Boosting Algorithm: Adaptive Boosting Method (AdaBoost)

Rafiq Islam

2024-12-02

Table of contents

## Introduction

Boosting is a powerful ensemble learning technique that focuses on improving the performance of weak learners to build a robust predictive model. Now the question is what the heck is weak learner? Well, roughly speaking, a statistical learning algorithm is called a weak learner if it is slightly better than just random guess. In contrast, a statistical learning algorithm is called a strong learner if it can be made arbitrarily close to the true value. Unlike bagging (bootstrap aggregating, e.g. random forest), which builds models independently, boosting builds models sequentially, where each new model corrects the errors of its predecessors. This approach ensures that the ensemble concentrates on the difficult-to-predict instances, making boosting highly effective for both classification and regression problems.

### Key Characteristics of Boosting:

1. **Sequential Model Building:** Boosting builds one model at a time, with each model improving upon the errors of the previous one.
2. **Weight Assignment:** It assigns weights to instances, emphasizing misclassified or poorly predicted ones in subsequent iterations.
3. **Weak to Strong Learners:** The goal of boosting is to combine multiple weak learners (models slightly better than random guessing) into a strong learner.

### Mathematical Visualization

Before writing the formal algorithm, let’s do some math by hand. Say, we have a toy dataset:

|  |  | y |
| --- | --- | --- |
| 1 | 2 | 1 |
| 2 | 1 | 1 |
| 3 | 2 | -1 |
| 4 | 3 | -1 |

Here:

* and are features.
* is the target label, with values or .

Now, let’s apply the AdaBoost algorithm step-by-step using this dataset.

#### Iteration 1

*Step 1: Initialize Weights*  
Initially, all data points are assigned equal weights:

Weights: .

*Step 2: Train Weak Learner*

Suppose we use a decision stump (a simple decision rule) as the weak learner. The first decision stump might split on as:

* Predict if , otherwise .

Note, that even though we are deciding based on the feature , however, for learner, is the row from the data set, i.e. . Therefore, for would mean that, we are feeding first row to the learner at iteration 1.

*Step 3: Evaluate Weak Learner*

Predictions for the dataset:

But our true labels are . So the error

where, is an indicator function that equals 1 when the prediction is incorrect and 0 otherwise. Therefore, in iteration 1:

*Step 4: Calculate*

*Step 5: Update Weights:*

For each instance:

Now you may wonder how and from where we came up with this updating rule? We will explain this update process in the next post, but for now let’s just focus on the update.

Updated weights (before normalization):

Normalize to ensure the weights sum to 1:

Final normalized weights: . Notice that, for the incorrect prediction, the weight increased and for the correct prediction the weights decreased.

#### Iteration 2

Similarly, we proceed with second iteration with the following weak learner:

For this learner, the prediction

where as the actual labels are . So, the error

and

Next, we update the weights

So, and after normalizing . The final ensemble model combines the weak learners using their weights ():

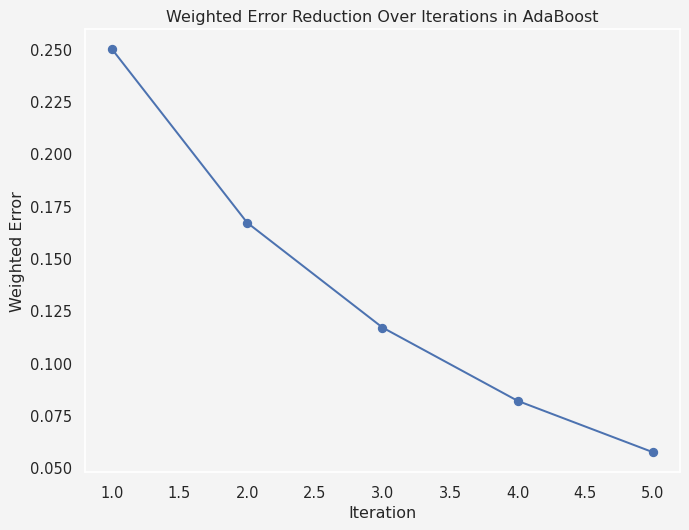
For the toy dataset:

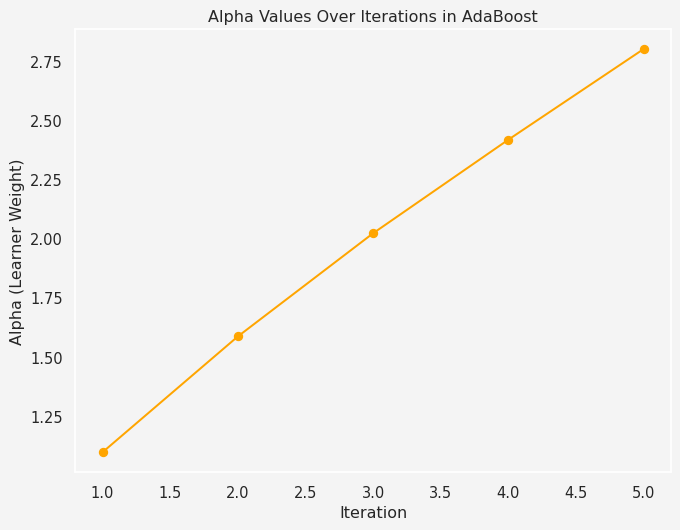
1. ,
2. ,

Weighted predictions:

If we keep iterating this way, we will have

import matplotlib.pyplot as plt  
import numpy as np   
from mywebstyle import plot\_style  
plot\_style('#f4f4f4')  
  
# Data points for visualization  
iterations = [1, 2]  
errors = [0.25, 0.167] # Errors from the two iterations  
alphas = [1.0968, 1.586] # Alpha values for the weak learners  
  
# Extend to further iterations  
# Simulating error reduction and alpha calculation for a few more iterations  
for i in range(3, 6): # Iterations 3 to 5  
 new\_error = errors[-1] \* 0.7 # Simulating decreasing errors  
 errors.append(new\_error)  
 alphas.append( np.log((1 - new\_error) / new\_error))  
 iterations.append(i)  
  
# Plot weighted errors over iterations  
plt.figure(figsize=(8, 6))  
plt.plot(iterations, errors, marker='o')  
plt.title("Weighted Error Reduction Over Iterations in AdaBoost")  
plt.xlabel("Iteration")  
plt.ylabel("Weighted Error")  
plt.grid()  
plt.show()  
  
# Plot alpha values (importance of weak learners)  
plt.figure(figsize=(8, 6))  
plt.plot(iterations, alphas, marker='o', color='orange')  
plt.title("Alpha Values Over Iterations in AdaBoost")  
plt.xlabel("Iteration")  
plt.ylabel("Alpha (Learner Weight)")  
plt.grid()  
plt.show()





## Adaptive Boosting (AdaBoost) Algorithm

Now it’s time to write the formal algorithm for Adaptive Boosting or AdaBoost method. It is one of the earliest and most widely used boosting algorithms. It was introduced by Freund and Schapire in 1996. AdaBoost combines weak learners, typically decision stumps (single-level decision trees), to form a strong learner.

| **Algorithm:** AdaBoost |
| --- |
| 1. Initialize the observation weights for 2. For to :   (a) Fit a classifier to the training data using weights   (b) Compute    (c) Compute   (d) Set $w\_i \rightarrow w\_i\cdot \exp{\left[\alpha\_m\cdot\mathbb{1}(y\_i\ne G\_m(x\_i))\right]},\hspace{2mm} i=1,2,\cdots, N$ 3. Output |

In the next posts, we will continue discussing on this algorithm, specially the loss function, optimization techniques, advantages and limitations of AdaBoost, and many other facts about this algorithm.

Thanks for reading this.

## Reference

* Hastie, Trevor, Robert Tibshirani, and Jerome Friedman. “The elements of statistical learning: data mining, inference, and prediction.” (2017).

**Share on**

**You may also like**