Gradient Boosting from Scratch

Bhanu Prasanna Koppolu

Table of contents

Source Code

Full implementation available at ML_from_scratch/Gradient Boosting

Gradient Boosting

The Gradient Boosting algorithm is an **iterative algorithm**, where you first start with the weakest learner, then learn gradually to become a good learner.

Actually the Gradient Boosting algorithm with classification and regression is the same thing, just the **loss function varies**, that's it.

Core Concept

As the name suggests, we are going to be dealing with **Gradients**. So, whatever loss function that we are going to be dealing with must be **differentiable**.

Mainly, we have a loss function $L(y, \hat{y})$ where y is true predictions and $\hat{y}$ are the predicted values.

The Algorithm

1. Initialize $\hat{y}$ to have the average of all targets and be same size as y
2. Take gradient with respect to our loss function
3. Fit a Decision Tree Regression on the gradients (residuals)
4. Predict and update $\hat{y}$ with learning rate
5. Repeat iteratively until convergence or max iterations

We use **Decision Tree Regression** for both classification and regression tasks.

Loss Functions

For Regression - Squared Error (SSE):

$$L(y, \hat{y}) = \frac{1}{2} \sum (y - \hat{y})^2$$

$$\text{Gradient} = -(y - \hat{y})$$

For Classification - Cross Entropy:

$$L(y, \hat{y}) = -y \cdot \log(\hat{y}) - (1 - y) \cdot \log(1 - \hat{y})$$

$$\text{Gradient} = \sigma(\hat{y}) - y$$

where σ is the sigmoid function.

Implementation

Loss Classes

```
class Loss(object):
    """Base class for loss functions used in gradient boosting."""

    def loss(self, y_true, y_pred):
        """Compute the loss value."""
        ...

    def gradient(self, y_true, y_pred):
        """Compute the gradient (negative residuals)."""
        ...

class SquaredError(Loss):
    """Squared Error loss for regression tasks.

    Loss: L(y, ŷ) = 0.5 * Σ(y - ŷ)²
    Gradient: -(y - ŷ)
    """

    def loss(self, y_true, y_pred):
        ...

    def gradient(self, y_true, y_pred):
        # Returns negative residuals: -(y_true - y_pred)
        ...

class CrossEntropy(Loss):
    """Cross Entropy loss for binary classification tasks.
```

```
    Loss: L(y, ŷ) = -y*log(ŷ) - (1-y)*log(1-ŷ)
    Gradient: sigmoid(ŷ) - y
    """

    def loss(self, y_true, y_pred):
        # Clips predictions to avoid log(0)
        ...

    def gradient(self, y_true, y_pred):
        # Returns sigmoid(y_pred) - y_true
        ...
```

Gradient Boosting Class

```
class GradientBoosting:
    """Gradient Boosting for classification and regression.

    Uses an ensemble of Decision Tree Regressors fitted on
    gradients (pseudo-residuals) of the loss function.
    """

    def __init__(self,
                 n_trees=50,           # Number of boosting iterations
                 max_depth=50,         # Max depth of each tree
                 min_samples_split=5,  # Min samples to split a node
                 learning_rate=0.01,   # Step size for updates
                 regression=True,      # True=regression, False=classification
                 impurity="variance"): # Impurity measure for trees

        # Select loss function based on task type
        self.loss_func = SquaredError() if regression else CrossEntropy()

        # Initialize ensemble of Decision Tree Regressors
        self.trees = [DecisionTreeRegressor(...) for _ in range(n_trees)]

    def fit(self, X_train, y_train):
        """Fit the gradient boosting model.

        1. Initialize predictions (mean for regression, log-odds for classification)
        2. For each tree:
```

```
            - Compute gradients (pseudo-residuals)
            - Fit tree on gradients
            - Update predictions with learning rate
        """
        ...

    def predict(self, X):
        """Make predictions.

        Sum contributions from all trees, then:
        - For regression: return raw predictions
        - For classification: apply sigmoid and threshold at 0.5
        """
        ...
```