■ **Course:** CS109A: Introduction to Data Science

■ **Lecture:** Lecture 24: AdaBoost

■ **Instructors:** Pavlos Protopapas, Kevin Rader, Chris Gumb

■ **Objective:** Understand AdaBoost as a boosting algorithm for classification, learn how it differs from Gradient Boosting, and master the weight update mechanism

Contents

1 Introduction to AdaBoost

Key Summary

AdaBoost (Adaptive Boosting) is a boosting algorithm designed for classification. Its key innovation:

Instead of fitting residuals, AdaBoost reweights the training data.

After each weak learner:

- Misclassified samples get **higher weights**
- Correctly classified samples get **lower weights**
- The next weak learner focuses on the "hard" examples

This is like studying for an exam by focusing on the problems you got wrong!

1.1 Historical Context

AdaBoost was introduced by Yoav Freund and Robert Schapire in 1996 and won the prestigious Gödel Prize in 2003. It was one of the first practical boosting algorithms and demonstrated that weak learners could be combined into a strong learner.

1.2 The "Error Notebook" Analogy

Example: Studying with an Error Notebook

Imagine you're preparing for a difficult exam:

1. **Practice test 1**: You get some questions right, some wrong
2. **Star the wrong ones**: Mark the problems you missed with a star ($\star$)
3. **Focus your study**: Spend more time on starred problems
4. **Practice test 2**: You might get the starred ones right, but miss some new ones
5. **Update stars**: Star the new mistakes, remove stars from ones you now understand
6. **Repeat**: Keep focusing on your current weaknesses
7. **Final exam**: Combine everything you learned from all practice tests

This is exactly how AdaBoost works! Each weak learner is like a practice test, and the sample weights are like your star markings.

2 Key Concepts

2.1 Weak Learners: Stumps

Definition: Decision Stump

A **decision stump** is a decision tree with only **one split**:

- One root node (asks one question)
- Two leaf nodes (makes two predictions)

A stump is the simplest possible decision tree. It's a "weak learner"—better than random guessing, but not by much.

Why use such a simple model?

- Each stump captures **one simple pattern**
- Many stumps combined can capture **complex patterns**
- Simple models are **less prone to overfitting** individually
- The boosting process handles complexity through **aggregation**

2.2 Label Encoding: -1 and $+1$

AdaBoost uses labels $y \in \{-1, +1\}$ instead of $\{0, 1\}$. This makes the math elegant:

Definition: The Sign Trick

When $y, \hat{y} \in \{-1, +1\}$:

- If **correct**: $y \cdot \hat{y} = +1$ (both same sign)
- If **wrong**: $y \cdot \hat{y} = -1$ (opposite signs)

Examples:

- $y = +1$, $\hat{y} = +1 \Rightarrow y \cdot \hat{y} = +1$ (correct)
- $y = -1$, $\hat{y} = -1 \Rightarrow y \cdot \hat{y} = +1$ (correct)
- $y = +1$, $\hat{y} = -1 \Rightarrow y \cdot \hat{y} = -1$ (wrong)
- $y = -1$, $\hat{y} = +1 \Rightarrow y \cdot \hat{y} = -1$ (wrong)

This property is crucial for the weight update formula!

2.3 Sample Weights

Unlike gradient boosting (which modifies the target), AdaBoost maintains a **weight for each sample**:

- Initially, all samples have equal weight: $w_i = \frac{1}{N}$
- After each iteration, weights are adjusted:
  - Misclassified samples: weight **increases**
  - Correctly classified samples: weight **decreases**
- Weights always sum to 1 (they form a distribution)

3 The AdaBoost Algorithm

Definition: AdaBoost Algorithm

Input: Training data (x_i, y_i) with $y_i \in \{-1, +1\}$, number of iterations M
Initialize: $w_i^{(0)} = \frac{1}{N}$ for all samples
For $m = 1, 2, \ldots, M$**:**

1. **Fit weak learner** h_m using weights $w^{(m-1)}$
2. **Compute weighted error**:

$$\epsilon_m = \sum_{i:h_m(x_i)\neq y_i} w_i^{(m-1)}$$

(Sum of weights of misclassified samples)

3. **Compute learner weight** (how much "say" this learner gets):

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

4. **Update sample weights**:

$$w_i^{(m)} = w_i^{(m-1)} \cdot \exp(-\alpha_m \cdot y_i \cdot h_m(x_i))$$

Then normalize so weights sum to 1.

5. **Add to ensemble**:

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

Final prediction: $\hat{y} = \text{sign}(H_M(x))$

3.1 Understanding the Learner Weight α

The formula $\alpha_m = \frac{1}{2}\ln\left(\frac{1-\epsilon_m}{\epsilon_m}\right)$ determines how much influence each weak learner has:

Table 1: *Learner Weight vs. Error Rate*

Error ϵ	**Meaning**	**Weight** α
0.0	Perfect classifier	$+\infty$
0.1	Very good	Large positive
0.3	Good	Moderate positive
0.5	Random guessing	0 (ignored!)
0.7	Worse than random	Negative (flips!)
1.0	Perfectly wrong	$-\infty$

Key Information

Key Insights:

- Better classifiers ($\epsilon < 0.5$) get positive weight—their votes count!
- Random classifiers ($\epsilon = 0.5$) get zero weight—ignored

- Worse-than-random classifiers ($\epsilon > 0.5$) get negative weight—their predictions are **flipped**!

3.2 Understanding the Weight Update

The weight update formula is:

$$w_i^{(new)} \propto w_i^{(old)} \cdot \exp(-\alpha \cdot y_i \cdot h(x_i))$$

Let's understand this:

- **If correctly classified**: $y_i \cdot h(x_i) = +1$
  - Exponent: $-\alpha \cdot (+1) = -\alpha < 0$
  - Factor: $e^{-\alpha} < 1$
  - Weight **decreases**
- **If misclassified**: $y_i \cdot h(x_i) = -1$
  - Exponent: $-\alpha \cdot (-1) = +\alpha > 0$
  - Factor: $e^{+\alpha} > 1$
  - Weight **increases**

Example: Weight Update Example

Suppose $\alpha = 0.5$ (a moderately good classifier):

- **Correctly classified**: Weight multiplied by $e^{-0.5} \approx 0.61$ (decreases by 39%)
- **Misclassified**: Weight multiplied by $e^{0.5} \approx 1.65$ (increases by 65%)

If $\alpha = 2.0$ (a very good classifier):

- **Correctly classified**: Weight multiplied by $e^{-2} \approx 0.14$ (decreases by 86%)
- **Misclassified**: Weight multiplied by $e^{2} \approx 7.4$ (increases by 640%!)

The better the classifier, the more dramatically we adjust weights!

4 How Weak Learners Use Weights

4.1 Option 1: Weighted Loss Function

When training the decision stump, instead of counting misclassifications, we compute the **weighted error**:

$$\text{Weighted Error} = \sum_{i:h(x_i)\neq y_i} w_i$$

For decision trees using Gini impurity:

$$\text{Weighted Gini} = \sum_i w_i \cdot \mathbf{1}[\text{sample } i \text{ in this node}] \cdot \text{impurity contribution}$$

This makes misclassifying high-weight samples more "costly," so the stump avoids those mistakes.

4.2 Option 2: Resampling

Alternatively, we can **resample** the training data according to the weights:

1. Generate a new training set by sampling with replacement
2. Probability of selecting sample i is proportional to w_i
3. Train a regular (unweighted) stump on this resampled data

This approach:

- Implicitly gives high-weight samples more influence
- Works with any base learner (doesn't need to support weights)
- Introduces additional randomness

5 Visual Example: 2D Classification

Example: AdaBoost in Action

Consider classifying two groups: orange circles and blue triangles.

Round 1:

- All 10 samples have weight $w = 0.1$
- First stump: "Is $x_1 > 4.6$?"
- Correctly classifies 7, misclassifies 3 (all orange circles)
- $\epsilon_1 = 0.1 + 0.1 + 0.1 = 0.3$
- $\alpha_1 = \frac{1}{2} \ln\left(\frac{0.7}{0.3}\right) \approx 0.42$

After Round 1 weight update:

- Correct samples: weights $\to 0.1 \times e^{-0.42} \approx 0.066$
- Misclassified samples: weights $\to 0.1 \times e^{0.42} \approx 0.152$
- After normalization: misclassified samples have ∼2.3x the weight

Round 2:

- Second stump must pay more attention to those 3 orange circles
- Chooses "Is $x_2 > 8$?" to separate them
- This might misclassify some previously correct blue triangles
- Process continues...

Final result:

- 3 simple axis-aligned splits
- Combined: complex decision boundary
- Can separate the groups better than any single stump!

6 AdaBoost vs. Gradient Boosting

Both are boosting methods, but they differ in key ways:

Table 2: *Comparison: AdaBoost vs. Gradient Boosting*

Aspect	AdaBoost	Gradient Boosting
Loss function	Exponential loss	Any differentiable loss
How it learns	Reweights samples	Fits to residuals/gradients
Learner weight α	Computed from formula	User-specified (learning rate)
Primary use	Classification	Classification and Regression
Sensitivity to outliers	High	Moderate
Overfitting tendency	Moderate	Can be controlled

6.1 The Connection: AdaBoost IS Gradient Boosting

Important: AdaBoost

AdaBoost can be viewed as a special case of gradient boosting where:

- Loss function: $L(y, f) = \exp(-y \cdot f)$ (exponential loss)
- This loss heavily penalizes confident wrong predictions

The weight update in AdaBoost is actually computing the gradient of exponential loss!

6.2 Exponential Loss

Definition: Exponential Loss

For labels $y \in \{-1, +1\}$ and prediction $f(x)$:

$$L(y, f) = \exp(-y \cdot f(x))$$

Properties:

- Correct confident prediction ($y \cdot f \gg 0$): Loss ≈ 0
- Wrong confident prediction ($y \cdot f \ll 0$): Loss $\to \infty$
- Serves as an upper bound on 0-1 classification error

7 Overfitting and Hyperparameters

7.1 AdaBoost Can Overfit!

Unlike some claims, boosting methods (including AdaBoost) **can overfit**:

Warning

Why AdaBoost Overfits:

- AdaBoost **obsesses** over misclassified samples
- Some "hard" samples might be outliers or noise
- With enough iterations, it memorizes these noise points
- Result: Perfect training accuracy, poor test performance

Solution: Use early stopping based on validation error!

7.2 Key Hyperparameters

Table 3: *AdaBoost Hyperparameters in sklearn*

Parameter	Default	Description
`n_estimators`	50	Number of weak learners
`learning_rate`	1.0	Shrinkage factor for α
`estimator`	DecisionTree(depth=1)	Base weak learner

Tuning advice:

- `n_estimators`: More = more complex. Use validation to find optimal.
- `learning_rate`: Lower values require more estimators but often generalize better.
- `estimator`: Stumps (depth 1) are standard; depth 2-3 can help but risk overfitting.

8 AdaBoost in Python

```python
from sklearn.ensemble import AdaBoostClassifier
from sklearn.tree import DecisionTreeClassifier
from sklearn.model_selection import train_test_split
import numpy as np
import matplotlib.pyplot as plt

# Prepare data
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2)

# Create AdaBoost classifier
# Note: sklearn uses SAMME algorithm (similar to original AdaBoost)
ada = AdaBoostClassifier(
    estimator=DecisionTreeClassifier(max_depth=1),  # Stump
    n_estimators=50,
    learning_rate=1.0,
    algorithm='SAMME',
    random_state=42
)

# Train
ada.fit(X_train, y_train)

# Evaluate
print(f"Train Accuracy: {ada.score(X_train, y_train):.4f}")
print(f"Test Accuracy: {ada.score(X_test, y_test):.4f}")
```

8.1 Visualizing Training Progress

```python
# Plot training and test error vs. number of estimators
train_errors = []
test_errors = []

for n in range(1, 101):
    ada = AdaBoostClassifier(
        estimator=DecisionTreeClassifier(max_depth=1),
        n_estimators=n,
        random_state=42
    )
    ada.fit(X_train, y_train)
    train_errors.append(1 - ada.score(X_train, y_train))
    test_errors.append(1 - ada.score(X_test, y_test))

plt.figure(figsize=(10, 6))
plt.plot(range(1, 101), train_errors, label='Training Error')
plt.plot(range(1, 101), test_errors, label='Test Error')
plt.xlabel('Number of Estimators')
plt.ylabel('Error Rate')
plt.title('AdaBoost Learning Curve')
plt.legend()
plt.show()

# Find optimal number
best_n = np.argmin(test_errors) + 1
print(f"Optimal n_estimators: {best_n}")
```

8.2 Getting Feature Importance

```python
# AdaBoost provides feature importance
importances = ada.feature_importances_

# Display
import pandas as pd
pd.DataFrame({
    'Feature': feature_names,
    'Importance': importances
}).sort_values('Importance', ascending=False)
```

9 When to Use AdaBoost

9.1 Advantages

- **Simple to implement**: Algorithm is straightforward
- **No hyperparameter for α**: Learner weights computed automatically
- **Works with any weak learner**: Not limited to trees
- **Feature importance**: Built-in interpretation
- **Historical importance**: Foundation of modern boosting

9.2 Disadvantages

- **Sensitive to outliers**: Keeps increasing weights on hard-to-classify points
- **Sensitive to noise**: Treats noisy samples as "important"
- **Can overfit**: Especially with many iterations
- **Limited to classification**: Original form; Gradient Boosting is more flexible

9.3 Modern Alternatives

In practice, you'll often use:

- **XGBoost**: Optimized gradient boosting with regularization
- **LightGBM**: Fast gradient boosting for large datasets
- **CatBoost**: Handles categorical features well

These are all variants of Gradient Boosting rather than AdaBoost, but understanding AdaBoost helps you understand the foundations.

10 Key Takeaways

Key Summary

AdaBoost Core Ideas:

- Combines weak learners (stumps) into a strong classifier
- Uses **sample weights** instead of residuals
- Misclassified samples get higher weights
- Better classifiers get more voting power (α)

The Algorithm:

1. Initialize all sample weights equally: $w_i = 1/N$
2. For each iteration:
   - Train stump on weighted data
   - Compute weighted error ϵ
   - Compute learner weight $\alpha = \frac{1}{2} \ln \frac{1-\epsilon}{\epsilon}$
   - Update sample weights: $w_i \leftarrow w_i \cdot \exp(-\alpha y_i h(x_i))$
3. Final prediction: weighted vote of all stumps

Key Formulas:

- Learner weight: $\alpha = \frac{1}{2} \ln \left(\frac{1-\epsilon}{\epsilon}\right)$
- Sample weight update: $w_{new} \propto w_{old} \cdot \exp(-\alpha \cdot y \cdot \hat{y})$
- Final prediction: $\text{sign}\left(\sum_m \alpha_m h_m(x)\right)$

Practical Tips:

- Use validation data to avoid overfitting
- Monitor both training and test error
- Consider XGBoost/LightGBM for production use

11 Practice Questions

1. **Conceptual**: Explain why AdaBoost uses labels $\{-1, +1\}$ instead of $\{0, 1\}$. How does this simplify the weight update formula?
2. **Calculation**: A weak learner has weighted error $\epsilon = 0.2$. Calculate α and explain what this value means for the learner's influence.
3. **Weight Update**: If a sample has current weight $w = 0.1$ and $\alpha = 0.5$, what is the new weight if:
   - The sample is correctly classified?
   - The sample is misclassified?
4. **Edge Cases**: What happens in AdaBoost if a weak learner achieves:
   - Perfect accuracy ($\epsilon = 0$)?
   - Random guessing ($\epsilon = 0.5$)?
   - Worse than random ($\epsilon = 0.7$)?
5. **Comparison**: Explain the key difference between how AdaBoost and Gradient Boosting "learn from mistakes."
6. **Overfitting**: Why can AdaBoost overfit even though each weak learner is simple? How would you detect and prevent this?
7. **Code**: Modify the sklearn AdaBoost example to:
   - Use depth-2 trees instead of stumps
   - Implement early stopping based on validation error
   - Compare performance to the default configuration