ADAPTIVE BOOSTING (AdaBoost)

From Decision Stumps to a Strong Classifier

The Three Main Ideas Behind AdaBoost

1. Weak Learners

AdaBoost combines a lot of **weak learners** to make classifications. The weak learners are almost always **stumps**.

2. Weighted Voting

Some stumps get **more say** in the classification than others based on their performance.

3. Sequential Learning

Each stump is made by taking the **previous stumps' mistakes** into account.

Key Insight: AdaBoost is different from Random Forest because it uses stumps instead of full trees, gives different weights to different stumps, and builds stumps sequentially rather than independently.

Decision Stumps: The Weak Learners

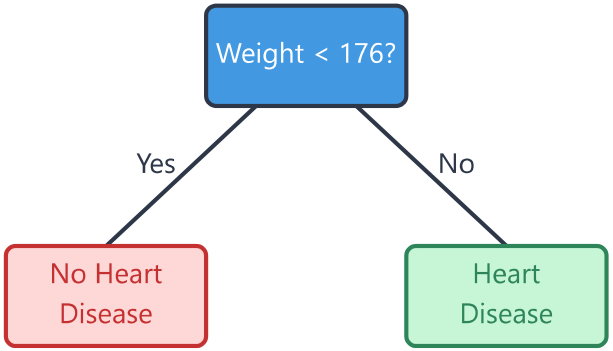
Random Forest

- Uses full-sized decision trees
- Each tree uses all variables
- Trees are independent
- Equal voting power

AdaBoost

- Uses decision stumps
- Each stump uses only ONE variable
- Stumps are sequential/dependent
- Weighted voting power

Figure 1: Decision Stump Structure



A decision stump = one internal node + two leaf nodes

Important: Stumps are technically "weak learners" - they're not great at making accurate classifications on their own, but AdaBoost combines many of them to create a strong classifier.

Example Dataset: Heart Disease Prediction

Sample	Chest Pain	Blocked Arteries	Patient Weight	Heart Disease	Initial Weight
1	Yes	Yes	205	Yes	1/8
2	No	No	180	No	1/8
3	Yes	No	210	Yes	1/8
4	Yes	Yes	167	Yes	1/8
5	No	No	156	No	1/8
6	No	Yes	125	No	1/8
7	No	No	168	No	1/8
8	Yes	No	172	Yes	1/8

Initial Sample Weights

At the start, all samples get the same weight:

$$w_i^{(1)} = \frac{1}{N} = \frac{1}{8}$$

where N = total number of samples

Step 1: Creating the First Stump

Weighted Gini Index Formula

$$\text{Weighted Gini} = \sum_{j} \frac{N_j}{N} \left(1 - \sum_{k} \left(\frac{\sum_{i \in S_j, y_i = k} w_i}{\sum_{i \in S_j} w_i} \right)^2 \right)$$

Where:

- N_j = number of samples in node j
- N = total number of samples
- S_j = set of samples in node j
- w_i = weight of sample i
- k = class label

Weighted Gini Calculation Example: Patient Weight < 176

Left Node (Weight < 176): Samples 4, 5, 6, 7

- Sample weights: 1/8, 1/8, 1/8, 1/8
- Total weight in node: 4/8 = 0.5
- Heart Disease: Sample 4 (weight = 1/8)
- No Heart Disease: Samples 5, 6, 7 (weight = 3/8)
- Proportion with Heart Disease: (1/8)/(4/8) = 0.25
- Proportion without: (3/8)/(4/8) = 0.75

$$\text{Gini}_{left} = 1 - (0.25^2 + 0.75^2) = 1 - (0.0625 + 0.5625) = 0.375$$

Right Node (Weight ≥ 176): Samples 1, 2, 3, 8

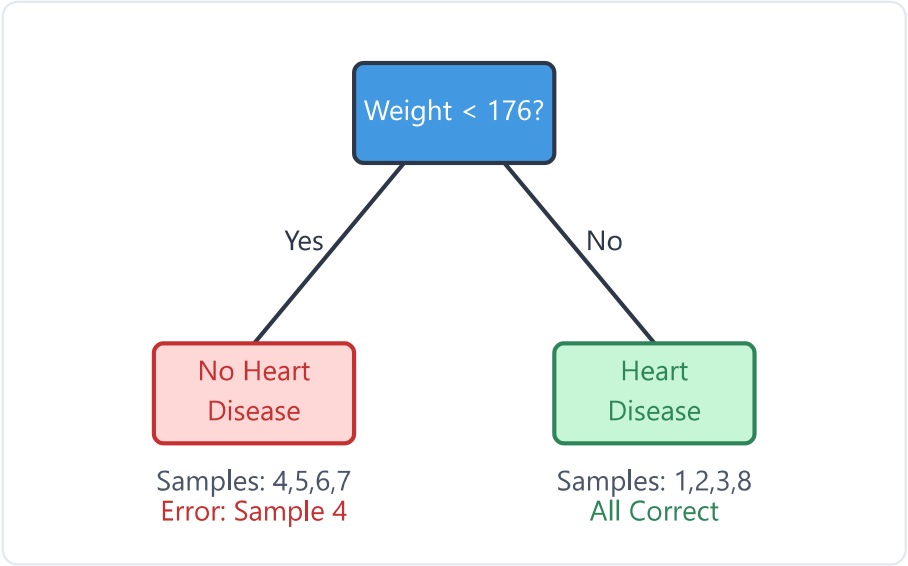
- All have Heart Disease → Pure node
- Proportion with Heart Disease: 1.0

$$\text{Gini}_{right} = 1 - (1.0^2 + 0.0^2) = 0$$

Weighted Average:

$$\text{Weighted Gini} = \frac{4}{8} \times 0.375 + \frac{4}{8} \times 0 = 0.1875$$

Figure 2: First Stump - Patient Weight < 176



Performance: 1 error out of 8 samples (Sample 4 misclassified)

Winner: Patient Weight with threshold 176 has the lowest weighted Gini index (0.1875), making it the best first stump!

Step 2: Calculate "Amount of Say" (Influence)

Amount of Say (Influence) Formula

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

where ε_m = total error = sum of weights of incorrectly classified samples

Amount of Say (Influence) vs Total Error

When error is small (good stump) → large positive α

When error = 0.5 (random) → α = 0

When error is large (bad stump) → large negative α

Calculation for First Stump

Total Error = 1/8 (only sample 4 is misclassified)

$$\alpha_1 = \frac{1}{2}\ln\left(\frac{1-\frac{1}{8}}{\frac{1}{8}}\right) = \frac{1}{2}\ln\left(\frac{7/8}{1/8}\right) = \frac{1}{2}\ln(7) = 0.973$$

Step 3: Update Sample Weights

Weight Update Rule

$$w_i^{(new)} = w_i^{(old)} \cdot e^{-\alpha_m \cdot y_i \cdot h_m(x_i)}$$

For incorrectly classified samples: $y_i \cdot h_m(x_i) = -1$

$$w_i^{(new)} = w_i^{(old)} \cdot e^{+\alpha_m}$$

(weight increases)

For correctly classified samples: $y_i \cdot h_m(x_i) = +1$

$$w_i^{(new)} = w_i^{(old)} \cdot e^{-\alpha_m}$$

(weight decreases)

Weight Update Calculation

For the misclassified sample (Sample 4):

$$w_4^{(new)} = \frac{1}{8} \cdot e^{+0.973} = \frac{1}{8} \cdot 2.64 = 0.33$$

For all correctly classified samples:

$$w_i^{(new)} = \frac{1}{8} \cdot e^{-0.973} = \frac{1}{8} \cdot 0.38 = 0.05$$

Step 4: Normalize the New Weights

Normalization

$$w_i^{(normalized)} = \frac{w_i^{(new)}}{\sum_{j=1}^{N} w_j^{(new)}}$$

This ensures that all weights sum to 1

Sample	Old Weight	New Weight	Normalized Weight
1	1/8 = 0.125	0.05	0.07
2	1/8 = 0.125	0.05	0.07
3	1/8 = 0.125	0.05	0.07

Sample	Old Weight	New Weight	Normalized Weight
4	1/8 = 0.125	**0.33**	0.50
5	1/8 = 0.125	0.05	0.07
6	1/8 = 0.125	0.05	0.07
7	1/8 = 0.125	0.05	0.07
8	1/8 = 0.125	0.05	0.07

Result: **Sample 4 (the misclassified one) now has much higher weight (0.50), making it more important for the next stump to classify correctly!**

Step 5: Build the Second Stump

Two Ways to Use Updated Weights

Method 1: Weighted Gini Index

Calculate weighted Gini indices using the new sample weights directly to find the best split.

$$\text{Weighted Gini} = \sum_{j} \frac{\sum_{i \in S_j} w_i}{\sum_{i=1}^{N} w_i} \left(1 - \sum_{k} \left(\frac{\sum_{i \in S_j, y_i = k} w_i}{\sum_{i \in S_j} w_i} \right)^2 \right)$$

Method 2: Resampling

Create a new dataset by sampling with replacement using weights as probabilities.

Resampling with Cumulative Probability Distribution

Sample	Weight	Cumulative Weight	Probability Range
1	0.07	0.07	[0.00, 0.07)
2	0.07	0.14	[0.07, 0.14)
3	0.07	0.21	[0.14, 0.21)
4	0.50	0.71	[0.21, 0.71)
5	0.07	0.78	[0.71, 0.78)
6	0.07	0.85	[0.78, 0.85)
7	0.07	0.92	[0.85, 0.92)
8	0.07	1.00	[0.92, 1.00]

Resampling Example with Random Numbers

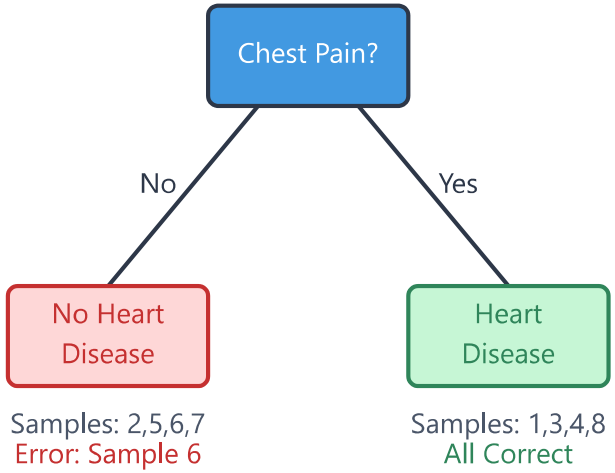
- Random number 0.42 → falls in [0.21, 0.71) → **Select Sample 4**
- Random number 0.05 → falls in [0.00, 0.07) → **Select Sample 1**

- Random number 0.63 → falls in [0.21, 0.71) → **Select Sample 4**
- Random number 0.18 → falls in [0.14, 0.21) → **Select Sample 3**
- Random number 0.55 → falls in [0.21, 0.71) → **Select Sample 4**
- Random number 0.12 → falls in [0.07, 0.14) → **Select Sample 2**
- Random number 0.89 → falls in [0.85, 0.92) → **Select Sample 7**
- Random number 0.95 → falls in [0.92, 1.00] → **Select Sample 8**

Resampled Dataset: [Sample 4, Sample 1, Sample 4, Sample 3, Sample 4, Sample 2, Sample 7, Sample 8]

Result: Sample 4 appears 3 times out of 8, roughly matching its 50% probability!

Figure 3: Second Stump - Chest Pain



Performance: Trained with weighted data focusing on Sample 4

Amount of Say / Influence (α_2): Based on weighted error calculation

Step 6: Making Final Classifications

Final Strong Classifier

$$H(x) = \text{sign}\left(\sum_{m=1}^{M} \alpha_m h_m(x)\right)$$

where:

- α_m = amount of say (influence) for stump m
- $h_m(x)$ = prediction of stump m (+1 or -1)
- M = total number of stumps

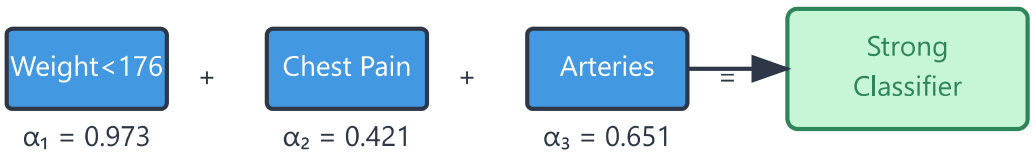
Example Classification

For a new patient, suppose we have 3 stumps with predictions:

Stump	Prediction	Amount of Say / Influence (α)	Weighted Vote
1	+1 (Heart Disease)	0.973	+0.973
2	-1 (No Heart Disease)	0.421	-0.421
3	+1 (Heart Disease)	0.651	+0.651
Final Sum:			**+1.203**

Final Classification: sign(+1.203) = +1 → **Heart Disease**

Figure 4: Ensemble of Three Stumps



Each stump contributes to the final decision with its weighted vote

AdaBoost Algorithm Summary

Complete AdaBoost Algorithm

Input: Training set $\{(x_1,y_1),\ldots,(x_n,y_n)\}$, T = number of rounds

1. **Initialize weights:** $w_i^{(1)} = \frac{1}{N}$ for i = 1,...,N
2. **For t = 1 to T:**
   - Train weak learner h_t using weights $w^{(t)}$
   - Calculate weighted error: $\varepsilon_t = \sum_{i:h_t(x_i)\neq y_i} w_i^{(t)}$
   - Calculate amount of say (influence): $\alpha_t = \frac{1}{2}\ln\left(\frac{1-\varepsilon_t}{\varepsilon_t}\right)$
   - Update weights: $w_i^{(t+1)} = w_i^{(t)} \cdot e^{-\alpha_t y_i h_t(x_i)}$
   - Normalize: $w_i^{(t+1)} = \frac{w_i^{(t+1)}}{\sum_j w_j^{(t+1)}}$

3. **Output:** $H(x) = \text{sign}\left(\sum_{t=1}^{T} \alpha_t h_t(x)\right)$

Key Equations

Essential AdaBoost Formulas

1. Initial weights:

$$w_i^{(1)} = \frac{1}{N}$$

2. Weighted error:

$$\varepsilon_t = \sum_{i=1}^{N} w_i^{(t)} \mathbf{1}[h_t(x_i) \neq y_i]$$

3. Amount of say (Influence):

$$\alpha_t = \frac{1}{2} \ln\left(\frac{1-\varepsilon_t}{\varepsilon_t}\right)$$

4. Weight update:

$$w_i^{(t+1)} = w_i^{(t)} \exp(-\alpha_t y_i h_t(x_i))$$

5. Normalization:

$$w_i^{(t+1)} \leftarrow \frac{w_i^{(t+1)}}{\sum_{j=1}^{N} w_j^{(t+1)}}$$

6. Final classifier:

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

Why Does AdaBoost Work?

Focus on Mistakes

Each new stump focuses on the samples that previous stumps got wrong, gradually improving overall performance.

Weighted Voting

Better stumps (lower error) get more influence in the final decision, ensuring quality predictions dominate.

Margin Maximization

AdaBoost maximizes the "margin" - the confidence of predictions, leading to better generalization.

Margin Definition

$$\text{margin}(x, y) = y \cdot \sum_{t=1}^{T} \alpha_t h_t(x)$$

Larger margins indicate more confident, reliable predictions

Key Insight: **Even when training error reaches zero, AdaBoost can continue running to increase margins and improve generalization to new data.**

Practical Considerations & Tips

When to Stop

- **Fixed number of rounds:** Set T in advance
- **Validation-based:** Stop when validation error stops improving
- **Perfect training:** Stop when training error reaches zero

Handling Edge Cases

When error = 0 or error = 1:

$$\alpha_t = \frac{1}{2}\ln\left(\frac{1-\varepsilon_t+\delta}{\varepsilon_t+\delta}\right)$$

Add small $\delta > 0$ to prevent division by zero

Requirements: **Weak learners should perform better than random chance (error < 0.5). If error > 0.5, the algorithm can still work because negative α values flip the prediction!**

Applications & Advantages

Common Applications

- **Face Detection:** Viola-Jones algorithm
- **Text Classification:** Spam filtering
- **Medical Diagnosis:** Binary classification problems
- **Object Recognition:** Computer vision tasks

Advantages

- **Simple:** Easy to understand and implement
- **Flexible:** Works with any weak learner
- **No hyperparameters:** Minimal tuning required
- **Good performance:** Often very effective

Limitations

- **Sensitive to noise:** Outliers get high weights
- **Can overfit:** Especially with noisy data
- **Sequential:** Cannot parallelize training

- **Requires weak learners better than random:** Base classifiers must be better than random

Final Review: The Three Ideas

1. Combine Weak Learners

AdaBoost combines many **weak learners** (usually stumps) to make classifications.

```
Stump 1  +  Stump 2  +  ...  +  Stump T
```

2. Weighted Voting

Some stumps get **more say** (influence) in the classification than others.

$$\alpha = \frac{1}{2}\ln\left(\frac{1-\varepsilon}{\varepsilon}\right)$$

3. Learn from Mistakes

Each stump focuses on the **previous stumps' mistakes**.

$$w_i \leftarrow w_i \cdot e^{-\alpha y_i h(x_i)}$$

Figure 6: AdaBoost Process Summary

Sequential learning process where each stump learns from previous mistakes

AdaBoost = Adaptive Boosting

"Boost" the performance of weak learners by "adapting" to their mistakes!