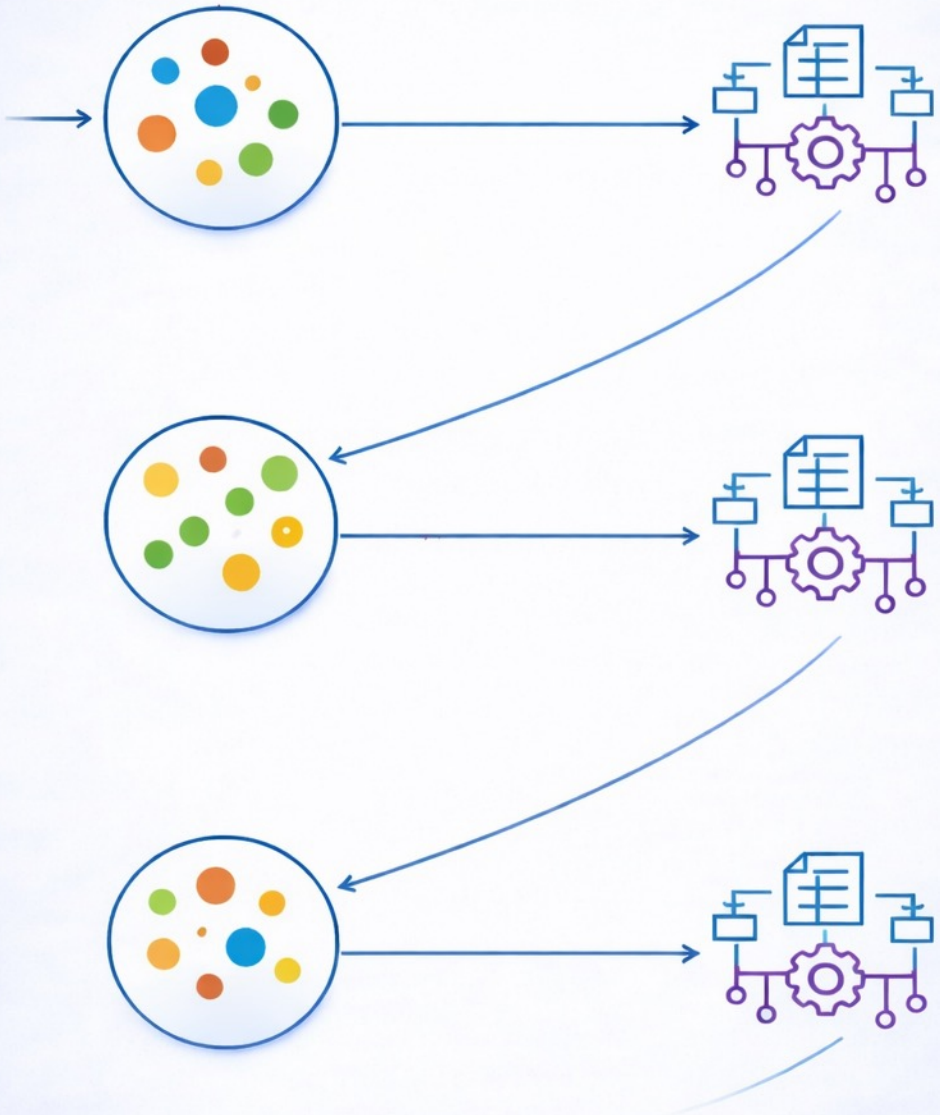
Georgia Tech

Boosting Algorithms



Mohsen Moghaddam, Ph.D.

Gary C. Butler Family Associate Professor
H. Milton Stewart School of Industrial and Systems Engineering
George W. Woodruff School of Mechanical Engineering
Georgia Institute of Technology

SAIL
Symbiotic & Augmented Intelligence Laboratory

Spring 2026

Learning Outcomes

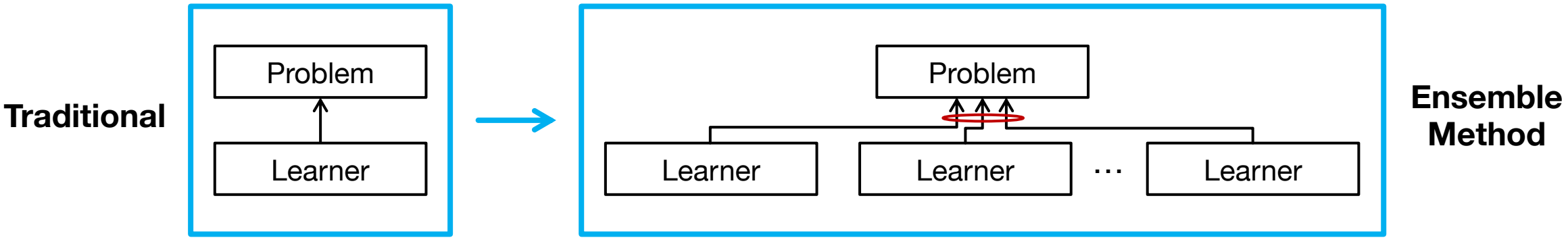
- Explain the motivation for boosting and how combining weak learners leads to improved predictive performance
- Compare boosting and bagging, focusing on their different impacts on bias and variance
- Describe the AdaBoost algorithm, including how training examples and weak learners are reweighted over iterations
- Interpret boosting from a geometric perspective by explaining how decision boundaries evolve as more learners are added
- Explain boosting as an optimization process that minimizes a suitable loss function
- Connect AdaBoost to gradient boosting and modern boosting methods used in practice

Motivation & Big Picture

Rationale: Combination of Methods

"Can a set of weak learners create a single strong learner?" — Kearns & Valiant, 1988–89

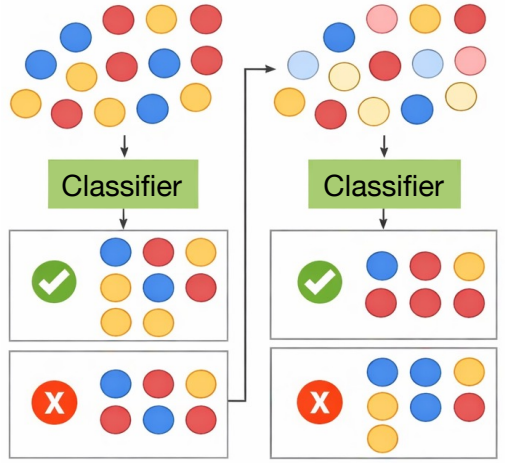
- In general machine learning tasks, there is no algorithm that is always the most accurate
- Different learners use different algorithms, hyperparameters, representations, training sets, and subproblems
- **Weak learner:** Any classifier that performs slightly better than random guessing
- **Key idea:** Instead of searching for one "best" model, construct a strong model by sequentially correcting mistakes made by simpler models



Two Main Ensemble Approaches

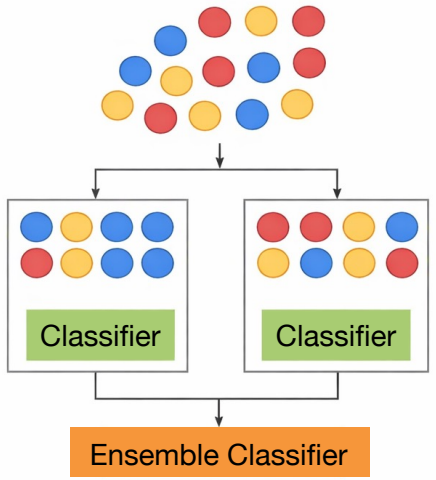
Boosting

- Run weak learners sequentially on **reweighted data**
- Combine weak learners **linearly** (weighted)
- Requires knowledge of each learner's performance
- Primarily **reduces bias** by correcting errors



Bagging

- Run weak learners on bootstrap samples of the training set (sampling with replacement to create multiple datasets)
- Average weak leaners
- Primarily **reduces variance** by averaging models



What Is Boosting?

General methods of converting weak learner into a stronger one

- Each learner depends on the previous ones and focuses on their errors, making the process sequential
- Incorrectly predicted data from previous rounds are given higher weights when training the next learner

How to adjust weights for data?

How to combine learners?

AdaBoost answers both using probability weights and optimization of a loss function

Core Idea of Boosting

Each round, the algorithm **“boosts”** the learner by:

- Increasing the importance of previously misclassified examples
- Combining weak learners with weights proportional to their accuracy
- Over multiple iterations, the decision boundary evolves from simple vertical/horizontal splits to a nonlinear boundary that fits complex data

Why does focusing on misclassified points might reduce bias rather than variance?

Weak Learners

Example: Credit Card Fraud Detection

- Fraud cases are rare (≈ 0.1%) and patterns are subtle
- A single model (e.g., logistic regression) misses complex interactions
- **Features:** amount, time, merchant type, location, device, etc.

Boosting algorithms (e.g., AdaBoost)

- Use many shallow decision trees as weak learners
- Each new tree focuses on mistakes from previous ones
- Combine all trees → strong predictive model
- Detect >95% of frauds with low false-alarm rate
- Deployed in real time for transaction scoring
- Interpretable features (e.g., foreign + high amount + odd hour)

Would bagging help as much in this scenario?

Decision Stumps

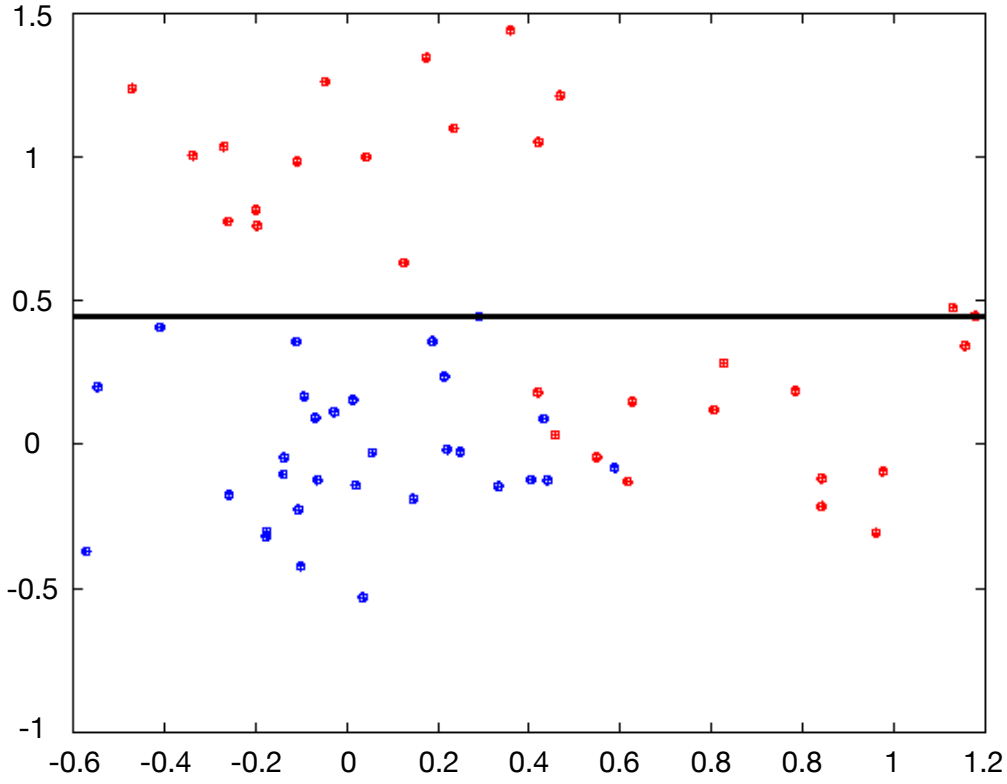
Each **decision stump** asks a single yes/no question about one feature—acting as a "one-question exam," creating a simple axis-aligned split

Given $y \in \{\pm 1\}$, the decision stump for the j^{th} dimension is

$$h(x; \theta_j) = \text{sign}(w_j x_j + b_j)$$

- w_j controls direction/sign (flip labels)
- b_j sets the threshold location

How can combining many such simple splits yield a nonlinear boundary?



Boosting Problem Setup

Objective: Build a classifier that outperforms every individual h_t

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

Weighted vote of all decision stumps

Given a set of base classifiers $\{h_1, h_2, \dots\}$, $h_i: X \rightarrow \{1, -1\}$, training data:

$$(x^i, y^i),\ i = 1, \dots, m, \quad x^i \in X \text{ and } y^i \in \{1, -1\}$$

Construct:

- A sequence of distributions (weights on data sum up to 1) $D(i)$, $i = 1, \dots m$
- A sequence α_t of nonnegative weights of base classifiers

AdaBoost Algorithm

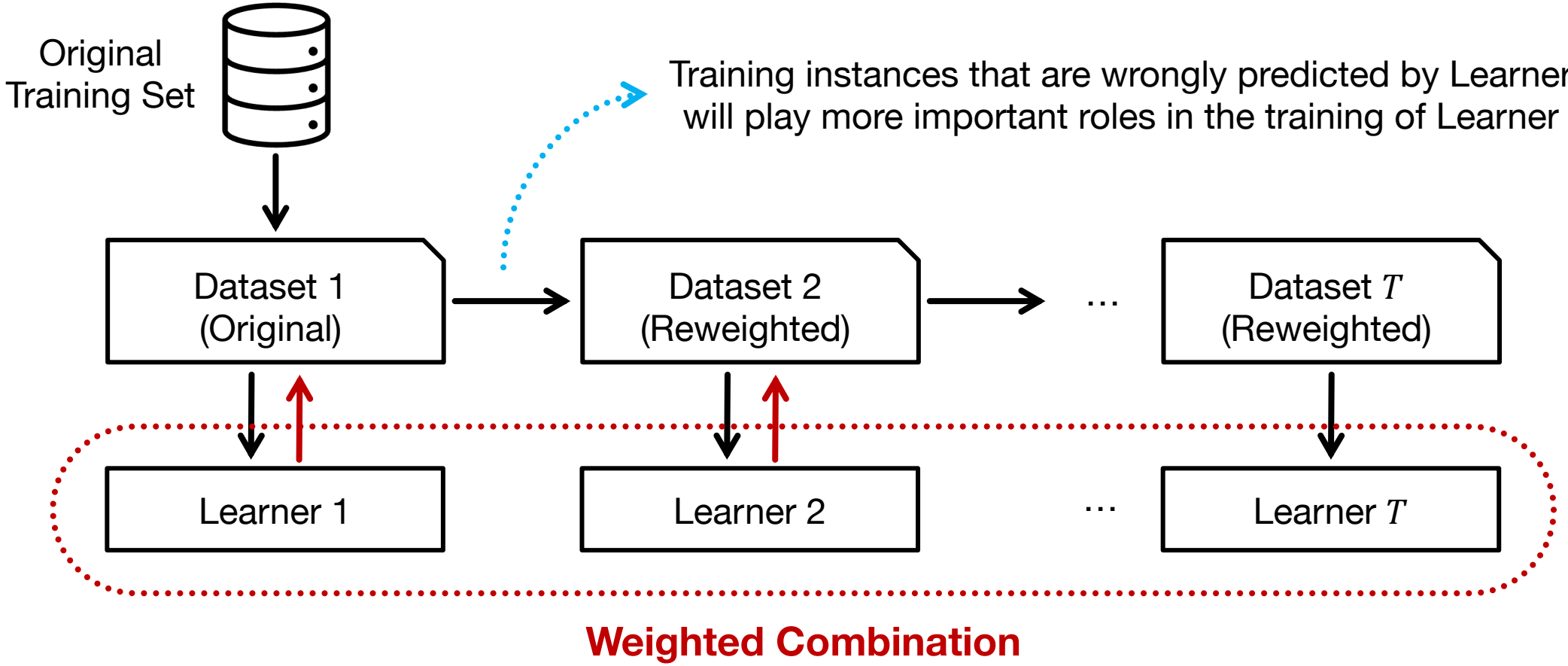
AdaBoost: Adaptive Boosting Algorithm

- Subsequent weak learners are tweaked to focus more on instances misclassified by previous classifiers
- Data distribution (weights) is updated adaptively after each round
- Many variants exist—here we present a basic version (winner of 2003 Gödel Prize)
- **Adaptive:** training example weights change each round based on errors

Why might over-focusing on hard points be risky?

AdaBoost Algorithm Logic

Train → Measure Error → Reweight → Repeat



AdaBoost Algorithm Details

Input: data $\{(x^i, y^i)\}_{i=1}^m$, $y^i \in \{-1, +1\}$, number of rounds T

Initialize: $D_1(i) = 1/m$, for all $i = 1, \ldots, m$

for $t = 1, 2, \ldots, T$

Train weak classifiers $h_t: X \longrightarrow \{-1, +1\}$ using distribution D_t

Compute weighted error $\epsilon_t = \sum_{i=1}^m D_t(i) \mathbb{I}\{y^i \neq h_t(x^i)\}$

Compute classifier weight $\alpha_t = \frac{1}{2} \ln\left(\frac{1-\epsilon_t}{\epsilon_t}\right)$

Update data weights

$$D_{t+1}(i) = \frac{D_t(i)}{Z_t} e^{-\alpha_t y^i h_t(x^i)} = \frac{D_t(i)}{Z_t} \times \begin{cases} e^{-\alpha_t} & \text{if } y^i = h_t(x^i) \\ e^{\alpha_t} & \text{otherwise} \end{cases}$$

where normalization is constant $Z_t = \sum_{i=1}^m D_t(i) e^{-\alpha_t y^i h_t(x^i)}$

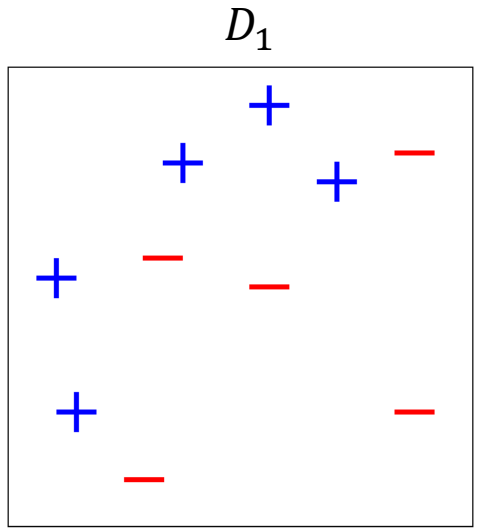
Output: Final classifier $H(x) = \text{sign}(\sum_{t=1}^T \alpha_t h_t(x))$

Geometric & Visual Understanding

Toy Example: Visualizing Boosting

Weak classifier (rule of thumb): vertical or horizontal half-planes (decision stump)

- Let $y \in \{\pm 1\}$
- Initial weights are uniform—no point is special yet
- We will use decision stumps—vertical or horizontal splits—as weak learners
- Weak learners are very simple and high-bias
- Each round will update weights and add a new stump, gradually refining the decision boundary



What kind of patterns can a single decision stump represent?

Boosting Round 1

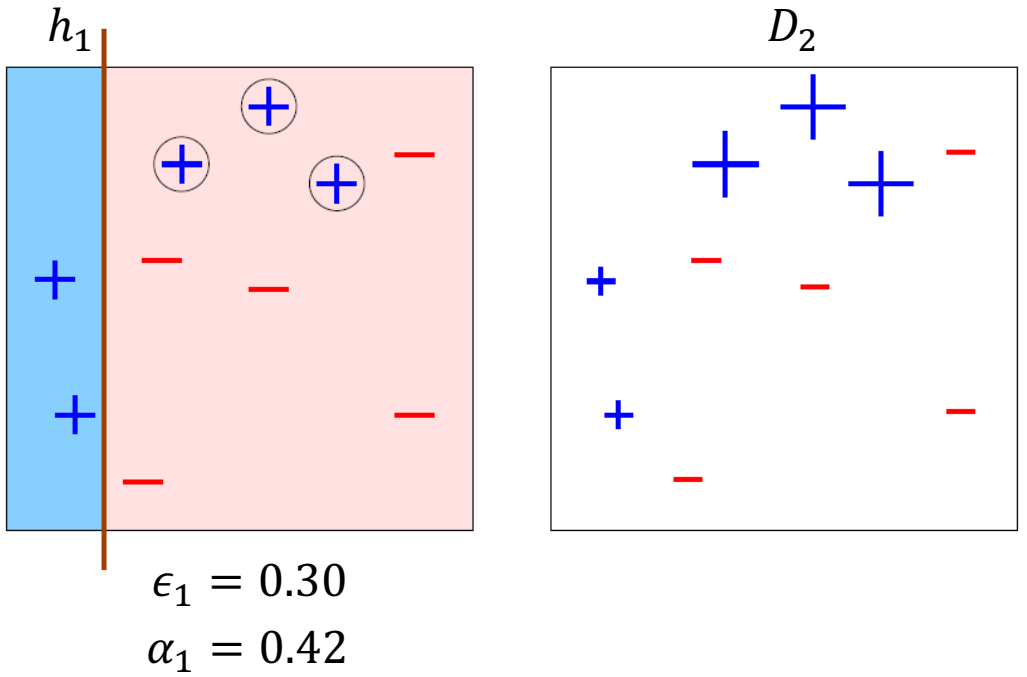
Choose a decision stump (weak classifier)—some data points obtain higher weights because they are classified incorrectly (circle size ∝ example weight)

$$m = 10 \qquad D_1(i) = \frac{1}{10} \qquad \epsilon_t = 3 \times \frac{1}{10} = 0.3$$

$$\alpha_t = \frac{1}{2} \ln\left(\frac{1-\epsilon_t}{\epsilon_t}\right) = 0.5 \times \ln\left(\frac{0.7}{0.3}\right) = 0.42$$

$$D_2(i) = \frac{0.1}{Z_1} \times \begin{cases} e^{-0.42} & \text{if "correct"} \\ e^{0.42} & \text{otherwise} \end{cases}$$

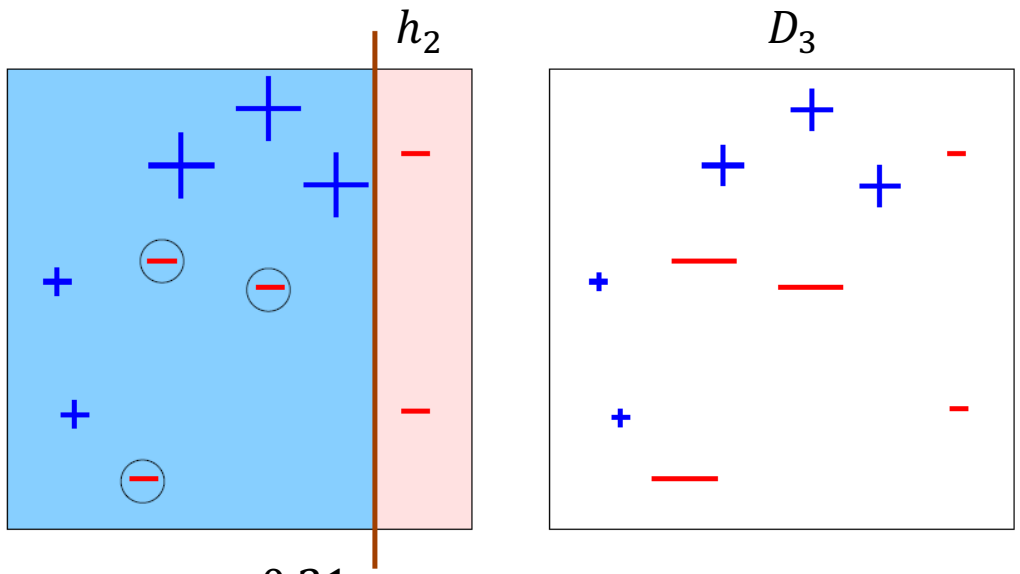
$$Z_1 = 3 \times (0.1 \times e^{.42}) + 7(0.1 \times e^{-.42}) = \cdots$$



Boosting Round 2

Choose a new decision stump—for incorrectly classified examples, weight increased (**reweight**)

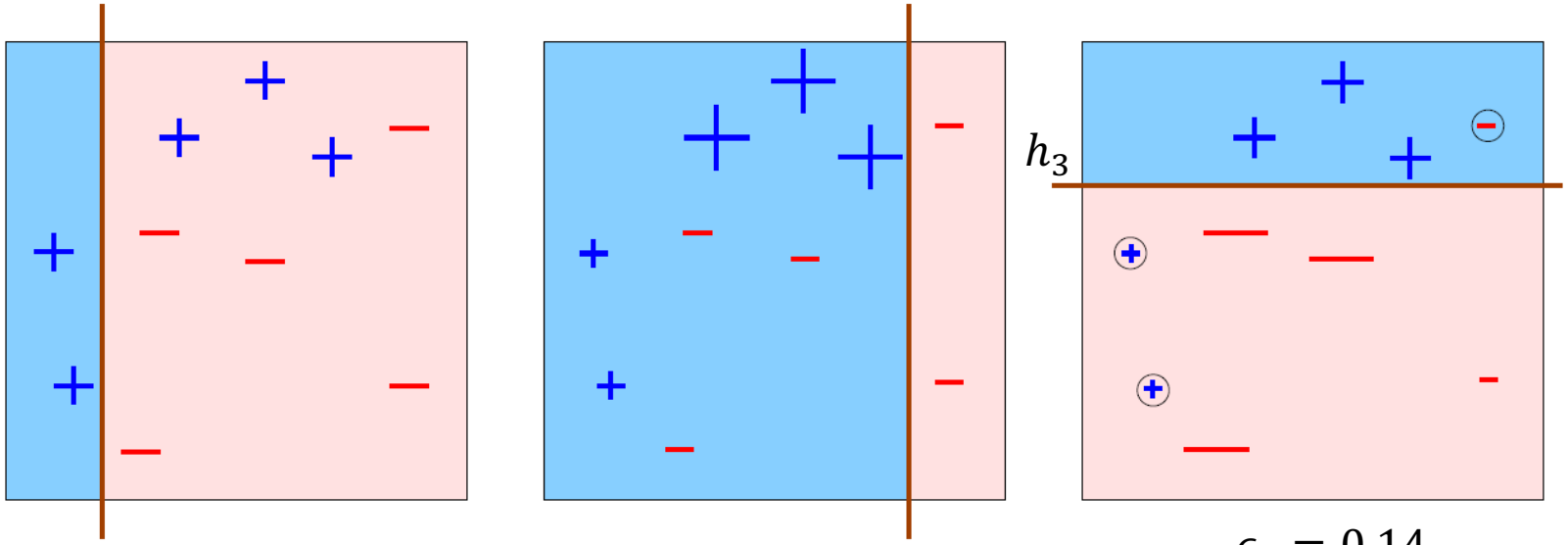
- Now the data distribution is no longer uniform
- The new stump adapts to previously misclassified points
- Weighted error (ϵ) drops, so influence (α) increases



Misclassified points receive higher influence

Boosting Round 3

Additive modeling—each learner adds a piece to the boundary and cleans up the remaining errors



$\epsilon_3 = 0.14$

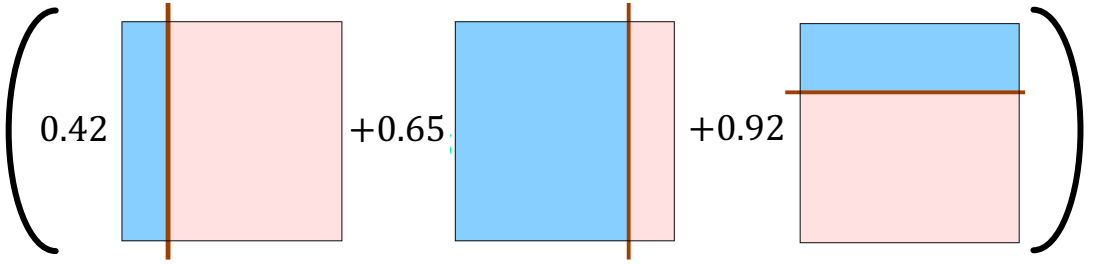
$\alpha_3 = 0.92$

What would happen if we kept adding more stumps?

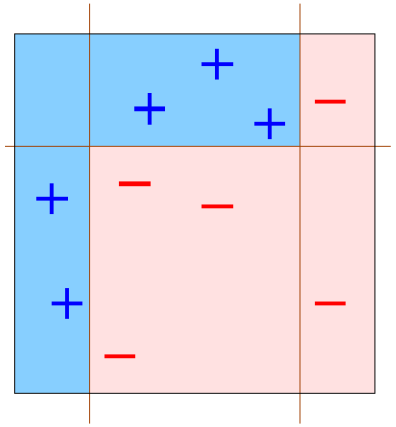
Final Boosted Classifier

The final classifier is the **weighted combination of weak classifiers**

$$H_{\text{final}} = \text{sign}\left(0.42 \; + 0.65 \; + 0.92 \right)$$



$$=$$

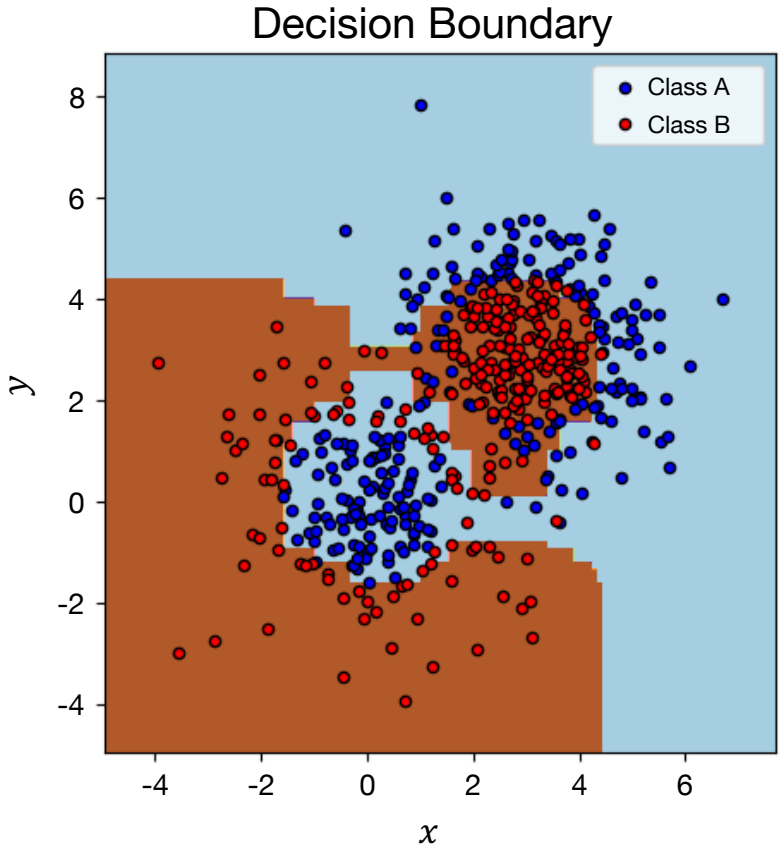


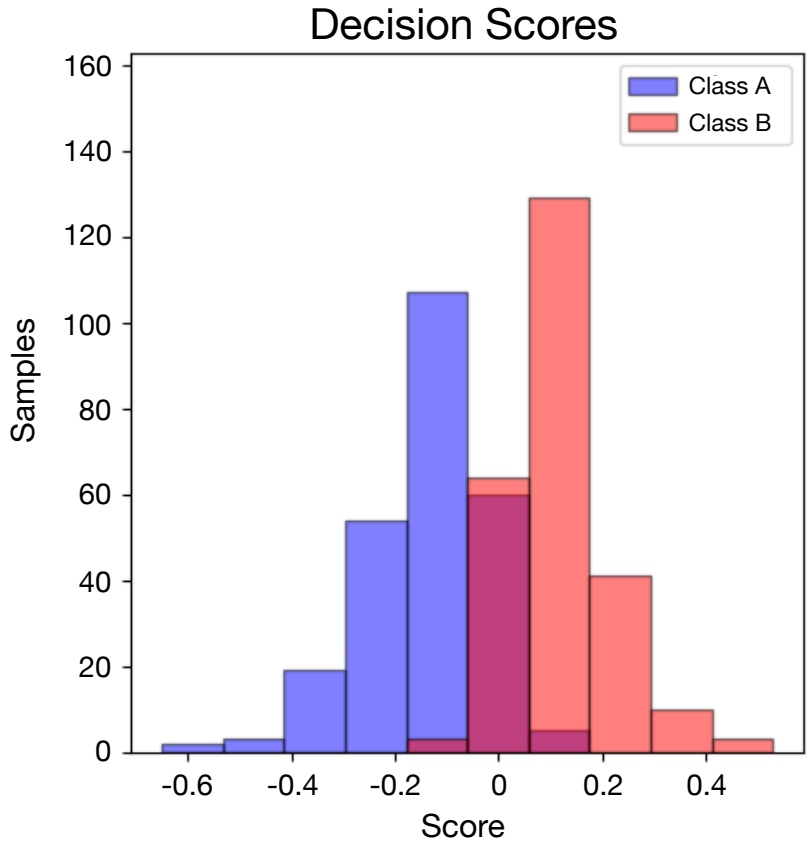
Multiple simple splits combine into a **nonlinear boundary**

Learners with lower error receive larger weights (α_t)—weighted vote

Demo: Two-Class AdaBoost

Boosting produces nonlinear boundaries using simple learners

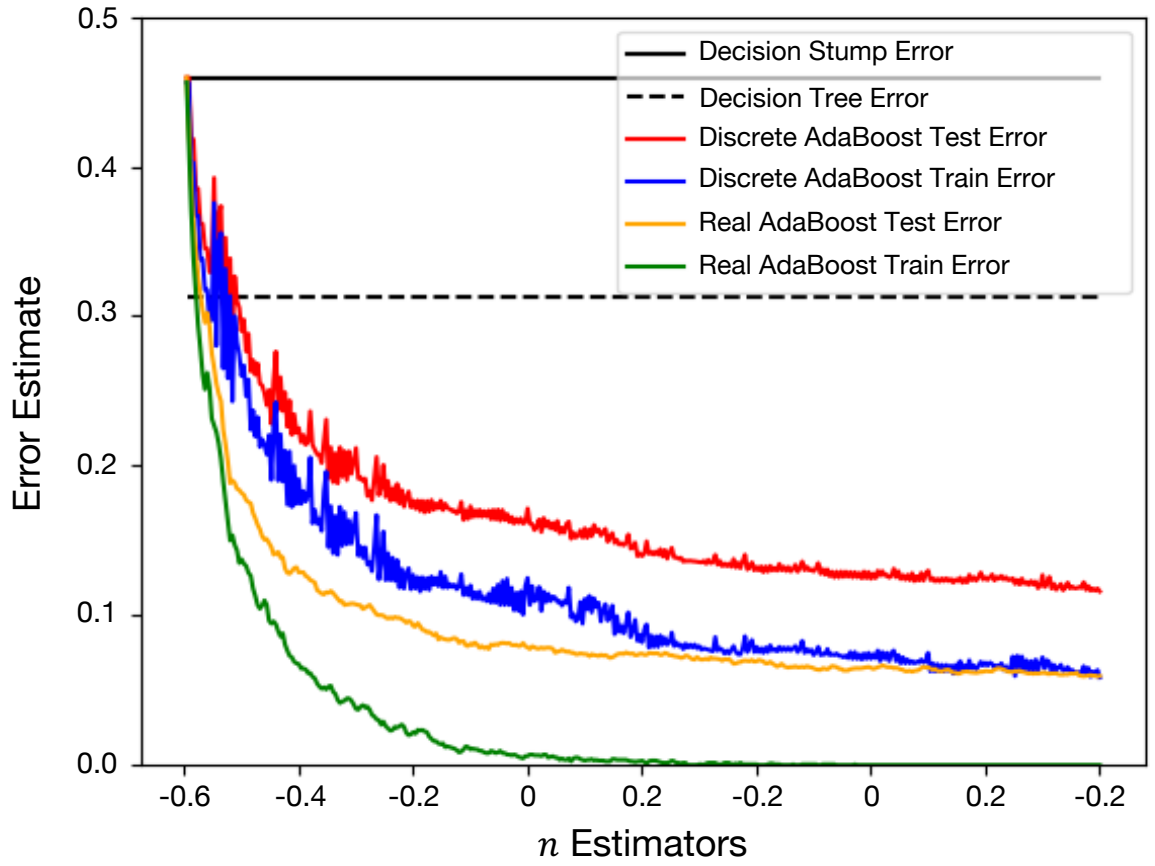




https://scikit-learn.org/stable/auto_examples/ensemble/plot_adaboost_twoclass.html#sphx-glr-download-auto-examples-ensemble-plot-adaboost-twoclass-py

Demo: Multi-Class AdaBoost

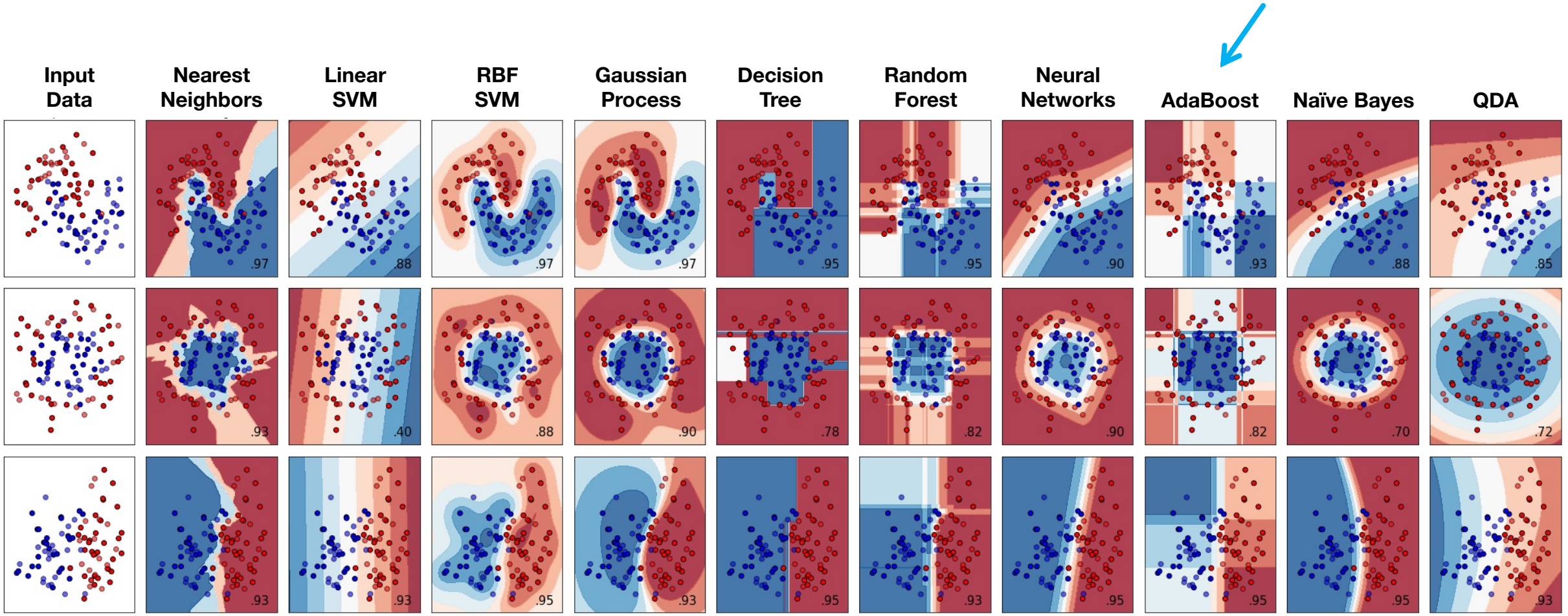
Training error decreases rapidly—test error may plateau or increase



https://scikit-learn.org/stable/auto_examples/ensemble/plot_adaboost_hastie_10_2.html#sphx-glr-auto-examples-ensemble-plot-adaboost-hastie-10-2-py

Comparing Decision Boundaries

AdaBoost balances flexibility and generalization



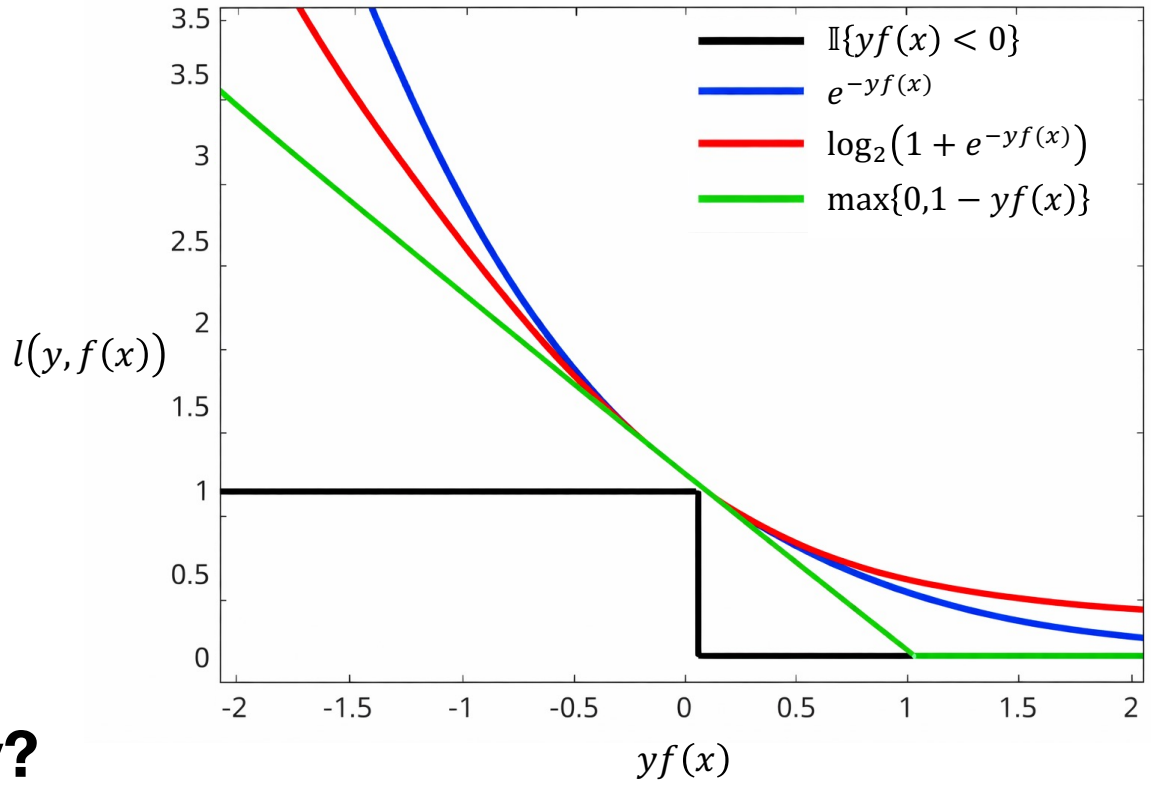
Why Boosting Works

Loss Functions for Classification

Boosting replaces the nonconvex 0–1 loss with a **smooth, convex surrogate** to facilitate convex optimization

- Loss function: $l\big(y, f(x)\big)$
- $\mathbb{I}\{yf(x) < 0\}$ directly measures misclassification but is nonconvex
- Nonconvex, replacing it using convex upper bound, e.g., exponential loss

$$l\big(y, f(x)\big) = e^{-yf(x)}$$



What does $yf(x)$ represent geometrically?

Understanding Boosting: Exponential Loss

Combined classifier:

$$f_t(x) = \alpha_1 h_1(x;\theta_1) + \cdots + \alpha_t h_t(x;\theta_t)$$

Exponential loss:

$$\hat{L}(f) = \frac{1}{m}\sum_{i=1}^{m} e^{-y^i f_t(x^i)}$$

$$= \frac{1}{m}\sum_{i=1}^{m} e^{-y^i\left(f_{t-1}(x^i)+\alpha_t h_t(x^i;\theta_t)\right)}$$

$$= \sum_{i=1}^{m} \underbrace{\frac{1}{m} e^{-y^i f_{t-1}(x^i)}}_{D_t(i)\prod_{s=1}^{t-1} Z_s} e^{-\alpha_t y^i h_t(x^i;\theta_t)}$$

Why does the loss factor into a previous term times a new exponential?

→ Proof: next slide

Property for $D_t(i)$

Carryover past importance

The key update

Weight update rule:

$$D_{t+1}(i) = \frac{D_t(i)}{Z_t} e^{-\alpha_t y^i h_t(x^i)}$$

$$Z_t = \sum_{i=1}^{m} D_t(i) e^{-\alpha_t y^i h_t(x^i)}$$

Recursive substitution:

$$= \frac{D_{t-1}(i)}{Z_t Z_{t-1}} e^{-\alpha_{t-1} y^i h_{t-1}(x^i)} e^{-\alpha_t y^i h_t(x^i)}$$

… keep expanding

Fully unrolled form:

$$= \frac{e^{-y^i \sum_{s=1}^{t} \alpha_s h_s(x^i)}}{m \prod_{s=1}^{t} Z_s}$$

$$D_1(i) = \frac{1}{m}$$

Express using ensemble model:

$$= \frac{e^{-y^i f_t(x^i)}}{m \prod_{s=1}^{t} Z_s}$$

Finding h_t: Linearization of Loss Function

Assuming α_t is small, the loss criterion can be approximated by Taylor expansion

$$e^{-\alpha_t y^i h_t(x^i;\theta_t)} \approx 1 - \alpha_t y^i h_t(x^i;\theta_t)$$

$$e^z = 1 + z + \frac{z^2}{2!} + \frac{z^3}{3!} + \cdots$$

We minimize the loss with respect to the new classifier, which reduces to a weighted error under D_t:

$$\hat{L}(f) = \left(\prod_{s=1}^{t-1} Z_s\right) \sum_{i=1}^{m} D_t(i)\, e^{-\alpha_t y^i h_t(x^i;\theta_t)}$$

$$\propto \sum_{i=1}^{m} D_t(i)\left(1 - \alpha_t y^i h_t(x^i;\theta_t)\right) = \sum_{i=1}^{m} D_t(i) - \alpha_t \sum_{i=1}^{m} D_t(i)\, y^i h_t(x^i;\theta_t)$$

$= 1$

Finding h_t: Weighted Agreement

At stage t we find θ_t that maximize "weighted agreement":

$$\theta_t \leftarrow \arg\max_{\theta} \sum_{i=1}^{m} D_t(i)\, y^i h_t\left(x^i; \theta\right)$$

Note that $y \in \{\pm 1\}$, and $h_t(x) \in \{\pm 1\}$

$$y^i h_t\left(x^i; \theta\right) \in \{\pm 1\}$$

This approximation minimizes

$$\epsilon_t := \sum_{i=1}^{m} D_t(i)\, \mathbb{I}\left\{y^i \neq h_t\left(x^i; \theta\right)\right\}$$

→ AdaBoost's weighted error term

Finding α_t: Weight of Weak Learner

Then we go back and find the "weight" α_t associated with the new classifier by minimizing the **original** weighted (exponential) loss

$$L(\alpha_t) = \sum_{i=1}^{m} \underbrace{\frac{1}{m} e^{-y^i f_{t-1}(x^i)}}_{D_t(i) \prod_{s=1}^{t-1} Z_s} e^{-\alpha_t y^i h_t(x^i;\theta_t)} \propto \sum_{i=1}^{m} D_t(i)\, e^{-\alpha_t y^i h_t(x^i;\theta_t)}$$

Set derivative to 0 and solve for α_t

$$\frac{\partial L}{\partial \alpha_t} = -\sum_{i=1}^{m} D_t(i) e^{-\alpha_t y^i h_t(x^i;\theta_t)}\, y^i h_t(x^i;\theta_t) = 0$$

$$\frac{d}{d\alpha_t} e^u = e^u \frac{du}{d\alpha_t}$$

What happens to α_t when the classifier performs no better than random?

Finding α_t: Weight of Weak Learner (cont.)

Note that $y^i \in \{\pm 1\}$, and $h_t(x^i) \in \{\pm 1\}$, and therefore, $y^i h_t(x^i;\theta) \in \{\pm 1\}$

$$\frac{\partial L}{\partial \alpha_t} = -\sum_{i=1}^{m} D_t(i) e^{-\alpha_t y^i h_t(x^i;\theta_t)} y^i h_t(x^i;\theta_t)$$

$$= -\sum_{i:h_t(x^i)=y^i} D_t(i) e^{-\alpha_t} + \sum_{i:h_t(x^i)\neq y^i} D_t(i) e^{\alpha_t}$$

$$= (\epsilon_t - 1) e^{-\alpha_t} + \epsilon_t e^{\alpha_t} = 0$$

$$\Rightarrow \quad \alpha_t = \frac{1}{2} \ln\left(\frac{1-\epsilon_t}{\epsilon_t}\right) \quad \rightarrow$$

How much we trust this learner

Extensions

AdaBoost Extensions

Extensions focus on improving robustness, flexibility, and performance—examples: LPBoost, TotalBoost, BrownBoost, MadaBoost, LogitBoost, and XGBoost

Generalization: Gradient Boosting

- General framework for regression and classification
- Builds an ensemble of weak learners (typically shallow trees) by iteratively minimizing a chosen differentiable loss function
- Supports flexible loss functions (e.g., squared loss, logistic loss)
- Optimizes the model via gradient descent in function space
- Achieves strong performance in modern large-scale applications

Key Takeaways

What We Learned This Week

- Boosting combines many weak learners into a strong model by training them sequentially, with each learner focusing on previous mistakes
- AdaBoost reweights training examples and assigns higher influence to more accurate weak learners, resulting in a weighted ensemble
- Decision stumps and shallow trees are effective weak learners that can yield highly nonlinear decision boundaries when boosted
- In binary classification, AdaBoost produces decision scores whose sign determines the class and whose magnitude reflects prediction confidence
- In multi-class settings, AdaBoost (SAMME) significantly improves accuracy over a single tree and naive baselines through stagewise additive modeling
- AdaBoost can be interpreted as greedy minimization of exponential loss, connecting ensemble learning to optimization and gradient boosting