

# Machine Learning (CE 40717)

## Fall 2025

Ali Sharifi-Zarchi

CE Department  
Sharif University of Technology

November 6, 2025



- 1 Introduction
- 2 Bagging
- 3 Boosting
- 4 AdaBoost
- 5 Comparison
- 6 References

## 1 Introduction

Condorcet's jury theorem

Ensemble learning

Ensemble Methods

## 2 Bagging

## 3 Boosting

## 4 AdaBoost

## 5 Comparison

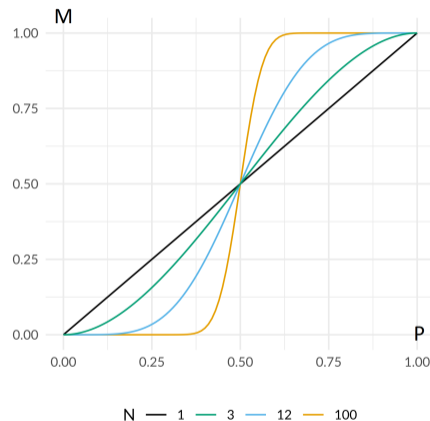
## 6 References

## 6 References

## Ensemble Methods

# Condorcet's jury theorem

- $N$  voters decide by **majority vote**.
- $\mathbb{P}(\text{vote correct}) = p$  independently for each voter.
- Let  $M = \mathbb{P}(\text{majority correct})$ .
- If  $p > 0.5$  and  $N \rightarrow \infty$ , then  $M \rightarrow 1$ .
  - Intuition: averaging many slightly-better-than-chance votes.



Adapted from Wikipedia

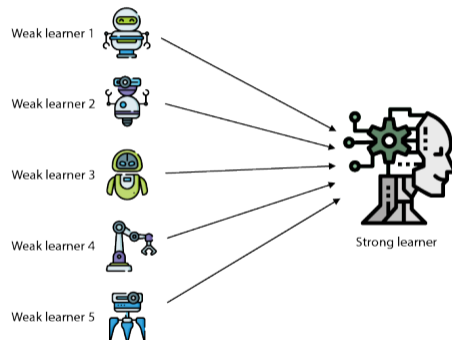


# Strong vs. weak Learners

- **Strong learner:** we seek to produce one classifier for which the classification error can be made arbitrarily small.
  - So far we were looking for such methods.
- **Weak learner:** a classifier which is just better than random guessing (for now this will be our only expectation).

# Basic idea

- Certain **weak learners** do well in modeling one aspect of the data, while others do well in modeling another.
- Learn several simple models and **combine** their outputs to produce the final decision.
- A **composite prediction** where the final accuracy is **better** than the accuracy of **individual models**.

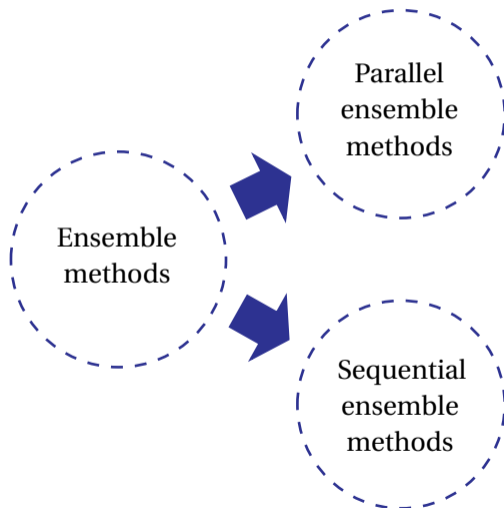


Adapted from [4]

## 6 References

## Ensemble Methods

# Ensemble Methods



- Weak learners are generated in **parallel**.
- Basic motivation is to use **independence** between the learners.

- Weak learners are generated **consecutively**.
- Basic motivation is to use **dependence** between the base learners.

# What we talk about

- Weak or simple learners
  - **Low variance:** they don't usually overfit
  - **High bias:** they can't learn complex functions
- **Bagging** (parallel): To decrease the variance
  - Random Forest
- **Boosting** (sequential): To decrease the bias (enhance their capabilities)
  - AdaBoost

## 1 Introduction

## 2 Bagging

## Basic idea & algorithm

## Decision tree (quick review)

## Random Forest

### ③ Boosting

#### 4 AdaBoost

## 5 Comparison

## 6 References

## 1 Introduction

## 2 Bagging

## Basic idea & algorithm

## Decision tree (quick review)

## Random Forest

### ③ Boosting

#### 4 AdaBoost

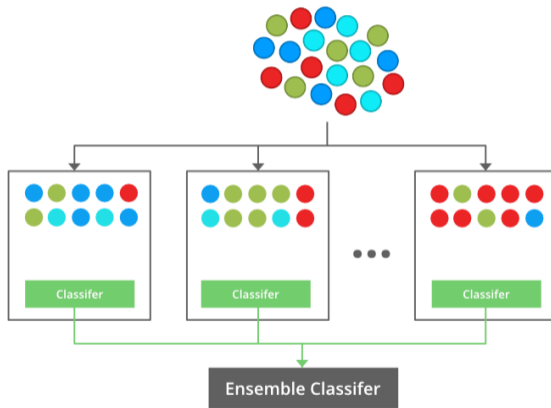
## 5 Comparison

## 6 References

# Basic idea

- **Bagging** = **B**ootstrap **a**ggregating
- It uses **bootstrap resampling** to generate different training datasets from the original training dataset.
  - Samples training data uniformly at random with replacement.
- On the training datasets, it trains different weak learners.
- During testing, it **aggregates** the weak learners by uniform averaging or majority voting.
  - Works best with unstable models (high variance models). Why?

# Basic idea, Cont.



Original Data

Bootstrapping

Aggregating

Bagging

Adapted from GeeksForGeeks

# Algorithm

---

## Algorithm 1 Bagging

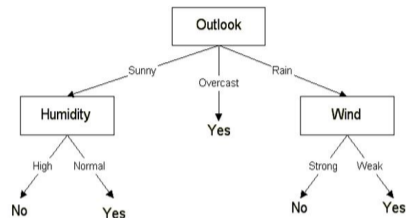
---

- 1: **Input:**  $M$  (required ensemble size),  $D = \{(\mathbf{x}^{(1)}, y^{(1)}), \dots, (\mathbf{x}^{(N)}, y^{(N)})\}$  (training set)
  - 2: **for**  $t = 1$  to  $M$  **do**
  - 3:     Build a dataset  $D_t$  by sampling  $N$  items randomly with replacement from  $D$   
    ▷ *Bootstrap resampling: like rolling an  $N$ -sided die  $N$  time*
  - 4:     Train a model  $h_t$  using  $D_t$  and add it to the ensemble
  - 5: **end for**
  - 6:  $H(\mathbf{x}) = \text{sign}(\sum_{t=1}^M h_t(\mathbf{x}))$   
    ▷ *Aggregate models by voting for classification or by averaging for regression*
-



# Structure

- **Terminal nodes** (leaves) represent target variable.
- Each **internal node** denotes a test on an attribute.



Outlook	Temperature	Humidity	Wind	Played football(yes/no)
Sunny	Hot	High	Weak	No
Sunny	Hot	High	Strong	No
Overcast	Hot	High	Weak	Yes
Rain	Mild	High	Weak	Yes
Rain	Cool	Normal	Weak	Yes
Rain	Cool	Normal	Strong	No
Overcast	Cool	Normal	Strong	Yes
Sunny	Mild	High	Weak	No
Sunny	Cool	Normal	Weak	Yes
Rain	Mild	Normal	Weak	Yes
Sunny	Mild	Normal	Strong	Yes
Overcast	Mild	High	Strong	Yes
Overcast	Hot	Normal	Weak	Yes
Rain	Mild	High	Strong	No

Adapted from Medium

# Learning

- Learning an optimal decision tree is **NP-Complete**.
  - Instead, we use a **greedy search** based on a heuristic.
  - We can't guarantee to return the globally-optimal decision tree.
- The most common strategy for DT learning is a greedy top-down approach.
- Tree is constructed by splitting samples into subsets based on an **attribute value test** in a recursive manner.

Adapted from G.E. Naumov, "NP-completeness of problems of construction of optimal decision trees", 1991

# Algorithm

---

## Algorithm 2 Constructing DT

---

```

1: procedure FINDTREE( $S, A$ )                                ▷ Input:  $S$  (samples),  $A$  (attributes)
2:   if  $A$  is empty or all labels in  $S$  are the same then
3:     status  $\leftarrow$  leaf
4:     class  $\leftarrow$  most common class in  $S$ 
5:   else
6:     status  $\leftarrow$  internal
7:      $a \leftarrow$  bestAttribute( $S, A$ )                        ▷ The attribute value test
8:     LeftNode  $\leftarrow$  FindTree( $S(a > t), A - \{a\}$ )        ▷  $t$  (threshold)
9:     RightNode  $\leftarrow$  FindTree( $S(a \leq t), A - \{a\}$ )
10:   end if
11: end procedure

```

---

# Which attribute is the best?

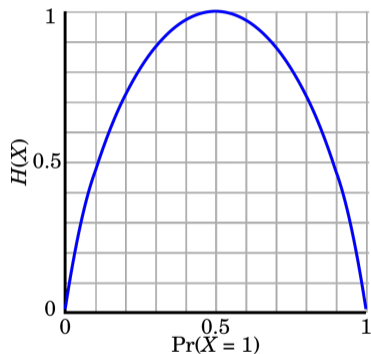
- **Entropy** measures label uncertainty.

$$H_S(Y) = - \sum_{y \in \text{Values}(Y)} \mathbb{P}_S(Y=y) \log_2 \mathbb{P}_S(Y=y)$$

- **Information Gain (IG)**

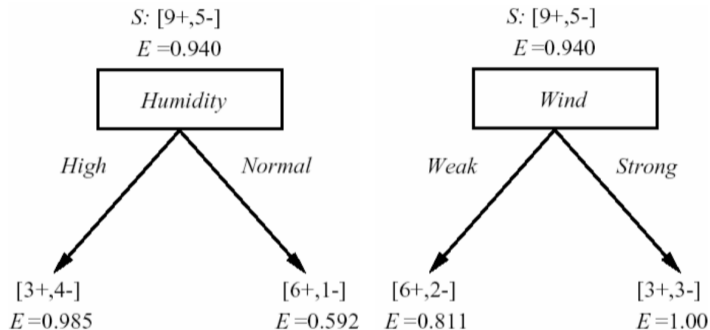
$$\text{Gain}(S,A) = H_S(Y) - \sum_{v \in \text{Values}(A)} \frac{|S_v|}{|S|} H_{S_v}(Y)$$

A: splitting attribute    Y: target    S: samples  
S<sub>v</sub>: subset with A = v



Adapted from Wikipedia

# Example



Adapted from [5]

$$\text{Gain}(S, \text{Humidity}) = 0.940 - (7/14)0.985 - (7/14)0.592 = 0.151$$

$$\text{Gain}(S, \text{Wind}) = 0.940 - (8/14)0.811 - (6/14)1.0 = 0.048$$

## 1 Introduction

## 2 Bagging

## Basic idea & algorithm

## Decision tree (quick review)

## Random Forest

### 3 Boosting

## 4 AdaBoost

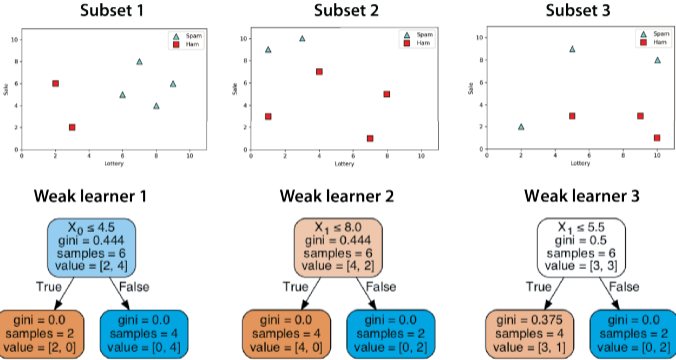
## 5 Comparison

## 6 References

# Bagging on decision trees?

Why decision trees?

- Interpretable
- Robust to outliers
- **Low bias**
- **High variance**



Adapted from [4]

## Perfect candidates

- Why are **DTs** good candidates for ensembles?
  - Consider averaging many (nearly) **unbiased** tree estimators.
  - Bias remains similar, but **variance is reduced**.
- Remember Bagging?
  - Train many trees on bootstrapped data, then aggregate (average/majority) the outputs.

# Algorithm

---

## Algorithm 3 Random Forest

---

- 1: **Input:**  $T$  (number of trees),  $m$  (number of variables used to split each node)
  - 2: **for**  $t = 1$  to  $T$  **do**
  - 3:     Draw a bootstrap dataset
  - 4:     At each node, sample  $m$  features uniformly from  $\{1, \dots, d\}$  as split candidates
  - 5: **end for**
  - 6: **Output:**
  - 7:     Regression: average of the outputs
  - 8:     Classification: majority voting
- 

▷ Usually:  $m \leq \sqrt{d}$

# Why Random Forest Works: Variance & Correlation

Consider  $M$  base learners with variance  $\sigma^2$  and pairwise correlation  $\rho$ . For the averaged predictor  $\bar{h}(\mathbf{x}) = \frac{1}{M} \sum_{t=1}^M h_t(\mathbf{x})$ :

$$\text{Var}(\bar{h}) = \rho \sigma^2 + \frac{1 - \rho}{M} \sigma^2$$

- **Bagging** reduces  $\frac{1-\rho}{M} \sigma^2$  via averaging ( $M \uparrow$ ).
- **Random Forest** also lowers  $\rho$  by random feature subsampling at each split.
- Takeaway: smaller  $\rho$  and larger  $M \Rightarrow$  lower ensemble variance without increasing bias much.

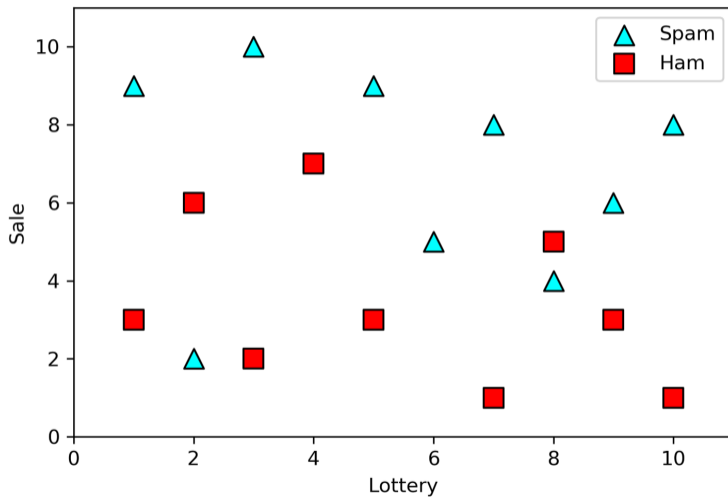
# Out-of-Bag (OOB) Estimation

- In each bootstrap sample of size  $N$ , the probability an example is *not* selected is  $\left(1 - \frac{1}{N}\right)^N \approx e^{-1} \approx 36.8\%$ .
- Use OOB samples for each tree to estimate generalization error *without* a separate validation split.
- Practical tips:
  - Plot OOB error vs. number of trees; stop once the curve flattens.
  - OOB can also support permutation-based feature importance.

## Feature Importance in Forests (MDI vs Permutation)

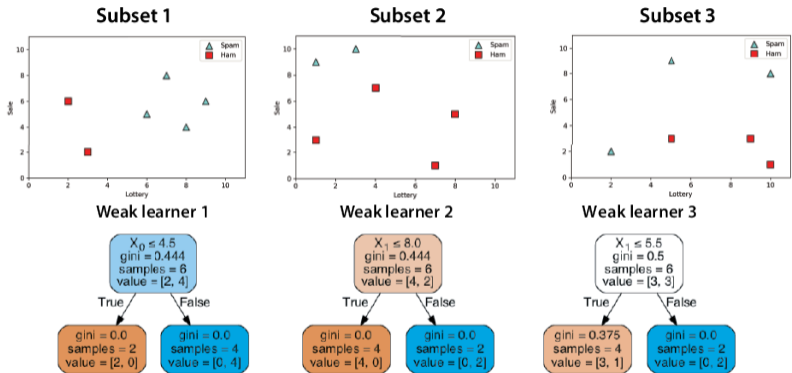
- **MDI (Gini/Entropy decrease)**: fast, but biased toward high-cardinality or noisy features.
- **Permutation importance (OOB)**: shuffle a feature and measure OOB error increase.
- Guidelines:
  - Prefer permutation importance for **comparisons** across features.
  - Beware correlation: grouped or conditional permutation can help.
  - For deeper interpretability, use PDP/ICE; for local attributions, consider SHAP (optional).

# Example



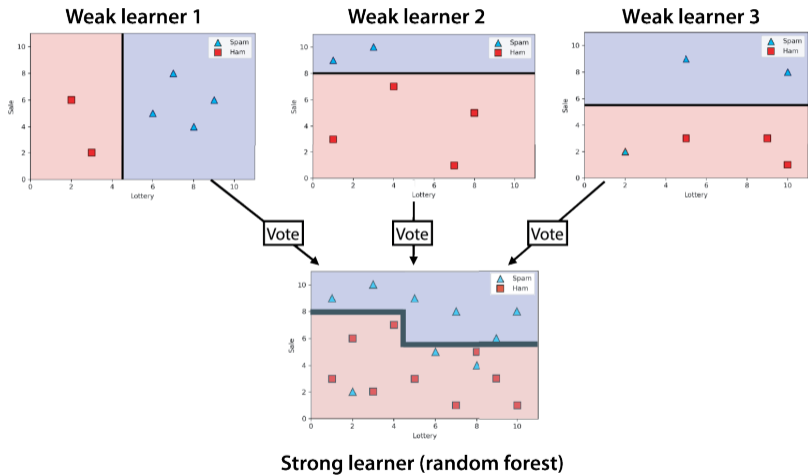
Adapted from [4]

# Example, Cont.



Adapted from [4]

# Example, Cont.



Adapted from [4]

## 1 Introduction

## ② Bagging

### ③ Boosting

Motivation & basic idea  
Algorithm

## 4 AdaBoost

## 5 Comparison

## 6 References

1 Introduction

2 Bagging

3 Boosting

Motivation & basic idea

Algorithm

4 AdaBoost

5 Comparison

6 References

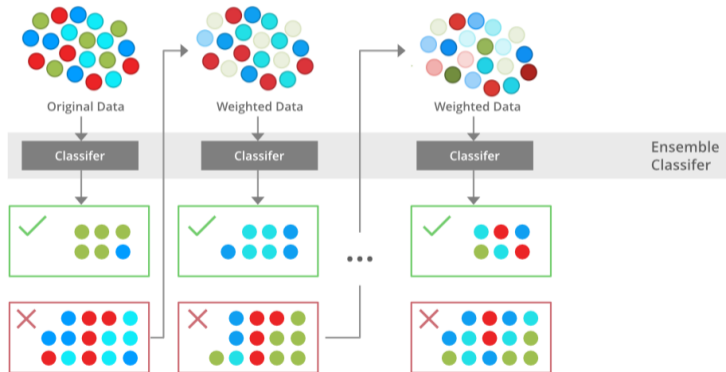
# Problems with bagging

- Bagging created a diversity of **weak learners** by creating random datasets.
  - Examples: Decision stumps (shallow decision trees), Logistic regression, ...
- Did we have full control over the usefulness of the weak learners?
  - The **diversity** or **complementarity** of the weak learners is not controlled in any way, it is left to chance and to the instability (variance) of the models.

# Basic idea

- We would expect a better performance if the weak learners also **complemented** each other.
  - They would have “expertise” on different subsets of the dataset.
  - So they would work better on different subsets.
- The basic idea of boosting is to generate a **series** of weak learners which complement each other.
  - For this, we will force each learner to focus on **the mistakes of the previous learner**.

# Basic idea, Cont.



Adapted from GeeksForGeeks

1 Introduction

2 Bagging

3 Boosting

Motivation & basic idea

Algorithm

4 AdaBoost

5 Comparison

6 References

# Algorithm

- Try to combine many simple **weak** learners (in sequence) to find a single **strong** learner (For simplicity, suppose that we have a classification problem from now on).
  - Each component is a simple binary  $\pm 1$  classifier
  - Voted combinations of component classifiers

$$H_M(\mathbf{x}) = \alpha_1 h(\mathbf{x}; \boldsymbol{\theta}_1) + \cdots + \alpha_M h(\mathbf{x}; \boldsymbol{\theta}_M)$$

- To simplify notations:  $h(\mathbf{x}; \boldsymbol{\theta}_i) = h_i(\mathbf{x})$

$$H_M(\mathbf{x}) = \alpha_1 h_1(\mathbf{x}) + \cdots + \alpha_M h_M(\mathbf{x})$$

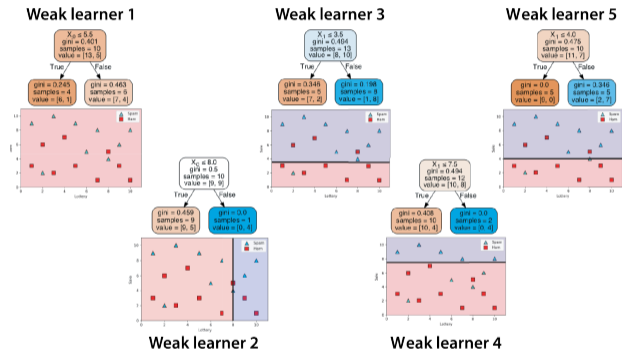
- **Prediction:**  $\hat{y} = \text{sign}(H_M(\mathbf{x}))$

Candidate for  $h_i(\mathbf{x})$

- Decision stumps
- Each classifier is based on only a single feature of  $\mathbf{x}$  (e.g.,  $x_k$ ):

$$h(\mathbf{x};\boldsymbol{\theta}) = \text{sign}(w_1 x_k - w_0)$$

$$\boldsymbol{\theta} = \{k, w_1, w_0\}$$



Adapted from [4]

# Gradient Boosting (Functional Gradient Descent)

- View boosting as minimizing a loss  $L(y, F(\mathbf{x}))$  by **steepest descent in function space**.

$$F_m(\mathbf{x}) = F_{m-1}(\mathbf{x}) + \nu \beta_m T_m(\mathbf{x}), \quad r_{im} = - \left[ \frac{\partial L(y_i, F)}{\partial F} \right]_{F=F_{m-1}(\mathbf{x}^{(i)})}$$

- Fit a tree  $T_m$  to pseudo-residuals  $\{r_{im}\}$ , choose  $\beta_m$  by line search;  $\nu \in (0, 1]$  is the **learning rate**.
- Choice of  $L$ : square loss (regression), logistic loss (classification), etc.
- Regularization: small  $\nu$ , shallow trees, subsampling of rows/columns.

## 6 References

## 1 Introduction

## ② Bagging

### ③ Boosting

#### 4 AdaBoost

## Basic idea & algorithm

## Loss function & proof

## Properties (extra-reading)

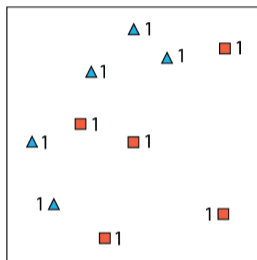
## 5 Comparison

## 6 References

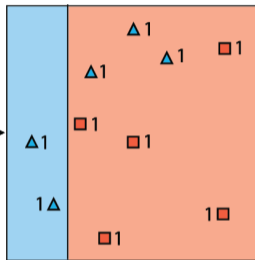
# Basic idea

- **Sequential** production of classifiers
  - Iteratively add the classifier whose addition will be most helpful.
- Represent the importance of each sample by assigning **weights** to them.
  - Correct classification  $\Rightarrow$  smaller weights
  - Misclassified samples  $\Rightarrow$  larger weights
- Each classifier is **dependent** on the previous ones.
  - Focuses on the **previous ones' error**.

# Example

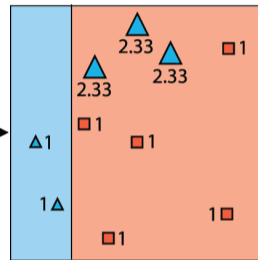


Add a weight of 1  
to every point.



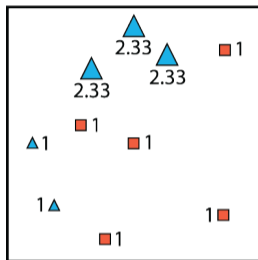
Fit a weak learner.  
Correct: 7  
Incorrect: 3

Adapted from [4]

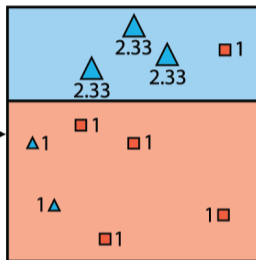


Rescale misclassified  
points by 7/3.

# Example, Cont.

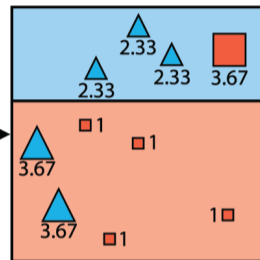


The rescaled dataset



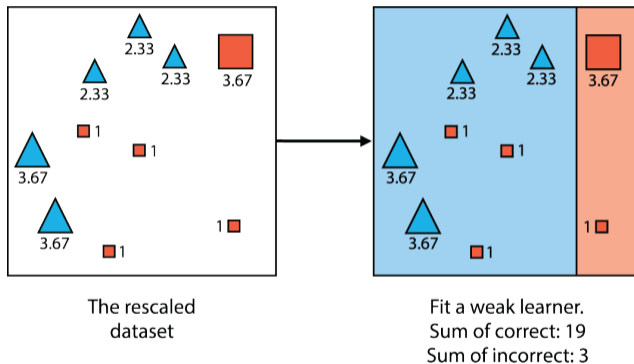
Fit a weak learner.  
Sum of correct: 11  
Sum of incorrect: 3

Adapted from [4]



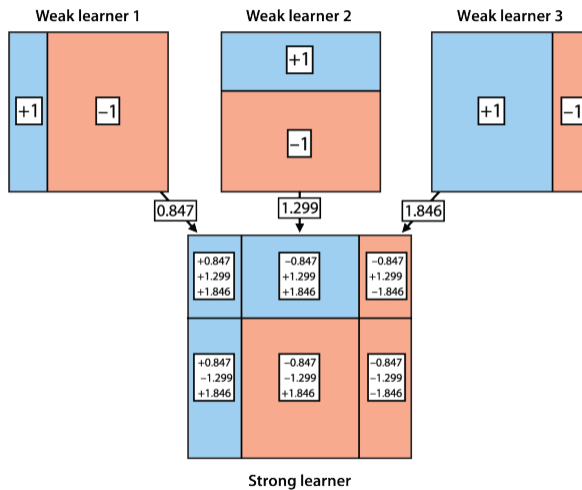
Rescale misclassified points by  $11/3$ .

## Example, Cont.



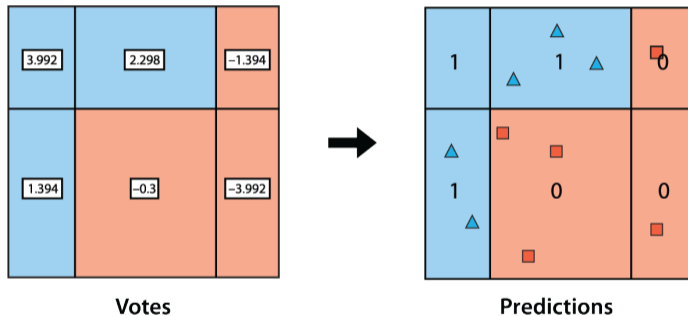
Adapted from [4]

# Example, Cont.



Adapted from [4]

# Example, Cont.



Adapted from [4]

# Algorithm

- $H_M(\mathbf{x}) = \frac{1}{2}[\alpha_1 h_1(\mathbf{x}) + \dots + \alpha_M h_M(\mathbf{x})] \rightarrow$  the complete model  $y^{(i)} \in \{-1, 1\}$ 
  - $h_m(\mathbf{x})$ :  $m$ -th weak learner
  - $\alpha_m = ? \rightarrow$  votes of the  $m$ -th weak learner
- $w_m^{(i)}$ : weight of sample  $i$  in iteration  $m$ 
  - $w_{m+1}^{(i)} = ?$
- $J_m = \sum_{i=1}^N w_m^{(i)} \times \mathbb{I}(y^{(i)} \neq h_m(\mathbf{x}^{(i)})) \rightarrow$  loss of the  $m$ -th weak learner
- $\epsilon_m = \frac{\sum_{i=1}^N w_m^{(i)} \times \mathbb{I}(y^{(i)} \neq h_m(\mathbf{x}^{(i)}))}{\sum_{i=1}^N w_m^{(i)}} \rightarrow$  weighted error of the  $m$ -th weak learner

## Algorithm, Cont.

- $H_M(\mathbf{x}) = \frac{1}{2}[\alpha_1 h_1(\mathbf{x}) + \dots + \alpha_M h_M(\mathbf{x})] \longrightarrow$  the complete model  $y^{(i)} \in \{-1, 1\}$ 
  - $h_m(\mathbf{x})$ :  $m$ -th weak learner
  - $\alpha_m = \ln\left(\frac{1 - \epsilon_m}{\epsilon_m}\right) \longrightarrow$  votes of the  $m$ -th weak learner
- $w_m^{(i)}$ : weight of sample  $i$  in iteration  $m$ 
  - $w_{m+1}^{(i)} = w_m^{(i)} e^{\alpha_m \mathbb{I}(y^{(i)} \neq h_m(\mathbf{x}^{(i)}))}$
- $J_m = \sum_{i=1}^N w_m^{(i)} \times \mathbb{I}(y^{(i)} \neq h_m(\mathbf{x}^{(i)})) \longrightarrow$  loss of the  $m$ -th weak learner
- $\epsilon_m = \frac{\sum_{i=1}^N w_m^{(i)} \times \mathbb{I}(y^{(i)} \neq h_m(\mathbf{x}^{(i)}))}{\sum_{i=1}^N w_m^{(i)}} \longrightarrow$  weighted error of the  $m$ -th weak learner

# Algorithm, Cont.

---

## Algorithm 4 AdaBoost

---

1: Initialize data weight  $w_1^{(i)} = \frac{1}{N}$  for all  $N$  samples

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

3:     Find  $h_m(\mathbf{x})$  by minimizing the loss:

$$J_m = \sum_{i=1}^N w_m^{(i)} \times \mathbb{I}(y^{(i)} \neq h_m(\mathbf{x}^{(i)}))$$

4:     Find the weighted error of  $h_m(\mathbf{x})$ :

$$\epsilon_m = \frac{\sum_{i=1}^N w_m^{(i)} \times \mathbb{I}(y^{(i)} \neq h_m(\mathbf{x}^{(i)}))}{\sum_{i=1}^N w_m^{(i)}}$$

5:     Assign votes  $\alpha_m = \ln\left(\frac{1 - \epsilon_m}{\epsilon_m}\right)$

6:     Update the weights:

$$w_{m+1}^{(i)} = w_m^{(i)} e^{\alpha_m \mathbb{I}(y^{(i)} \neq h_m(\mathbf{x}^{(i)}))}$$

7: **end for**

8: **Combined classifier:**  $\hat{y} = \text{sign}(H_M(\mathbf{x}))$  where  $H_M(\mathbf{x}) = \frac{1}{2} \sum_{m=1}^M \alpha_m h_m(\mathbf{x})$

---

## 6 References

# Loss function

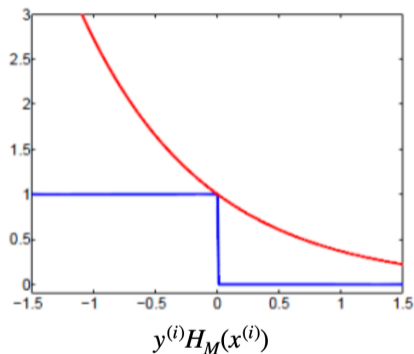
- There are many options for the loss function.
  - AdaBoost is equivalent to using the following **exponential loss**.

$$\mathcal{L}(y, H_M(\mathbf{x})) = e^{-y \times H_M(\mathbf{x})}$$

$$\hat{y} = \text{sign}(H_M(\mathbf{x}))$$

# Why the exponential loss?

- Differentiable approximation (bound) of the **0/1 loss**
  - Easy to optimize
  - Optimizing an upper bound on classification error.



Adapted from [2]

# Step 1: Calculating the exponential loss

- We need to calculate the exponential loss for:

$$H_m(\mathbf{x}) = \frac{1}{2} [\alpha_1 h_1(\mathbf{x}) + \dots + \alpha_m h_m(\mathbf{x})]$$

$\uparrow$

To have a cleaner form later

- Idea:** consider adding the  $m$ -th component:

$$\begin{aligned} \mathcal{L}_m &= \sum_{i=1}^N e^{-y^{(i)} H_m(\mathbf{x}^{(i)})} = \sum_{i=1}^N e^{-y^{(i)} [H_{m-1}(\mathbf{x}^{(i)}) + \frac{1}{2} \alpha_m h_m(\mathbf{x}^{(i)})]} \\ &= \sum_{i=1}^N e^{-y^{(i)} H_{m-1}(\mathbf{x}^{(i)})} e^{-\frac{1}{2} \alpha_m y^{(i)} h_m(\mathbf{x}^{(i)})} = \sum_{i=1}^N \underbrace{e^{-y^{(i)} H_{m-1}(\mathbf{x}^{(i)})}}_{w_m^{(i)}} e^{-\frac{1}{2} \alpha_m y^{(i)} h_m(\mathbf{x}^{(i)})} \end{aligned}$$

$\uparrow$   
 Suppose it is fixed at stage  $m$

$\uparrow$   
 Should be optimized at stage  $m$  by seeking  $h_m(\mathbf{x})$  and  $\alpha_m$

# Step 2: Deriving the weighted error function

- We need to derive the weighted error function,  $J_m$

$$\begin{aligned}
 \mathcal{L}_m &= \sum_{i=1}^N w_m^{(i)} e^{-\frac{1}{2} \alpha_m y^{(i)} h_m(\mathbf{x}^{(i)})} \\
 &= e^{\frac{-\alpha_m}{2}} \left( \sum_{y^{(i)} = h_m(\mathbf{x}^{(i)})} w_m^{(i)} \right) + e^{\frac{\alpha_m}{2}} \left( \sum_{y^{(i)} \neq h_m(\mathbf{x}^{(i)})} w_m^{(i)} \right) \\
 &= (e^{\frac{\alpha_m}{2}} - e^{\frac{-\alpha_m}{2}}) \underbrace{\left( \sum_{y^{(i)} \neq h_m(\mathbf{x}^{(i)})} w_m^{(i)} \right)}_{J_m} + e^{\frac{-\alpha_m}{2}} \left( \sum_{i=1}^N w_m^{(i)} \right) \\
 J_m &= \sum_{i=1}^N w_m^{(i)} \times \underset{\uparrow}{\mathbb{I}} \left( y^{(i)} \neq h_m(\mathbf{x}^{(i)}) \right)
 \end{aligned}$$

Find  $h_m(\mathbf{x})$  that minimizes  $J_m$

## Step 3: Deriving $\epsilon_m$ and $\alpha_m$

- We need to derive  $\epsilon_m$  and  $\alpha_m$  by setting the derivative equal to zero:

$$\frac{\partial \mathcal{L}_m}{\partial \alpha_m} = 0$$

- Idea:** separate the derivative into misclassified and correctly classified samples.

$$\Rightarrow \frac{1}{2} (e^{\frac{\alpha_m}{2}} + e^{-\frac{\alpha_m}{2}}) \left( \sum_{y^{(i)} \neq h_m(\mathbf{x}^{(i)})} w_m^{(i)} \right) = \frac{1}{2} e^{-\frac{\alpha_m}{2}} \left( \sum_{i=1}^N w_m^{(i)} \right)$$

$$\Rightarrow \frac{e^{-\frac{\alpha_m}{2}}}{(e^{\frac{\alpha_m}{2}} + e^{-\frac{\alpha_m}{2}})} = \frac{\sum_{y^{(i)} \neq h_m(\mathbf{x}^{(i)})} w_m^{(i)}}{\sum_{i=1}^N w_m^{(i)}}$$

- Set  $\epsilon_m = \frac{\sum_{i=1}^N w_m^{(i)} \mathbb{I}(y^{(i)} \neq h_m(\mathbf{x}^{(i)}))}{\sum_{i=1}^N w_m^{(i)}} \Rightarrow \alpha_m = \ln \left( \frac{1 - \epsilon_m}{\epsilon_m} \right)$

## Step 4: Justifying the weight update mechanism

- We need to justify the weight update mechanism.
- Idea:** we have  $w_m^{(i)}$  from the first step as  $w_{m+1}^{(i)} = e^{-y^{(i)} H_m(\mathbf{x}^{(i)})}$

$$\xRightarrow{\text{separate } h_m(\mathbf{x}^{(i)})} w_{m+1}^{(i)} = w_m^{(i)} e^{-\frac{1}{2} \alpha_m y^{(i)} h_m(\mathbf{x}^{(i)})}$$

$$\xRightarrow{y^{(i)} h_m(\mathbf{x}^{(i)}) = 1 - 2\mathbb{I}(y^{(i)} \neq h_m(\mathbf{x}^{(i)}))} w_{m+1}^{(i)} = w_m^{(i)} e^{-\frac{1}{2} \alpha_m} e^{\alpha_m \mathbb{I}(y^{(i)} \neq h_m(\mathbf{x}^{(i)}))}$$

↑

Independent of  $i$  and can be ignored

$$\Rightarrow w_{m+1}^{(i)} = w_m^{(i)} e^{\alpha_m \mathbb{I}(y^{(i)} \neq h_m(\mathbf{x}^{(i)}))}$$

## 1 Introduction

## ② Bagging

### ③ Boosting

## 4 AdaBoost

## Basic idea & algorithm

## Loss function & proof

### Properties (extra-reading)

## 5 Comparison

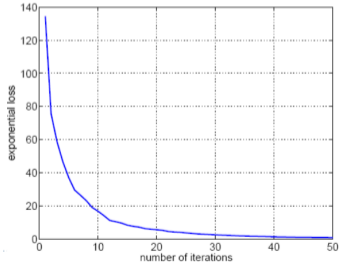
## 6 References

# Exponential loss properties

- In each boosting iteration, assuming we can find  $h(\mathbf{x}; \boldsymbol{\theta}_m)$  whose weighted error is better than chance.

$$H_m(\mathbf{x}) = \frac{1}{2} [\alpha_1 h(\mathbf{x}; \boldsymbol{\theta}_1) + \dots + \alpha_m h(\mathbf{x}; \boldsymbol{\theta}_m)]$$

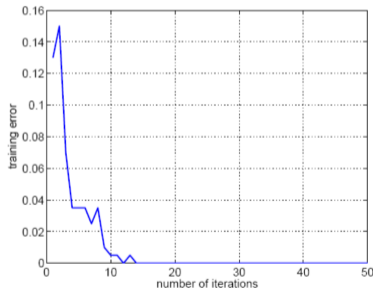
- Thus, **lower exponential loss** over training data is guaranteed.



Adapted from [6]

# Training error properties

- Boosting iterations typically **decrease** the **training error** of  $H_M(\mathbf{x})$  over training examples.

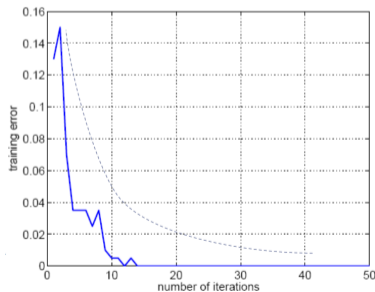


Adapted from [6]

## Training error properties, Cont.

- **Training error** has to go **down exponentially fast** if the weighted error of each  $h_m$  is strictly better than chance (i.e.,  $\epsilon_m < 0.5$ )

$$E_{\text{train}}(H_M) \leq \prod_{m=1}^M 2\sqrt{\epsilon_m(1-\epsilon_m)}$$

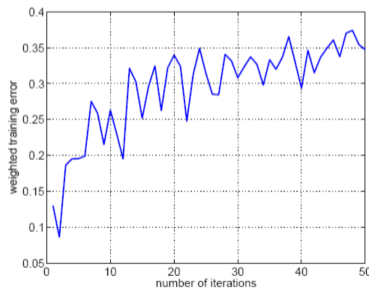


Adapted from [6]

## Weighted error properties

- **Weighted error** of each new component classifier tends to **increase** as a function of boosting iterations.

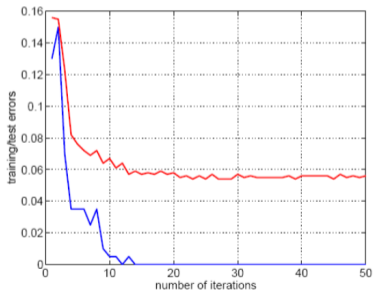
$$\epsilon_m = \frac{\sum_{i=1}^N w_m^{(i)} \mathbb{I}(y^{(i)} \neq h_m(\mathbf{x}^{(i)}))}{\sum_{i=1}^N w_m^{(i)}}$$



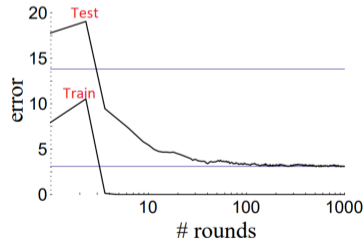
Adapted from [6]

# Test error properties

- **Test error** can still **decrease** after training error is flat (even zero).
- But, is it robust to overfitting?
  - May easily overfit in the presence of labeling noise or overlap of classes.



Adapted from [6]



Adapted from [3]

# Boosting in Practice: Avoiding Overfitting

- **Label noise & overlap:** exponential loss can overweight hard/noisy points.
- Remedies:
  - Use smoother losses (e.g., logistic) or early stopping on a validation/OOB proxy.
  - Small learning rate  $\nu$  (e.g., 0.05–0.1) with more iterations.
  - Shallow base trees (stumps to depth 3–5) to control variance.
  - Row/feature subsampling to decorrelate learners.
  - Handle class imbalance via class weights or balanced sampling; evaluate with PR-AUC.

# Typical behavior

- **Exponential loss goes strictly down.**
- **Training error of  $H$  goes down.**
- Weighted error  $\epsilon_m$  goes **up**  $\implies$  share of votes  $\alpha_m$  goes **down**.
- **Test error decreases** even after a flat training error.



# Bagging vs. Boosting

	Bagging	Boosting
Training Strategy	Parallel training	Sequential training
Data Sampling	Bootstrapping (random subsets)	Weighted (by instance importance)
Learners Dependency	Independent	Dependent (on the previous models)
Learner Weighting	Equal weights	Varying weights (based on importance)
Tolerance to Noise	More robust (due to aggregation)	More sensitive (may overfit to noise)
Properties	Reduces variance	Reduces bias and variance (focus on bias)

- 1 Introduction
- 2 Bagging
- 3 Boosting
- 4 AdaBoost
- 5 Comparison
- 6 References

# Contributions

- **This slide has been prepared thanks to:**
  - Nikan Vasei
  - Mahan Bayhaghi
  - Aida Jalali

- [1] C. M., *Pattern Recognition and Machine Learning*.  
Information Science and Statistics, New York, NY: Springer, 1 ed., Aug. 2006.
- [2] M. Soleymani Baghshah, “Machine learning.” Lecture slides.
- [3] R. E. Schapire, “The boosting approach to machine learning: An overview,”  
*Nonlinear estimation and classification*, pp. 149–171, 2003.
- [4] L. Serrano, *Grokking machine learning*.  
New York, NY: Manning Publications, Jan. 2022.
- [5] T. Mitchell, *Machine Learning*.  
McGraw-Hill series in computer science, New York, NY: McGraw-Hill Professional,  
Mar. 1997.
- [6] T. Jaakkola, “Machine learning course slides.” Lecture slides.