2: Apply the chain rule. Treat $(y - F)$ as an inner function:

$$u(F) = y - F, \qquad L = \frac{1}{2}u^2.$$

Then:

$$\frac{dL}{dF} = \frac{dL}{du} \cdot \frac{du}{dF}.$$

Step 3: Differentiate each part.

$$\frac{dL}{du} = \frac{1}{2} \cdot 2u = u = y - F,$$

$$\frac{du}{dF} = \frac{d}{dF}(y - F) = -1.$$

Step 4: Multiply them.

$$\frac{\partial L}{\partial F} = (y - F) \cdot (-1) = F - y.$$

So the gradient with respect to the prediction is

$$\frac{\partial L}{\partial F(x)} = F(x) - y.$$

The **negative gradient** (the value we actually fit) is

$$-\frac{\partial L}{\partial F(x)} = y - F(x),$$

which is exactly the usual regression residual.

3.2 Binary Classification (Logistic Loss) and Its Derivative

For binary classification, we often use labels $y \in \{-1, +1\}$ and the logistic loss

$$L(y, F(x)) = \log\bigl(1 + e^{-yF(x)}\bigr).$$

Again, write $F(x)$ as F to simplify notation:

$$L(y, F) = \log\bigl(1 + e^{-yF}\bigr).$$

Step 1: Apply the chain rule. Let

$$u(F) = 1 + e^{-yF}, \qquad L = \log u.$$

Then

$$\frac{dL}{dF} = \frac{dL}{du} \cdot \frac{du}{dF}.$$

Step 2: Differentiate each part. First:

$$\frac{dL}{du} = \frac{1}{u} = \frac{1}{1 + e^{-yF}}.$$

Next, for $u(F) = 1 + e^{-yF}$:

$$\frac{du}{dF} = \frac{d}{dF}\bigl(1 + e^{-yF}\bigr) = 0 + e^{-yF} \cdot \frac{d(-yF)}{dF} = e^{-yF} \cdot (-y) = -ye^{-yF}.$$

Step 3: Multiply them.

$$\frac{\partial L}{\partial F} = \frac{1}{1 + e^{-yF}} \cdot \bigl(-ye^{-yF}\bigr) = -y\frac{e^{-yF}}{1 + e^{-yF}}.$$

Notice that

$$\frac{e^{-yF}}{1 + e^{-yF}} = \frac{1}{e^{yF} + 1} = \frac{1}{1 + e^{yF}},$$

so we can rewrite:

$$\frac{\partial L}{\partial F} = -\frac{y}{1 + e^{yF}}.$$

The **negative gradient** used as the pseudo-target is then

$$-\frac{\partial L}{\partial F(x)} = \frac{y}{1 + e^{yF(x)}}.$$

Logit and Probability: Why This Form Appears In gradient boosting for classification, the model output $F(x)$ is often interpreted as a *logit*, i.e. a log-odds:

$$F(x) = \log \frac{p(x)}{1-p(x)},$$

where $p(x) = \mathbb{P}(y = +1 \mid x)$.

We can convert from logit F to probability p as follows:

$$F = \log \frac{p}{1-p} \quad \Longrightarrow \quad e^F = \frac{p}{1-p}.$$

Solve for p:

$$\begin{aligned} e^F(1-p) = p \quad &\Longrightarrow \quad e^F - e^F p = p \\ e^F &= p + e^F p = p(1+e^F) \\ p &= \frac{e^F}{1+e^F} = \frac{1}{1+e^{-F}}. \end{aligned}$$

Thus the probability $\mathbb{P}(y = +1 \mid x)$ is

$$p(x) = \sigma\big(F(x)\big) = \frac{1}{1+e^{-F(x)}}.$$

Similarly,

$$\mathbb{P}(y = -1 \mid x) = 1 - p(x) = \frac{1}{1+e^{F(x)}}.$$

This is exactly the term that appears in the gradient:

$$\frac{\partial L}{\partial F} = -\frac{y}{1+e^{yF}} = \begin{cases} -\dfrac{1}{1+e^F} & \text{if } y = +1, \\ +\dfrac{1}{1+e^{-F}} & \text{if } y = -1, \end{cases}$$

which can be expressed in terms of probabilities of the classes.

4. Binary Classification (Logistic Loss): Logit-Space and Probability-Space Gradients

For binary classification with the logistic loss, there are **two equivalent ways** to compute the gradient. They differ only in the representation of the labels and the model output, but both lead the boosting algorithm to update predictions in the same direction.

Method 1: Logit-Space Formulation (Original Boosting View). We use labels $y \in \{-1, +1\}$ and the model output $F(x)$ is a *logit* (log-odds). The logistic loss is:

$$L(y, F) = \log\big(1 + e^{-yF}\big).$$

As derived earlier, the gradient is:

$$\frac{\partial L}{\partial F} = -\frac{y}{1+e^{yF}},$$

and the **negative gradient** (the residual used as a pseudo-target) is:

$$r_i^{(\text{logit})} = \frac{y_i}{1 + e^{y_i F(x_i)}}.$$

This is the formulation used in the original gradient boosting framework.

Method 2: Probability-Space Formulation. The model can be rewritten in terms of a **probability**:

$$p_i = \sigma(F(x_i)) = \frac{1}{1 + e^{-F(x_i)}},$$

and convert labels to $y_i' \in \{0, 1\}$.

In probability space, the gradient of the logistic (cross-entropy) loss w.r.t. the probability is:

$$r_i^{(\text{prob})} = y_i' - p_i.$$

This looks intuitive: *residual = actual class − predicted probability.* However, boosting still needs to update the *logit* $F(x)$, not the probability. Therefore, using this method must convert:

$$F = \log \frac{p}{1-p}, \qquad p = \sigma(F).$$

Thus, after fitting a tree to $y' - p$, the update is applied to the logit $F(x)$, and then optionally converted back to a probability.

Why the Two Methods Are Equivalent. The key fact is that both gradients point in the *same improvement direction.* If $y' = 1$ but p is too small, both gradients tell the model to increase $F(x)$; if $y' = 0$ but p is too large, both gradients tell the model to decrease $F(x)$.

Mathematically, the two expressions are linked by the identity:

$$\frac{y}{1 + e^{yF}} = \begin{cases} 1 - p & \text{if } y' = 1, \\ -p & \text{if } y' = 0, \end{cases}$$

where $y' = (y + 1)/2$ converts $\{-1, +1\}$ labels to $\{0, 1\}$.

Thus:

$$r_i^{(\text{logit})} \propto y_i' - p_i = r_i^{(\text{prob})}.$$

The two forms differ by a positive scaling factor, which does not affect the direction of the fitted tree. Therefore, both methods are theoretically consistent and lead to the same gradient-boosting updates.

5. Mathematical Algorithm

Let (x_i, y_i), $i = 1, \ldots, N$, be the training data, and let M be the total number of trees. Let ν be the learning rate.

Algorithm 1 Gradient Boosting Tree (General Form)

```
1: Initialize the model with a constant
       $$F_0(x) = \arg\min_c \sum_{i=1}^{N} L\big(y_i, c\big).$$
2: for $m = 1$ to $M$ do
3:     Compute negative gradients (pseudo-residuals):
           $$g_{im} = -\left.\frac{\partial L\big(y_i, F(x_i)\big)}{\partial F(x_i)}\right|_{F(x)=F_{m-1}(x)}, \quad i = 1, \dots, N.$$
4:     Fit a regression tree $h_m(x)$ to the targets $\{g_{im}\}_{i=1}^{N}$.
5:     For each leaf $j$ of the tree, compute the optimal leaf weight
           $$\gamma_{jm} = \arg\min_\gamma \sum_{x_i \in R_{jm}} L\big(y_i, F_{m-1}(x_i) + \gamma\big),$$
       where $R_{jm}$ is the set of samples in leaf $j$ of tree $h_m$.
6:     Update the model:
           $$F_m(x) = F_{m-1}(x) + \nu \sum_j \gamma_{jm} \, \mathbb{I}\big(x \in R_{jm}\big).$$
7: end for
8: Return the final model $F_M(x)$.
```

6. Summary

Gradient boosting builds a strong model by:

- Starting from a simple constant prediction.
- Repeatedly fitting decision trees to the negative gradient of the loss.
- Updating the model slowly using a learning rate.

This framework unifies regression, classification, and many other tasks, and forms the foundation for modern algorithms such as XGBoost, LightGBM, and CatBoost.