Understanding AdaBoost: The Power of Teamwork in Machine Learning

An introduction to AdaBoost as an effective meta-classifying technique.

Hernández Pérez Erick Fernando

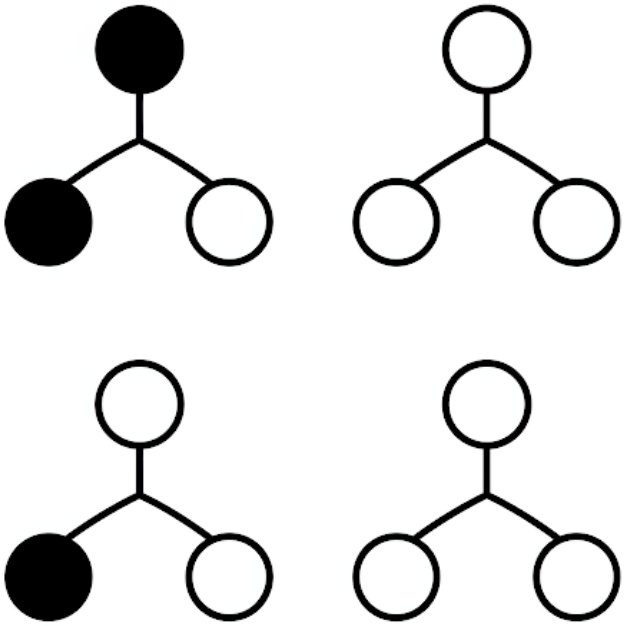

Figure 1

Imagine you're trying to make an important decision, but instead of relying on just one opinion, you ask a group of experts. Some of them might not be very reliable on their own, but together, their combined insights lead to a strong and accurate conclusion. That's the basic idea behind AdaBoost, short for Adaptive Boosting.

AdaBoost is a meta classifier, meaning it doesn't work alone—it improves the performance of other classifiers by combining them in a smart way. It starts with a simple model (often a weak learner, like a decision stump), checks where it makes mistakes, and then adjusts its focus to learn from those errors. It repeats this process multiple times, creating a powerful team of models that work together to make more accurate predictions.

Think of it like training a group of students for a test. If one student struggles with math, the teacher gives them extra attention in that area. Over time, the entire class gets better because weaknesses are systematically addressed. AdaBoost does something similar—it learns from mistakes and continuously improves.

While AdaBoost is simple to understand, it has proven to be incredibly effective in various applications, from image recognition to fraud detection. It's a great example of how a little teamwork can go a long way in machine learning!

The basics concepts

Before delving into the complexities of Boosting and related concepts, it is essential to revisit the classification problem. Although we have touched upon these ideas in previous sections, the approach we will present here provides a more suitable framework for understanding them.

According to Ganaie et al. (2022) [1], the classification problem is defined as the task of categorizing new observations based on the hypothesis learned from the training data set. The hypothesis represents a mapping of input data features to their corresponding target labels/classes. Our primary objective is for the hypothesis

to approximate the true unknown function as closely as possible, thereby minimizing generalization error. In simpler terms, it should perform better when applied to unseen data.

> Boosting began with a question from Michael Kearns, about whether a weak learning algorithm can be transformed into a strong learning algorithm. Suppose a learning algorithm is only guaranteed, with high probability, to be slightly more accurate than random guessing. Is it possible, despite how weak this algorithm is, to use it to create a classifier whose error rate is arbitrarily close to 0? (Rudin, 2012) [4]

According to Morales (2015) [2], the term boosting refers to a type of algorithm whose goal is to create a strong hypothesis by combining simple, weak hypotheses [As we saw with Rudin, a weak hypothesis works slightly than chance.]. These boosting techniques all share the common objective of improving a weak classifier's performance by focusing on its mistakes, adjusting the learning process, and combining models to build a much stronger classifier.

AdaBoost

> Adaboost uses a greedy technique for minimizing a convex surrogate function upper bounded by misclassification loss via augmentation, at each iteration, the current model with the appropriately weighted predictor. AdaBoost learns an efective ensemble classifier as it leverages the incorrectly classified sample at each stage of the learning. (Ganaie et al., 2022)[1]

Yes, I know, it's a lot to take in, and it might feel a bit overwhelming at first. The concepts can be tricky, and it's not always easy to break them down into simple words. But don't worry! The key is to take it step by step. Once we start connecting the dots, everything will begin to make more sense.

At first, AdaBoost starts with a weak model (for example, a simple decision tree) and tries to make predictions. Some of those predictions will be wrong. The key part of AdaBoost is that it focuses on the mistakes it made and gives more importance (or weight) to those incorrectly classified examples in the next round.

In each round, AdaBoost adjusts the model to pay more attention to the examples it got wrong. It keeps doing this over several rounds, and by the end, it combines all these "learned" models to create a much stronger model that performs better overall.

Think of it like a group project where each member of the team makes a mistake at first, but as they go along, they learn from each other's mistakes, eventually creating a much better final result!

Now it's time to get a little more serious. We will follow the approach outlined by Rojas (2009) [3] (An incredible work).

Assume that you are working on a two-class pattern recognition problem for which you are given a large pool of weak classifiers $\mathcal{H} = \{h(x)\}$. You want to put together a "dream team" of classifiers extracted from the pool.

For a given pattern x_i, each classifier h_j gives its verdict $h_j(x_i) \in \{-1, 1\}$. The final result is $\text{sign}(H(x_i))$, the sign of the weighted sum of expert opinions, where:

$$H(x_i) = \alpha_1 h_1(x_i) + \alpha_2 h_2(x_i) + ... + \alpha_m h_m(x_i) = \sum_{j=1}^{M} \alpha_j h_j(x_i)$$

Where α_i are weights we assign for the "opinions" of each classifier.

Let's assume that our pool of classifiers has a total of L classifiers (Others scholars uses the most common of infinity, but we want to keep it simple).

AdaBoost has three main steps: Scouting → Drafting → Weighting.

Scouting

Scouting is performed by testing the classifiers in the pool using a training set T of N multidimensional data points x_i. Each point x_i has an associated label $y_i = 1$ or $y_i = -1$.

We evaluate and rank all classifiers in the expert pool by assigning:

- A cost e^{β} for each misclassification (miss).

- A cost $e^{-\beta}$ for each correct classification (hit).

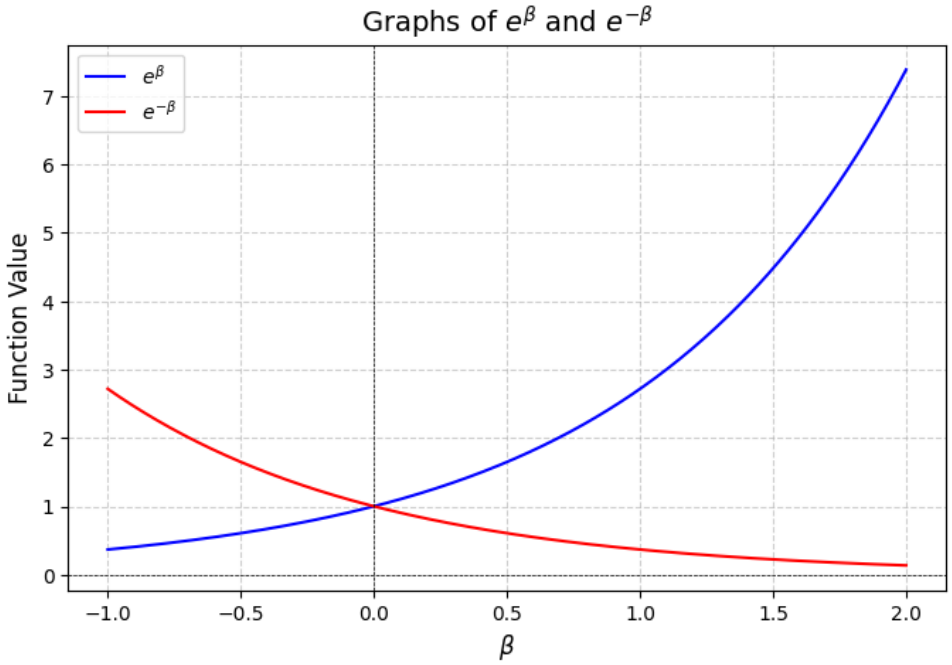

Figure 2: e^{β} and $e^{-\beta}$

We require that $\beta > 0$ so that misclassifications are penalized more than correct classifications. Although it may seem unusual to penalize a correct classification, as long as the success penalty is smaller than the penalty for a miss ($e^{-\beta} < e^{\beta}$), the system remains ok.

This type of error function, which differs from the usual squared Euclidean distance to the classification target, is known as an **exponential loss function**. AdaBoost employs exponential loss as its error criterion.

AdaBoost works iteratively, selecting one classifier from the available pool in each of the M iterations. Throughout this process, the data set elements are assigned weights that indicate their importance or urgency at each stage. Initially, all elements are given the same weight, $\frac{1}{N}$

As the drafting progresses, the more challenging examples, those where the committee is still performing poorly, are given larger and larger weights. The drafting process focuses on selecting new classifiers for the committee that can help with the still misclassified examples. It would be redundant to draft a classifier that always, or almost always, gives the same output as a classifier already drafted. If we wanted to draft the same classifier again, we could simply duplicate its weight.

Drafting

In each iteration we need to rank all classifiers, so that we can select the current best out of the pool. At the m-th iteration we have already included $m-1$ classifiers. The current linear combinations is:

$$H_{m-1}(x_i) = \alpha_1 h_1(x_i) + \alpha_2 h_2(x_i) + ... + \alpha_{m-1} h_{m-1}(x_i)$$

and we want to extend it to

$$H_m(x_i) = H_{m-1}(x_i) + \alpha_m h_m(x_i)$$

Now, we define the total cost, or total error, of the extended classifier as the exponential loss

$$E = \sum_{i=1}^{N} e^{-y_i(H_{m-1}(x_i)+\alpha_m h_m(x_i))}$$

where α_m and h_m are yet to be determined in an optimal way. Since our intention is to draft h_m we rewrite

$$E = \sum_{i=1}^{N} w_i^{(m)} e^{-y_i(\alpha_m h_m(x_i))}$$

And now we can split de sum:

$$E = \sum_{y_i = h_m(x_i)} w_i^{(m)} e^{-\alpha_m} + \sum_{y_i \neq h_m(x_i)} w_i^{(m)} e^{\alpha_m}$$

The total cost is the weighted cost of all hits plus the weighted cost of all misses. And simplify a little bit:

$$E = \sum_{y_i = h_m(x_i)} w_i^{(m)} e^{-\alpha_m} + \sum_{y_i \neq h_m(x_i)} w_i^{(m)} e^{\alpha_m} = W_c e^{-\alpha_m} + W_e e^{\alpha_m}$$

For the selection of h_m the exact value of $\alpha_m > 0$ is irrelevant, since for a fixed α_m minimizing E is equivalent to minimizing $e^{\alpha_m} E$ and because

$$e^{\alpha_m} E = W_c + W_e e^{2\alpha_m}$$

Since $e^{2a_m} > 1$, we can rewrite the above expression as

$$e^{\alpha_m} E = (W_c + W_e) + W_e(e^{2\alpha_m} - 1)$$

Now, $(W_c + W_e)$is the total sum W of the weights of all data points, that is, a constant in the current iteration. The right hand side of the equation is minimized when at the m-th iteration we pick the classifier with the lowest total cost W_e

Weighting

Now, having picked the m-th member of the committee we need to determine its weight α_m. Reviewing the following equation

$$E = W_c e^{-\alpha_m} + W_e e^{\alpha_m}$$

We can immediately see that

$$\frac{dE}{d\alpha_m} = -W_c e^{-\alpha_m} + W_e e^{\alpha_m}$$

Equating this expression to zero and multiplying by e^{α_m} we obtain
$-W_c + W_e e^{2\alpha_m} = 0$
The optimal α_m is thus:

$$\alpha_m = \frac{1}{2} \ln\left(\frac{W_c}{W_e}\right)$$

And using the fact that $W = W_c + W_e$

$$\alpha_m = \frac{1}{2} \ln\left(\frac{W - W_e}{W_e}\right) = \frac{1}{2} \ln\left(\frac{1 - e_m}{e_m}\right)$$

Where $e_m = \frac{W_e}{W}$ is the percentage rate of error given the weights of the data points.

Pseudocode

Given a training set T consisting of data points x_i and their corresponding labels y_i in a two-class classification problem $(+1, -1)$, we start by assigning an initial weight of $w_i^{(1)} = 1$ to every data point x_i. Our goal is to create a committee of M classifiers selected from a pool of classifiers. We carry out M iterations, and at each iteration, let W represent the total weight of all data points, while W_e denotes the total weight of the data points where the current classifier makes an incorrect prediction.

Algorithm 1 AdaBoost

1: **for** $m = 1$ to M **do**

2: Select and extract from the pool of classifiers the classifier h_k which minimizes

$$W_e = \sum_{y_i \neq h_m(x_i)} w_i^m$$

3: Set the weight α_m of the classifier to

$$\alpha_m = \frac{1}{2} \ln \left(\frac{1 - e_m}{e_m} \right)$$

where $e_m = \frac{W_e}{W}$

4: Update the weights of the data points for the next iteration. If $h_m(x_i)$ is a miss, set

$$w_i^{(m+1)} = w_i^{(m)} e^{\alpha_m} = w_i^{(m)} \sqrt{\frac{1 - e_m}{e_m}}$$

Otherwise

$$w_i^{(m+1)} = w_i^{(m)} e^{-\alpha_m} = w_i^{(m)} \sqrt{\frac{e_m}{1 - e_m}}$$

5: **end for**

References

[1] Ganaie, M., Hu, M., Malik, A., Tanveer, M. & Suganthan, P. (2022). *Ensemble deep learning: A review.* Engineering Applications of Artificial Intelligence, 115, 105151. https://doi.org/10.1016/j.engappai.2022.105151

[2] Morales, A. (2015). *Uso de características no lineales para identificar llantos de recién nacidos con un conjunto clasificador* (Bachelor's thesis). Universidad de las Américas Puebla. https://catarina.udlap.mx/u_dl_a/tales/documentos/lmt/morales_s_aa/

[3] Rojas, R. (2009). *AdaBoost and the Super Bowl of classifiers: A tutorial introduction to adaptive boosting.* Computer Science Department, Freie Universität Berlin. https://www.inf.fu-berlin.de/inst/ag-ki/adaboost4.pdf

[4] Rudin, C. (2012). *Boosting (MIT 15.097 course notes).* Massachusetts Institute of Technology. https://ocw.mit.edu/courses/15-097-prediction-machine-learning-and-statistics-spring-2012/bd1326207712d907fe3418bcc9043b55_MIT15_097S12_lec10.pdf