Handout: A Quick Introduction to Gradient Boosting

Machine Learning Class Notes

November 8, 2025

Abstract

This document outlines the core concepts of Gradient Boosting. We start with the simple case of fitting residuals in regression, show how this is a special case of a more general "gradient" boosting algorithm, and then extend the framework to other loss functions (like L1) and to binary classification.

1 The Boosting Ensemble

The core idea of boosting is to build a strong predictive model by sequentially adding weak learners (underfitted models with high bias and low variance). Each new weak learner (typically a small decision tree) is trained to correct the errors made by all the previous learners.

A boosted model $F_M(x)$ is an **additive model** of the form:

$$F_M(x) = F_0(x) + \sum_{m=1}^{M} h_m(x)$$

Where:

- $F_0(x)$ is an initial "base" prediction (e.g., the mean of all y values).
- $h_m(x)$ is the m-th weak learner (a regression tree).

2 Simple Case: Boosting for Regression with L2 Loss

Let's start with the most intuitive case: regression with a **Squared Error (L2)** loss function. Our goal is to minimize the total loss:

$$L_{Total} = \sum_{i=1}^{N} L(y_i, F(x_i)) = \sum_{i=1}^{N} \frac{1}{2}(y_i - F(x_i))^2$$

The simplest version of boosting for this problem is often called "Residual Fitting":

1. **Initialize:** Start with an initial prediction. A good one is the mean of the target values.

$$F_0(x) = \frac{1}{N} \sum_{i=1}^{N} y_i$$

2. **Iterate for $m = 1$ to M trees:**

   (a) **Compute Residuals:** Find the error for each data point based on the *current* model $F_{m-1}(x)$.

$$r_{im} = y_i - F_{m-1}(x_i)$$

(b) **Fit a Weak Learner:** Train a new regression tree $h_m(x)$ to predict these residuals r_{im}. The tree is learning to predict the *error* of the current model.

(c) **Update the Model:** Add the new tree's prediction to the overall model.

$$F_m(x) = F_{m-1}(x) + h_m(x)$$

3. **Final Model:** The final model is $F_M(x)$.

3 The Gradient Connection

It turns out that residual fitting is a special case of a more powerful idea: **gradient descent**.

Let's look at the loss for a single data point: $L(y_i, F(x_i)) = \frac{1}{2}(y_i - F(x_i))^2$.

What if we took the **negative gradient** of this loss function with respect to the model's output $F(x_i)$?

$$\begin{aligned} -\frac{\partial L}{\partial F(x_i)} &= -\frac{\partial}{\partial F(x_i)}\left[\frac{1}{2}(y_i - F(x_i))^2\right] \\ &= -\left[(y_i - F(x_i)) \cdot (-1)\right] = y_i - F(x_i) \end{aligned}$$

This means: For the Squared Error (L2) loss, the **residual is exactly the negative gradient** of the loss function.

$$r_{im} = y_i - F_{m-1}(x_i) = -\left[\frac{\partial L(y_i, F(x_i))}{\partial F(x_i)}\right]_{F(x)=F_{m-1}(x)}$$

This means our "Residual Fitting" algorithm was, secretly, a form of gradient descent. It was fitting a new tree to the negative gradient of the loss, trying to "step" the function $F(x)$ in the direction that most rapidly decreases the loss. Note that in Gradient Boosting, we calculate the gradient of the output for each training sample (as opposed to the gradient of the weights in other algorithms).

4 The General Gradient Boosting Algorithm

This insight allows us to generalize boosting to *any* differentiable loss function. We just replace the "residual" with the "negative gradient." We simply use this negative gradient as the new target for our weak learners.

1. **Initialize:** Start with an initial prediction $F_0(x)$ that minimizes the total loss.

$$F_0(x) = \arg\min_{\gamma} \sum_{i=1}^{N} L(y_i, \gamma)$$

(For L2 loss, this is the mean. For L1 loss, this is the median.)

2. **Iterate for $m = 1$ to M trees:**

(a) **Compute Negative Gradients:** For each data point, compute the negative gradient of the loss function, evaluated at the current model's prediction.

$$r_{im} = -\left[\frac{\partial L(y_i, F(x_i))}{\partial F(x_i)}\right]_{F(x)=F_{m-1}(x)}$$

(b) **Fit a Weak Learner:** Train a new *regression tree* $h_m(x)$ to predict these negative gradients r_{im}.

(c) **Find Optimal Step Length:** We need to find the best step length γ_m for this new tree. Note that for L2 regression, the gradient coincided with the residual, and thus the optimal step length was one. But for other functions, this does not have to be the case. The Step length is chosen to minimize the total loss. This is a 1D optimization problem:

$$\gamma_m = \arg\min_{\gamma} \sum_{i=1}^{N} L(y_i, F_{m-1}(x_i) + \gamma \cdot h_m(x_i))$$

This step is typically solved using a line search.

(d) **Update the Model:** Add the new tree's contribution, scaled by the step length.

$$F_m(x) = F_{m-1}(x) + \gamma_m \cdot h_m(x)$$

3. **Final Model:** The final model is $F_M(x)$.

5 Example: Gradient Boosting with L1 (Absolute) Loss

What if we want a model that is more robust to outliers? We can replace the L2 loss with the **Absolute Error (L1)** loss.

1. **Loss Function:** $L(y_i, F(x_i)) = |y_i - F(x_i)|$
2. **Gradient (Sub-gradient):**

$$\frac{\partial L}{\partial F(x_i)} = \frac{\partial}{\partial F(x_i)} |y_i - F(x_i)| = \text{sign}(F(x_i) - y_i)$$

Where sign(z) is -1 if $z < 0$ and +1 if $z > 0$.

3. **The Negative Gradient (New Target):**

$$r_{im} = -\left[\frac{\partial L}{\partial F(x_i)}\right]_{F=F_{m-1}} = -\text{sign}(F_{m-1}(x_i) - y_i) = \text{sign}(y_i - F_{m-1}(x_i))$$

Insight: When using L1 loss, the negative gradient is just the *sign* of the error (i.e., +1 or -1) and does not care about error magnitude. Here, it is quite clear why we need a step length.

6 Example: Gradient Boosting for Binary Classification

How do we adapt this for a binary $y \in \{0, 1\}$ classification problem?

6.1 The Output: Log-Odds

First, we can't have our trees output a probability $p \in [0, 1]$ directly. The sum of trees can output any real number from $(-\infty, \infty)$. Instead, we define the model $F(x)$ to be the **log-odds (or logit)** of the probability:

$$F(x) = \log\left(\frac{p(x)}{1 - p(x)}\right) \quad \text{where } p(x) = P(y = 1|x)$$

We can always recover the probability $p(x)$ from the model's raw output $F(x)$ by using the **sigmoid (logistic) function**:

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

6.2 The Loss Function: Log Loss

The standard loss function for binary classification is the **Binary Cross Entropy**. We write it in terms of the raw model output $F(x_i)$ and the sigmoid function $\sigma(\cdot)$:

$$L(y_i, F(x_i)) = -\left[y_i \log(\sigma(F(x_i))) + (1-y_i)\log(1-\sigma(F(x_i)))\right]$$

Where $\sigma(F(x_i))$ is the sigmoid function defined in the previous section.

6.3 The Gradient (New Target)

We need to find the gradient of this loss with respect to our model's output, $F(x_i)$.

It can be shown (recall the gradient of logistic regression from the past lecture) that:

$$\frac{\partial L}{\partial F(x_i)} = p(x_i) - y_i$$

Therefore, the new target (the negative gradient) for binary classification is:

$$r_{im} = -\left[\frac{\partial L}{\partial F(x_i)}\right]_{F=F_{m-1}} = -(p_{m-1}(x_i) - y_i) = \mathbf{y_i - p_{m-1}(x_i)}$$

Where $p_{m-1}(x_i) = \sigma(F_{m-1}(x_i))$.

6.4 The Algorithm and Interpreting the Output

The algorithm is the same as the general one, but with this specific gradient target.

1. **Initialize:** $F_0(x) = \log\left(\frac{\bar{y}}{1-\bar{y}}\right)$ (the log-odds of the base rate).
2. **Iterate for $m = 1$ to M trees:**
   (a) **Compute Probabilities:** $p_i = \sigma(F_{m-1}(x_i))$
   (b) **Compute Gradients (Targets):** $r_{im} = y_i - p_i$
   (c) **Fit a Regression Tree:** Train $h_m(x)$ to predict r_{im}.
   (d) **Find Optimal Step Length γ_m:** Find the step length that minimizes the log-loss:

   $$\gamma_m = \arg\min_{\gamma} \sum_{i=1}^{N} L(y_i, F_{m-1}(x_i) + \gamma \cdot h_m(x_i))$$

   (This is also solved with a line search.)

   (e) **Update the Model:** $F_m(x) = F_{m-1}(x) + \gamma_m \cdot h_m(x)$
3. **Final Model:** $F_M(x)$

How to use the final model $F_M(x)$:

- **To get a probability:** Pass the raw output through the sigmoid function.

$$P(y=1|x) = \sigma(F_M(x)) = \frac{1}{1+e^{-F_M(x)}}$$

- **To get a hard class prediction:** Check the sign of the log-odds (or if the probability is ¿ 0.5).

$$\text{Class} = \begin{cases} 1 & \text{if } F_M(x) > 0 \\ 0 & \text{if } F_M(x) \leq 0 \end{cases}$$

7 Regularization & Hyperparameters

Gradient Boosting models can be very powerful, but they can also overfit the training data. To improve their generalization performance on unseen data, several regularization techniques are used, which are controlled by key hyperparameters.

7.1 Number of Trees (M)

This is the total number of boosting iterations, i.e., the number of weak learners in the final ensemble. This is the primary parameter to tune. A very large M will eventually lead to overfitting. The best value for M is typically found using **early stopping** with a validation set: you train the model for a large number of trees and stop when the performance on the validation set stops improving.

7.2 Learning Rate (Shrinkage) (ν)

In our general algorithm, we found an optimal step length γ_m for each new tree using a line search. A very common and effective regularization strategy is to take a smaller than optimal step, where we multiply the optimal step size by a factor called **shrinkage** (ν), with a typical value 0.1:

The model update step then becomes:

$$F_m(x) = F_{m-1}(x) + \nu\gamma_m \cdot h_m(x)$$

This slows down the learning process, forcing the model to take smaller steps. This requires a larger number of trees (M) to be trained, but it often leads to a much better and more robust final model. There is a direct trade-off between ν and M (a smaller ν requires a larger M).

7.3 Limiting Tree Depth

The weak learners ($h_m(x)$) are meant to be simple. We can control their complexity by limiting them. The most common method is to set a **maximum depth** (e.g., `max_depth = 4`).

- A depth of 1 creates an "additive model" where trees are just "stumps".
- A depth of 2-8 is common. Deeper trees can capture more complex feature interactions but are more computationally expensive and more likely to overfit.

Other constraints, like setting a minimum number of samples per leaf, are also used.

7.4 Subsampling

This technique introduces randomness and reduces overfitting. At each iteration m, a *fraction* of the training data (e.g., 80%) is randomly sampled *without replacement*.

- The negative gradients r_{im} are computed only for this subsample.
- The new tree $h_m(x)$ is trained only on this subsample.