CHAPTER 15

AdaBoost

> "AdaBoost builds a strong classifier by combining many weak classifiers, reweighting the data at each step so the model focuses on harder-to-classify examples."

15.1 Introduction

AdaBoost, short for Adaptive Boosting, is one of the most influential ensemble methods in machine learning. Introduced by Freund and Schapire [1], it has been widely used for classification tasks, notably in computer vision and text categorization. The algorithm iteratively trains weak learners, typically decision stumps, and combines them into a single, strong classifier. Despite its conceptual simplicity, AdaBoost has a deep theoretical foundation and has been shown to be effective even when the weak learners are only slightly better than random guessing.

15.2 Intuitive Explanation (Without Math)

15.2.1 Intuition & Examples

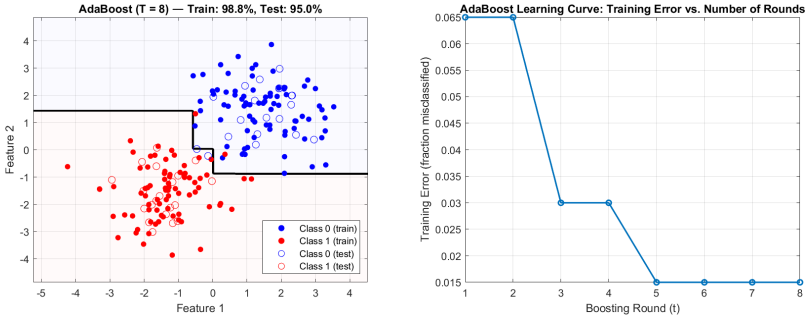
AdaBoost can be thought of as a "committee" of weak classifiers. Imagine you have a group of experts (each a weak classifier) who are not very accurate individually. However, by combining their opinions and giving more weight to experts who are more reliable on the hard-to-classify examples, the overall decision becomes very accurate.

For instance, consider a binary classification problem (e.g., distinguishing spam from non-spam emails). AdaBoost begins by training a simple classifier (such as a decision stump) on the data. It then identifies the misclassified emails and increases their weight so that the next classifier focuses

more on these "difficult" cases. This process repeats over several rounds. The final prediction is based on a weighted vote of all the classifiers.

15.2.2 Graphical Illustration

Figure 15.1 provides an illustration. Each weak classifier is depicted as a small decision boundary; misclassified points are emphasized, and their influence is increased in subsequent iterations.



(a) (b)

Figure 15.1: An illustration of AdaBoost classification: (a) after a few iterations the decision boundary (black line) is calculated for classification of new "test" data based on learning the "training" data for a two-class dataset, (b) Training error vs. boosting round (learning curve).

15.3 Tutorial: Implementation From Scratch

In this section, we provide a practical step-by-step tutorial for implementing a simple AdaBoost algorithm. Complete – from-scratch – MATLAB, Python, and R implementations are available on the companion website.[1]

15.3.1 Pseudo-Code

Algorithm 1 is the pseudo-code of the equations defined in Section 15.4.

[1] `https://pfroysdon.github.io/publications/`

Algorithm 1 AdaBoost Algorithm for Binary Classification

Input: Training data $\{(\mathbf{x}_i, y_i)\}_{i=1}^n$ with $y_i \in \{-1, +1\}$, number of rounds T

Initialize weights $w_i^{(1)} = \frac{1}{n}$ for all i

for $t = 1$ to T **do**

Train weak classifier $h_t(\mathbf{x})$ using weights $\mathbf{w}^{(t)}$

Compute weighted error:

$$\epsilon_t = \sum_{i=1}^{n} w_i^{(t)} \mathbf{1}\{h_t(\mathbf{x}_i) \neq y_i\}$$

Compute classifier weight:

$$\alpha_t = \tfrac{1}{2} \ln\left(\tfrac{1-\epsilon_t}{\epsilon_t}\right)$$

Compute normalization constant:

$$Z_t = \sum_{j=1}^{n} w_j^{(t)} \exp(-\alpha_t y_j h_t(\mathbf{x}_j))$$

Update sample weights:

$$w_i^{(t+1)} = \frac{w_i^{(t)} \exp(-\alpha_t y_i h_t(\mathbf{x}_i))}{Z_t}, \quad i = 1, \ldots, n$$

Normalize so that $\sum_i w_i^{(t+1)} = 1$

(Optionally) Update additive model: $F_t(\mathbf{x}) = F_{t-1}(\mathbf{x}) + \alpha_t h_t(\mathbf{x})$

end for

Output: Final classifier:

$$H(\mathbf{x}) = \text{sign}\left(\sum_{t=1}^{T} \alpha_t h_t(\mathbf{x})\right)$$

15.3.2 Code Implementation

The code below is a *simple AdaBoost* algorithm that implements the steps outlined in Algorithm 1. The code for this section is extensive so we will only cover the key elements. For brevity, we use the "$\cdots$" to denote the omission of some code.

Let's begin by defining the main function for `adaboostTrain()` that trains AdaBoost with decision stumps.

```
function [stumps, alphas] = adaboostTrain(X, y, T, eta)
  ...
  for t = 1:T
    % Train a decision stump with the current weights
    [stump, error, pred] = decisionStump(X, y, D);
    % Avoid division by zero (or log of zero)
    error = max(error, 1e-10);
    alpha = eta * 0.5 * log((1 - error) / error);
    stumps{t} = stump;
    alphas(t) = alpha;
    % Update the weights:
    % Increase weights on misclassified points.
    D = D .* exp(-alpha * y .* pred);
    D = D / sum(D);  % Normalize to sum to 1.
  end
end
```

Next, define the function `decisionStump()`.

```
function [bestStump, bestError, bestPred] = decisionStump
    (X, y, D)
  ...
  % Loop over each feature
  for j = 1:d
  % Get unique values in feature j for thresholds
    thresholds = unique(X(:,j));
    for i = 1:length(thresholds)
      thresh = thresholds(i);
      % Try both possible polarity assignments: 1 and -1.
      for polarity = [1, -1]
        % Make predictions using the decision rule
        if polarity == 1
          pred = ones(n, 1);
          pred(X(:,j) >= thresh) = -1;
        else
          pred = -ones(n, 1);
          pred(X(:,j) >= thresh) = 1;
        end
        % Compute the weighted error
        err = sum(D .* (pred ~= y));
        ...
      end
    end
  end
end
```

Tutorial Explanation

The explanation below summarizes how the full from-scratch code implementations on the companion website map directly onto the pseudo-code and code snippet above and to the equations in the following section – bridging the gap between the math and the executable code.

Data Generation

Two classes are generated from Gaussian distributions: Class 0 is centered at $(2, 2)$, with label 0; Class 1 is centered at $(-2, -2)$, with label 1. A total of 100 samples per class are generated, then combined and shuffled.

AdaBoost Training & Prediction

The function `adaboostTrain()` employs the following steps:

1. **Initialization:** All samples are assigned equal weights.
2. **Iteration:** For a specified number of boosting iterations (here, $T = 50$): A decision stump is trained on the weighted data. weighted error of the stump is computed. The weight α for the stump is calculated. The sample weights are updated by increasing the weights of misclassified examples.
3. **Ensemble Construction:** The final classifier is an ensemble of all the weak classifiers, where predictions are aggregated using weighted majority voting.

The function `adaboostPredict()` then applies the ensemble model to input data, aggregating the predictions of each decision stump weighted by its corresponding α value.

Expected Results

- **Classifier Performance:** The AdaBoost ensemble is expected to achieve high classification accuracy on the synthetic dataset because the two classes are well-separated.
- **Visualization:** Figure 15.1a plots the decision regions with different colors (e.g., blue for class 0 and red for class 1) overlaid on a scatter plot of the original data points and the decision boundary. Figure 15.1b plots the learning curves for both *test* and *train* versus boosting round.

Pro Tips ★

Here are a few things to remember about AdaBoost:

- `nEstimators` (number of boosting rounds)
  - Too few: underfit, limited capacity.
  - Too many: risk overfitting, especially with noisy data.
- `eta` (learning rate)
  - Lower values (0.01-0.1): require more estimators, but increase stability.
  - Higher values (>1): can cause oscillations or overfitting.
- Sensitivity to noise and outliers
  - AdaBoost upweights misclassified points – including mislabeled noise. Performance can degrade badly if labels are noisy.
  - Preprocess carefully: remove or clean outliers before training.
- Training stability checklist
  - Start with `nEstimators=100`, `eta`=0.1–1.0, and depth-1 stumps.
  - Monitor training vs. validation error – divergence suggests overfitting or noisy data.
  - Use early stopping with validation error.
  - If overfitting quickly, reduce `eta` or `nEstimators`.
  - If underfitting, increase estimator depth slightly (to 2-3) or add more rounds.

15.4 Mathematical Derivations & Proofs

15.4.1 Introduction

AdaBoost constructs a *strong* classifier by additive combination of *weak* classifiers. At each round it (i) reweights the training samples to emphasize hard (misclassified) points, (ii) fits a weak learner to minimize the *weighted* error, (iii) chooses a step size by minimizing an exponential surrogate loss, and (iv) updates the sample weights multiplicatively. We derive the optimal weak-learner weight, the weight-update rule, an error bound, and present the functional-gradient view that motivates the procedure.

15.4.2 Data & Notation

Let the training set be

$$\mathcal{D} = \{(\mathbf{x}_i, y_i)\}_{i=1}^{n}, \qquad \mathbf{x}_i \in \mathbb{R}^d \text{ (column vector)}, \quad y_i \in \{-1, +1\} \text{ (scalar)}.$$

At boosting round $t = 1, \ldots, T$:

- $\mathbf{w}^{(t)} = (w_1^{(t)}, \ldots, w_n^{(t)})^\top \in \mathbb{R}^n$ are nonnegative sample weights with $\sum_i w_i^{(t)} = 1$; initialize $w_i^{(1)} = \frac{1}{n}$.

- $h_t : \mathbb{R}^d \to \{-1, +1\}$ is the selected weak classifier.
- $\alpha_t \in \mathbb{R}$ is its coefficient in the additive model.

The *additive* score function and the final classifier are

$$F_T(\mathbf{x}) \;=\; \sum_{t=1}^{T} \alpha_t \, h_t(\mathbf{x}), \qquad H(\mathbf{x}) \;=\; \operatorname{sign}\big(F_T(\mathbf{x})\big). \tag{15.1}$$

15.4.3 Model Formulation

AdaBoost can be obtained by forward stagewise minimization of the empirical *exponential* loss

$$\widehat{R}_{\exp}(F) \;=\; \frac{1}{n} \sum_{i=1}^{n} \exp\big(-y_i F(\mathbf{x}_i)\big). \tag{15.2}$$

Given F_{t-1}, we add a new term αh by solving

$$(\alpha_t, h_t) \in \arg\min_{\alpha \in \mathbb{R},\, h} \; \frac{1}{n} \sum_{i=1}^{n} \exp\big(-y_i (F_{t-1}(\mathbf{x}_i) + \alpha\, h(\mathbf{x}_i))\big). \tag{15.3}$$

Introduce the normalized weights

$$w_i^{(t)} \;\triangleq\; \frac{\exp\big(-y_i F_{t-1}(\mathbf{x}_i)\big)}{\sum_{j=1}^{n} \exp\big(-y_j F_{t-1}(\mathbf{x}_j)\big)}, \qquad \sum_{i} w_i^{(t)} = 1, \tag{15.4}$$

under which Eqn. (15.3) is proportional to the *partition function*

$$Z_t(\alpha, h) \;=\; \sum_{i=1}^{n} w_i^{(t)} \, \exp\big(-\alpha\, y_i h(\mathbf{x}_i)\big). \tag{15.5}$$

15.4.4 Optimal Step Size & Choice of Weak Learner

For a discrete weak learner $h : \mathbb{R}^d \to \{-1, +1\}$, define its *weighted error*

$$\epsilon_t(h) \;=\; \sum_{i=1}^{n} w_i^{(t)} \, \mathbf{1}\{y_i \neq h(\mathbf{x}_i)\} \;=\; \frac{1 - \sum_i w_i^{(t)} \, y_i h(\mathbf{x}_i)}{2}. \tag{15.6}$$

Because $y_i h(\mathbf{x}_i) \in \{\pm 1\}$,

$$Z_t(\alpha, h) \;=\; (1 - \epsilon_t(h))\, e^{-\alpha} + \epsilon_t(h)\, e^{\alpha}.$$

Line search (optimal α_t)

Differentiating and setting $\frac{\partial Z_t}{\partial \alpha} = 0$ gives

$$\boxed{\alpha_t^\star \;=\; \frac{1}{2}\ln\!\left(\frac{1-\epsilon_t}{\epsilon_t}\right)}, \qquad \epsilon_t \equiv \epsilon_t(h_t). \tag{15.7}$$

Plugging Eqn. (15.7) back into Eqn. (15.6) yields the minimized partition value

$$Z_t^\star(h) \;=\; 2\sqrt{\epsilon_t(h)\,(1-\epsilon_t(h))}. \tag{15.8}$$

Hence minimizing $Z_t^\star(h)$ is equivalent to minimizing the weighted error $\epsilon_t(h)$; choose

$$h_t \in \arg\min_h \;\epsilon_t(h) \quad \text{under weights } \mathbf{w}^{(t)}. \tag{15.9}$$

15.4.5 Sample-Weight Update (Emphasizing Hard Examples)

With (α_t, h_t) chosen, update the weights multiplicatively and renormalize:

$$w_i^{(t+1)} = \frac{w_i^{(t)}\,\exp\!\big(-\alpha_t\, y_i h_t(\mathbf{x}_i)\big)}{Z_t},$$
$$Z_t = \sum_{j=1}^{n} w_j^{(t)}\,\exp\!\big(-\alpha_t\, y_j h_t(\mathbf{x}_j)\big). \tag{15.10}$$

Thus correctly classified points ($y_i = h_t(\mathbf{x}_i)$) get down-weighted by $e^{-\alpha_t}$, while misclassified points get up-weighted by $e^{+\alpha_t}$.

15.4.6 Training-Error Bound & Edge

The empirical training error of $H(\mathbf{x}) = \operatorname{sign}(F_T(\mathbf{x}))$ obeys

$$\frac{1}{n}\sum_{i=1}^{n} \mathbf{1}\{y_i \neq H(\mathbf{x}_i)\} \;\le\; \prod_{t=1}^{T} Z_t^\star \;=\; \prod_{t=1}^{T} 2\sqrt{\epsilon_t(1-\epsilon_t)}. \tag{15.11}$$

Let the *edge* be $\gamma_t \triangleq \frac{1}{2} - \epsilon_t$. Then $Z_t^\star = \sqrt{1-4\gamma_t^2} \le e^{-2\gamma_t^2}$ and

$$\frac{1}{n}\sum_{i=1}^{n} \mathbf{1}\{y_i \neq H(\mathbf{x}_i)\} \;\le\; \exp\!\Big(-2\sum_{t=1}^{T}\gamma_t^2\Big), \tag{15.12}$$

so any sequence with $\epsilon_t < \frac{1}{2}$ reduces the bound exponentially.

15.4.7 Functional-Gradient View

The gradient of Eqn. (15.2) at the training points is

$$\frac{\partial \widehat{R}_{\exp}}{\partial F}(\mathbf{x}_i) \;=\; -\frac{1}{n}\, y_i \, \exp\big(-y_i F(\mathbf{x}_i)\big) \;\propto\; -y_i \, w_i^{(t)}.$$

Thus, choosing h_t to minimize $\sum_i w_i^{(t)} \mathbf{1}\{y_i \neq h(\mathbf{x}_i)\}$ is a projection of the negative gradient onto the class of weak learners, while Eqn. (15.7) is the exact line search along h_t.

15.4.8 Variants

Real AdaBoost (confidence-rated)

Allow $h_t : \mathbb{R}^d \to \mathbb{R}$ (e.g., leaf-wise real scores). Minimizing $\sum_i w_i^{(t)} e^{-y_i h_t(\mathbf{x}_i)}$ yields, for a region R (e.g., a leaf),

$$h_t^\star|_R \;=\; \tfrac{1}{2} \log \frac{\sum_{i \in R: y_i = +1} w_i^{(t)}}{\sum_{i \in R: y_i = -1} w_i^{(t)}},$$

and one can take $\alpha_t = 1$ (absorbed into h_t).

Multiclass (SAMME)

For K classes and discrete h_t, the coefficient becomes

$$\alpha_t \;=\; \log \frac{1-\epsilon_t}{\epsilon_t} + \log(K-1), \quad \widehat{y}(\mathbf{x}) = \arg\max_k \sum_{t=1}^{T} \alpha_t \, \mathbf{1}\{h_t(\mathbf{x}) = k\}.$$

15.4.9 Summary

From first principles: AdaBoost is the forward stagewise minimizer of the empirical exponential risk: (i) define weights from the current additive model Eqn. (15.4); (ii) pick h_t by minimizing weighted error Eqn. (15.9); (iii) take the exact line-search step Eqn. (15.7); (iv) update weights multiplicatively Eqn. (15.10); (v) aggregate predictions as in Eqn. (15.1). The product-of-partition-functions bound Eqn. (15.11) shows exponential decay of training error when each weak learner has edge $\gamma_t > 0$.

15.5 Advantages, Disadvantages, Alternatives, & Applications

Advantages

- ✓ **Improved Accuracy:** AdaBoost can combine many weak classifiers to achieve high accuracy.
- ✓ **Flexibility:** It is applicable to a wide range of classification tasks.
- ✓ **Theoretical Guarantees:** AdaBoost has strong theoretical foundations, including bounds on training error.

Disadvantages

- ✗ **Sensitivity to Noisy Data:** AdaBoost may overfit noisy datasets or outliers.
- ✗ **Weak Learner Dependency:** Its performance depends on the quality of the weak classifiers.

Alternatives

- ↬ **Random Forests:** An ensemble method that builds multiple decision trees using bagging and random feature selection.
- ↬ **Support Vector Machines (SVM):** A robust alternative for classification tasks.

Real-World Applications & Use Cases

- **Computer Vision:** Face detection and object recognition.
- **Text Classification:** Spam filtering and sentiment analysis.
- **Medical Diagnosis:** Classifying patient data for disease prediction.

15.6 Conclusion

AdaBoost remains a powerful and widely used boosting technique in many machine learning domains. The seminal paper by Freund and Schapire [1] and further discussions in Hastie, Tibshirani, and Friedman's *The Elements of Statistical Learning* [2] have established AdaBoost as one of the most widely cited and robust algorithms in machine learning.

Bibliography

[1] Y. Freund and R. E. Schapire, "A decision-theoretic generalization of on-line learning and an application to boosting," *Journal of Computer and System Sciences*, vol. 55, no. 1, pp. 119–139, 1997.

[2] T. Hastie, R. Tibshirani, and J. Friedman, *The Elements of Statistical Learning*. Springer, 2009, this seminal work has been cited over 10,000 times.