Boosting Algorithms

Lidor Erez

June 23, 2024

1 Introduction

This paper aims to provide an introduction to boosting algorithms. Boosting is a machine learning ensemble algorithm that aims to combine the predictions of several weak learners (models that are only slightly better than random guessing) to create a strong learner. The idea behind it is to train a sequence of weak learners, where each one tries to correct the mistakes of its predecessor. This way, the overall model becomes more complex and thus reduces the bias. There are several boosting algorithms, such as AdaBoost, Gradient Boosting, XGBoost, LightGBM, and CatBoost among others. In this paper, we will focus on how boosting works rather than the specific algorithms.

2 Background

The main idea in boosting is to combine the predictions of several weak learners to create a strong learner. The way it works is by training a sequence of weak learners (typically decision trees) where each one tries to correct the mistakes of its predecessor. Consider the following example: Let's say we want to classify between the two classes 1 and 0. We gather some data, conduct research, and decide we want to use boosting. Thus, we start by training a weak learner (a decision tree) on the data, denoted as f_1, and get the following results:

Data Point	**Prediction**	**Residual**
1	1	0
0	1	-1
0	0	0
1	0	1

Table 1: Residuals after the first weak learner

The weak learner f_1 has made some mistakes, so to compensate for that, we train a second weak learner f_2 while giving a bigger weight to the observations that were misclassified by the first weak learner using the residuals. The residuals are the difference between the actual value and the prediction of the model. The residuals are defined as:

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

where y_i is the actual value and $\hat{y}_i$ is the prediction of the model. The residuals for the example above are as shown in Table 1. We use these residuals to give higher weight to the observation that were misclassified by the first weak learner. For this problem, two weak learners were enough as shown in Table 2. However, most real life problems may be solved using hundreds or even thousands of trees.

Data Point	**Prediction**
1	1
0	0
0	0
1	1

Table 2: Predictions by the second weak learner f_2

2.1 Decision Trees

Decision trees are the most common weak learners used in boosting algorithms. A decision tree is a tree-like model of decisions and their possible consequences. The tree is composed of nodes and edges. The nodes represent the features in the dataset and the edges represent the decision rules. The tree is built by recursively splitting the data into subsets based on the features that give the most information gain (The split is determined by different criteria such as Gini impurity or entropy). As the tree grows, the model becomes more complex and can capture more patterns however, it is also prone to overfitting. In boosting, we use decision trees with a small depth (i.e. 3-10 levels) to prevent overfitting.

2.2 Objective Function

When using decision trees as weak learner we try to solve objective function

$$\sum_{i=1}^{n} L(y_i, \hat{y}_i) + \sum_{k=1}^{K} \Omega(f_k) + const$$

where L is the loss function, $\hat{y}_i$ is i'th the prediction of the model, $\Omega(f_k)$ is the regularization term, f_k is the k'th weak learner and K is the number of weak learners.

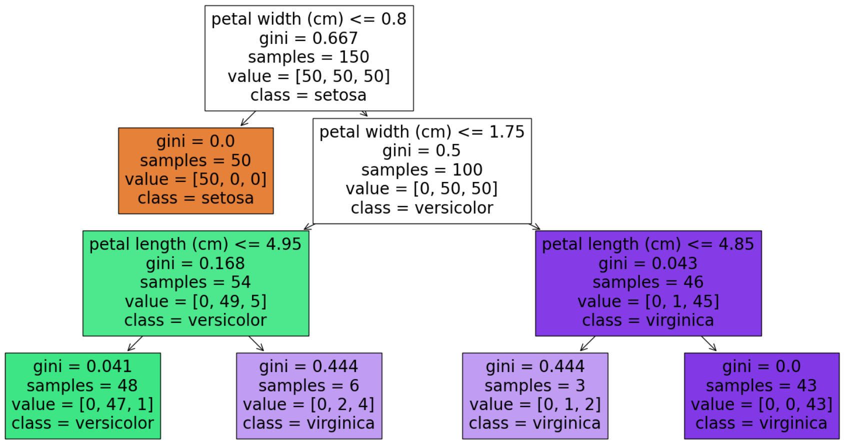

Figure 1: Decision Tree Diagram. Source: `https://scikit-learn.org/stable/auto_examples/tree/plot_iris_dtc.html`

2.2.1 Regularization Term

The regularization term $\Omega(f_k)$ is used to prevent overfitting. The regularization term is defined as:

$$\Omega(f_k) = \gamma T + \frac{1}{2}\lambda \sum_{j=1}^{T} w_j^2$$

where T is the number of leaves in the tree, w_j is the prediction of the j'th leaf and γ is the complexity parameter that penalizes the number of leaves in the tree (deep trees are prone to overfitting) and λ is the regularization parameter that penalizes the complexity of the tree which is simply the L2 regularization term on the weights of the leaves. The best γ and λ for a specific problem can be find using the cross-validation method.

3 Additive Training

Before we dive into the optimization process, we first need to define the concept of additive training. The idea behind additive training is to add a new weak learner to the model that minimizes the objective function. Let $\hat{y_i}^{(0)}$ be the initial prediction for the i'th observation. We can use additive training to define the way we make predictions.

The prediction for the i'th observation can be defined as:

$$\hat{y_i}^{(0)} = 0$$
$$\hat{y_i}^{(1)} = \hat{y_i}^{(0)} + \eta f_1(x_i)$$

$$\hat{y_i}^{(2)} = \hat{y_i}^{(1)} + \eta f_2(x_i)$$

$$.$$

$$.$$

$$.$$

$$\hat{y_i}^{(K)} = \hat{y_i}^{(k-1)} + \eta f_K(x_i)$$

Where $f_k(x_i)$ is the prediction of the k-th weak learner for the i'th observation, $\hat{y_i}^{(k-1)}$ are the predictions from previous rounds and η is the learning rate parameter which controls the contribution of each tree. So, applying additive training to our objective function, we get:

$$\sum_{i=1}^{n} L(y_i, \hat{y_i}^{(k-1)} + \eta f_K(x_i)) + \sum_{k=1}^{K} \Omega(f_k) + const$$

So, the goal is to find the weak learner f_k that minimizes the objective function which can be done using Taylor expansion.

4 Taylor Expansion

Recall the Taylor expansion of a function $f(x)$ around a point x_0 is:

$$f(x) = \sum_{n=0}^{\infty} \frac{f^{(n)}(x_0)(x - x_0)^n}{n!}$$

However, we can ignore the higher order terms and approximate the function as:

$$f(x) \approx f(x_0) + f'(x_0)(x - x_0) + \frac{f''(x_0)(x - x_0)^2}{2!}$$

We can apply the Taylor expansion to the objective function since we're always around the point $f_k(x_i)$. Thus, we get:

$$obj^{(k)} \approx \sum_{i=1}^{n} [L(y_i, \hat{y_i}^{k-1}) + g_i \eta f_k(x_i) + \frac{h_i \eta^2 f_k^2(x_i)}{2!}] + \Omega(f_k)$$

where g_i and h_i are the first and second order derivatives of the loss function with respect to the prediction from the previous round $k-1$. The derivatives are defined as:

$$g_i = \frac{\partial L(y_i, \hat{y_i}^{k-1})}{\partial \hat{y_i}^{k-1}}$$

$$h_i = \frac{\partial^2 L(y_i, \hat{y_i}^{k-1})}{\partial^2 \hat{y_i}^{k-1}}$$

4.1 Why do we use the Taylor expansion ?

In boosting, we aim to find the weak learner f_k that minimizes the objective function, which includes the loss and regularization term (as defined above). However, directly minimizing this function is difficult because decision trees are not differentiable. Taylor expansion allows us to approximate the objective function in a way that makes it easier to optimize.

4.2 How do we find the weak learner f_k ?

Different boosting algorithms use different methods to find the weak learner f_k that minimizes the objective function. Some use the gradient in their Taylor expansion, while others may use both the gradient and the hessian. This part will not be covered in this paper as I will focus on the general idea of boosting rather than the specific algorithms.

5 Boosting Process

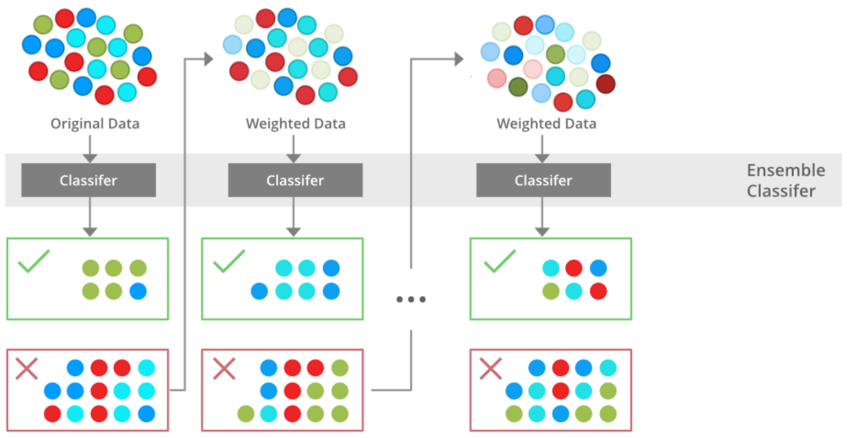

Figure 2: Boosting Process Diagram. Source: `https://www.geeksforgeeks.org/boosting-in-machine-learning-boosting-and-adaboost/`

The boosting process can be summarized as follows:

1. Initialize the prediction to a simple model, such as the mean (for regression) or the mode (for classification) of the target variable. This initial prediction is denoted as $\hat{y}_i^{(0)}$.

2. Iteratively Add Weak Learners:

(a) **Create a Subset of the Data:** by randomly sampling (without replacement) using weighted sampling.

(b) **Fit a Weak Learner:** f_k to the subset of the data with the sample weights.

(c) **Update the Predictions:** update the predictions using additive training:

$$\hat{y_i}^{(k)} = \hat{y_i}^{(k-1)} + \eta f_k(x_i)$$

(d) **Compute the Residuals:**

$$r_i = y_i - \hat{y_i}^{(k)}$$

(e) **Update the Sample Weights:** Can be done using various heuristics such as:

$$w_i = w_i * \exp(-\eta r_i)$$

3. Repeat step 2 until the number of weak learners reaches a stopping criterion:

   - A maximum number of weak learners (Iterations) K.
   - The improvement in the model's performance falls below a certain threshold.
   - Cross-validation indicates no further improvement in the model's performance.

Note: the rule of cross-validation regarding the number of trees is to choose the right number of trees that gives the best performance but also doesn't overfit the data.

6 Bibliography

- Chen, T. (2014). Introduction to boosted trees. *University of Washington Computer Science*, 22(115), 14-40.

- Meir, R., & Rätsch, G. (2003). An introduction to boosting and leveraging. In *Advanced Lectures on Machine Learning: Machine Learning Summer School 2002 Canberra, Australia, February 11–22, 2002 Revised Lectures* (pp. 118-183). Berlin, Heidelberg: Springer Berlin Heidelberg.