Machine Learning (CE 40717)

Fall 2024

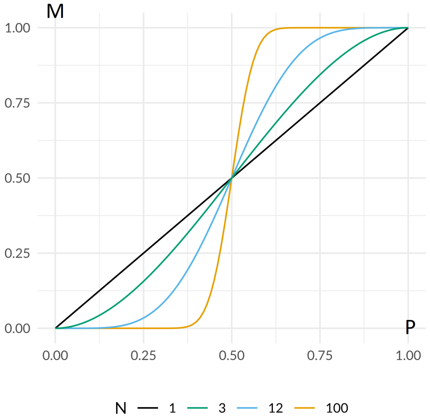
Ali Sharifi-Zarchi

CE Department
Sharif University of Technology

September 19, 2024

Condorcet's jury theorem

- N voters wish to reach a decision by **majority vote**.
- Each voter has an independent probability p of voting for the correct decision.
- Majority votes for the correct decision with probability M.
- If $p > 0.5$ and $N \to \infty$, then $M \to 1$
  - How?



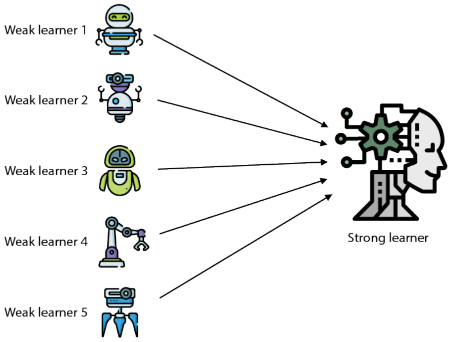
Adopted from Wikipedia

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).

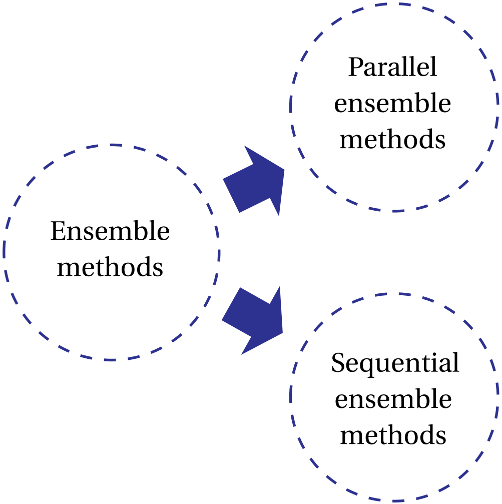
Basic idea

- 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**.



Adopted from [4]

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

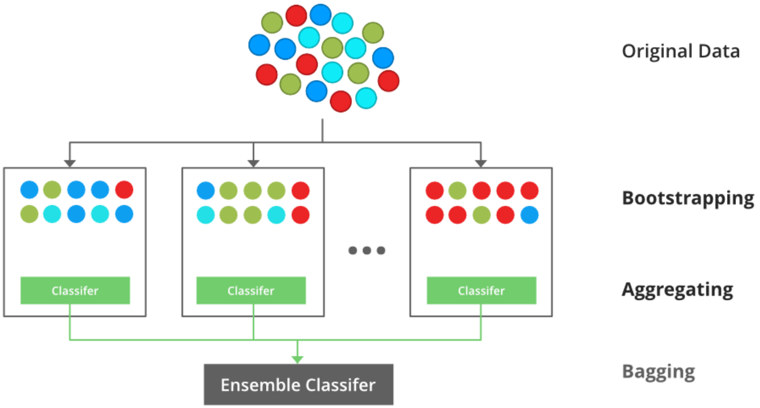
- Weak or simple learners
  - **Low variance**: they don't usually overfit
  - **High bias**: they can't learn complex functions
- **Bagging** (parallel): To decrease the variance
  - Random Forest
- **Boosting** (sequential): To decrease the bias (enhance their capabilities)
  - AdaBoost

1. Introduction
2. Bagging
   - Basic idea & algorithm
   - Decision tree (quick review)
3. Random Forest
4. Boosting
5. AdaBoost
6. Comparison
7. References

Basic idea

- **Bagging** = **B**ootstrap **agg**regat**ing**
- 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.



Adopted from GeeksForGeeks

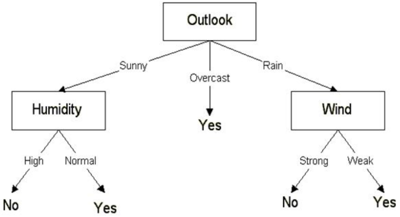
Algorithm

Algorithm 1 Bagging

1: **Input:** M (required ensemble size), $D = \{(\boldsymbol{x}^{(1)}, y^{(1)}), \ldots, (\boldsymbol{x}^{(N)}, y^{(N)})\}$ (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 N-face dice N times*
4: Train a model h_t using D_t and add it to the ensemble
5: **end for**
6: $H(x) = \text{sign}\left(\sum_{t=1}^{M} h_t(\boldsymbol{x})\right)$
▷ *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

Adopted 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.

Algorithm

Algorithm 2 Constructing DT

procedure FINDTREE(S, A) ▷ Input: S (samples), A (attributes)
    if A is empty **or** all labels in S are the same **then**
        status ← leaf
        class ← most common class in S
    else
        status ← internal
        $a \leftarrow$ bestAttribute(S, A) ▷ The attribute value test
        LeftNode ← FindTree($S(a = 1)$, $A \setminus \{a\}$)
        RightNode ← FindTree($S(a = 0)$, $A \setminus \{a\}$)
    end if
end procedure

Which attribute is the best?

- **Entropy** measures the uncertainty in a specific distribution.

$$H(X) = -\sum_{\boldsymbol{x}_i \in \boldsymbol{x}} P(\boldsymbol{x}_i) \log P(\boldsymbol{x}_i)$$

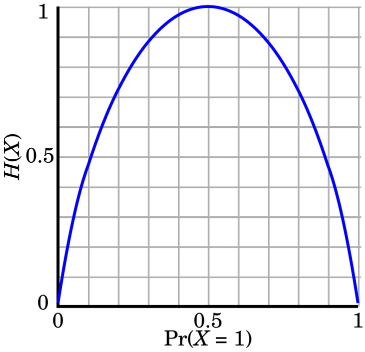
- **Information Gain (IG)**

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

A: variable used to split samples

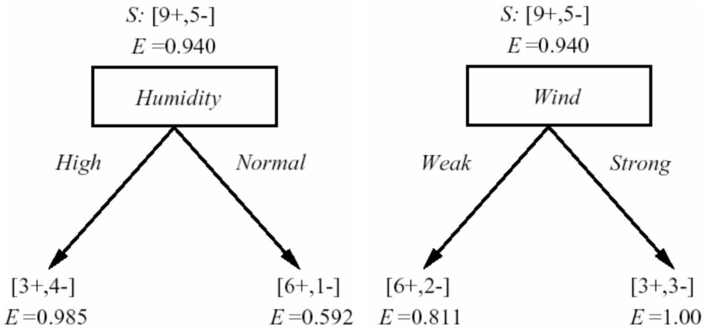
Y: target variable

S: samples



Adopted from Wikipedia

Example



Adopted from [5]

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

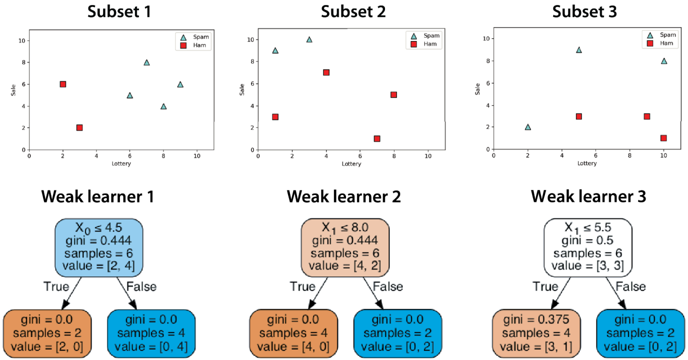
$$\text{Gain}(S, \textit{Humidity}) = 0.940 - (8/14)0.811 - (6/14)1.0 = 0.48$$

1. Introduction
2. Bagging
3. **Random Forest**
4. Boosting
5. AdaBoost
6. Comparison
7. References

Bagging on decision trees?

Why decision trees?

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



Adopted from [4]

Perfect candidates

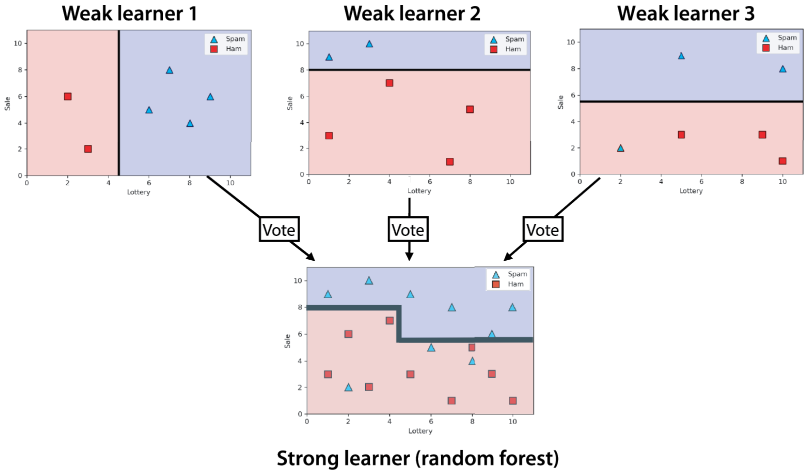
- Why are **DTs** perfect candidates for ensembles?
  - Consider averaging many (nearly) **unbiased** tree estimators.
  - Bias remains similar, but **variance is reduced**.
- Remember Bagging?
  - Train many trees on bootstrapped data, then average the outputs.

Algorithm

Algorithm 3 Random Forest

Input: T (number of trees), m (number of variables used to split each node)
for $t = 1$ to T **do**
    Draw a bootstrap dataset
    Learn a tree on this dataset
    ▷ *Select* $\boldsymbol{m}$ *features randomly out of* $\boldsymbol{d}$ *features as candidates before splitting*
end for
Output: ▷ *Usually:* $m \leq \sqrt{d}$
    Regression: average of the outputs
    Classification: majority voting

Example



Adopted from [4]

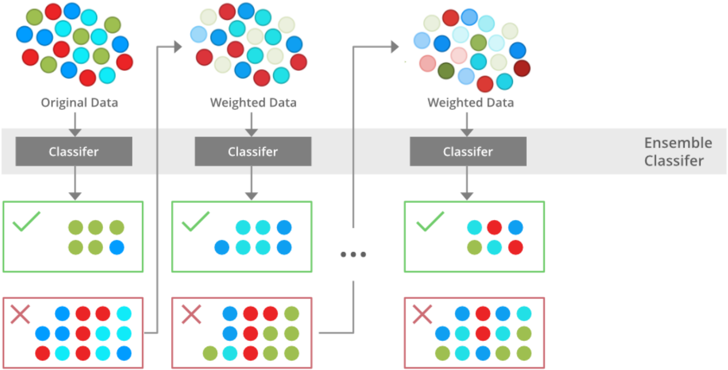
Problems with bagging

- 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 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.



Adopted from GeeksForGeeks

Algorithm

- Try to combine many simple **weak** classifiers (in sequence) to find a single **strong** classifier.
  - Each component is a simple binary ± 1 classifier
  - Voted combinations of component classifiers

$$H_m(\boldsymbol{x}) = \alpha_1 h(\boldsymbol{x}; \boldsymbol{\theta}_1) + \cdots + \alpha_m h(\boldsymbol{x}; \boldsymbol{\theta}_m)$$

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

$$H_m(\boldsymbol{x}) = \alpha_1 h_1(\boldsymbol{x}) + \cdots + \alpha_m h_m(\boldsymbol{x})$$

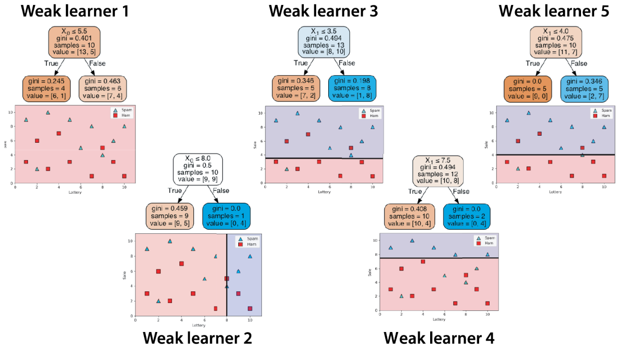
- **Prediction**: $\hat{y} = \text{sign}(H_m(\boldsymbol{x}))$

Candidate for $h_i(x)$

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

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

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

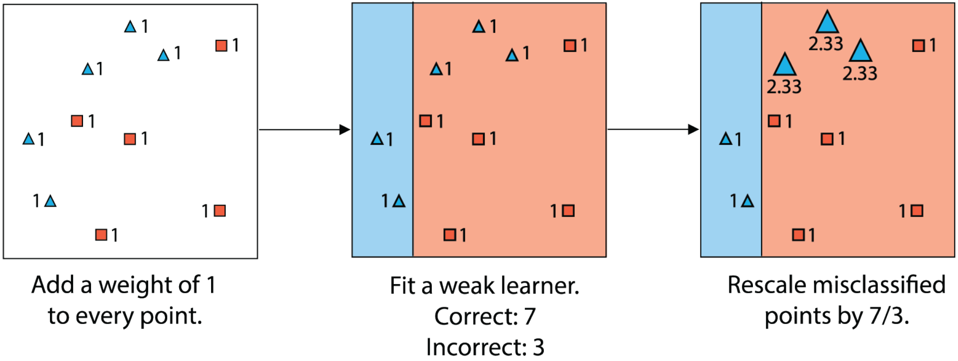


Adopted from [4]

Basic idea

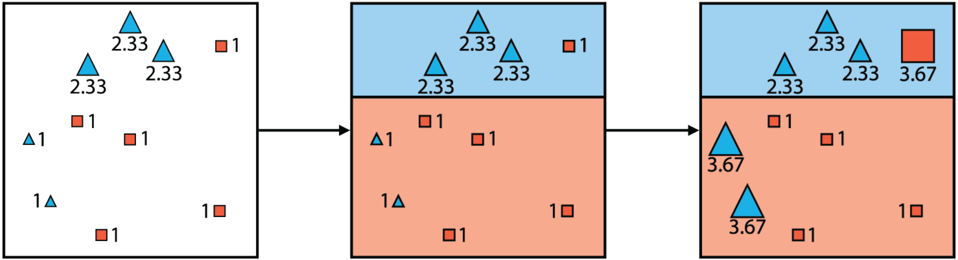
- **Sequential** production of classifiers
  - Iteratively add the classifier whose addition will be most helpful.
- Represent the important of each sample by assigning **weights** to them.
  - Correct classification $\implies$ smaller weights
  - Misclassified samples $\implies$ larger weights
- Each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

Example



Adopted from [4]

Example, Cont.



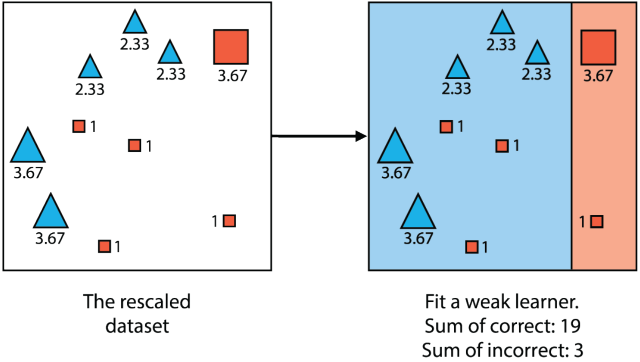
The rescaled dataset

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

Rescale misclassified points by 11/3.

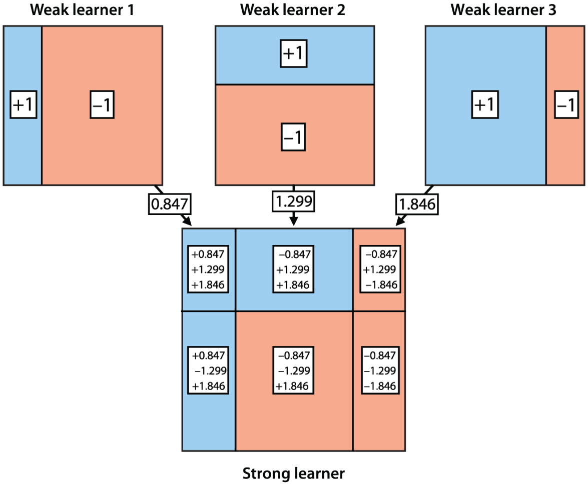
Adopted from [4]

Example, Cont.



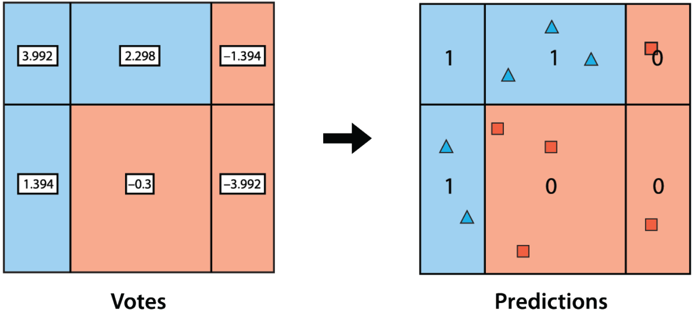
Adopted from [4]

Example, Cont.



Adopted from [4]

Example, Cont.



Adopted from [4]

Algorithm

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: $J_m = \sum_{i=1}^{N} w_m^{(i)} \times I(y^{(i)} \neq h_m(\boldsymbol{x}^{(i)}))$ ▷ Find $h_m(\boldsymbol{x})$ by minimizing the weighted error
4: $\epsilon_m = \dfrac{\sum_{i=1}^{N} w_m^{(i)} \times I(y^{(i)} \neq h_m(\boldsymbol{x}^{(i)}))}{\sum_{i=1}^{N} w_m^{(i)}}$ ▷ Find the weighted error of $h_m(\boldsymbol{x})$
5: $\alpha_m = \ln\left(\dfrac{1-\epsilon_m}{\epsilon_m}\right)$ ▷ Assign votes based on the error
6: $w_{m+1}^{(i)} = w_m^{(i)} e^{\alpha_m I(y^{(i)} \neq h_m(\boldsymbol{x}^{(i)}))}$ ▷ Update normalized weights
7: **end for**
8: **Combined classifier:** $\hat{y} = \text{sign}(H_M(\boldsymbol{x}))$, $H_M(\boldsymbol{x}) = \sum_{m=1}^{M} \alpha_m h_m(\boldsymbol{x})$

Notations & conditions

- $w_m^{(i)}$: weighting coefficient of data point i in iteration m
- α_m: weighting coefficient of m-th base classifier in the final ensemble
  - ϵ_m: weighted error rate of m-th classifier

- Only when $h_m(\boldsymbol{x})$ with $\epsilon_m < 0.5$ (better than chance) is found, boosting continues.
  - Condorcet's jury theorem?

Loss function

- We need a loss function for the combination
  - To Determine the new component, $h(\boldsymbol{x};\theta)$
  - And how many votes it should receive, α

$$H_m(\boldsymbol{x}) = \alpha_1 h_1(\boldsymbol{x}) + \cdots + \alpha_m h_m(\boldsymbol{x})$$

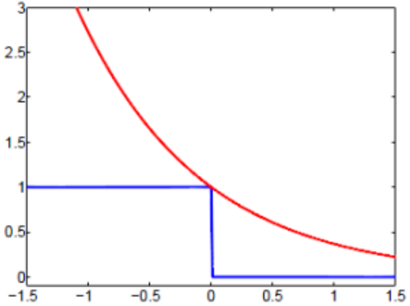
- Many options for the loss function
  - AdaBoost is equivalent to using the following **exponential loss**

$$\text{Loss}(y, \hat{y}) = e^{-y \times H_m(\boldsymbol{x})}$$

$$\hat{y} = \text{sign}(H_m(\boldsymbol{x}))$$

Why the exponential loss?

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



Adopted from [2]

Calculations

- Consider adding the m-th component: $$H_m(\boldsymbol{x}) = \underset{\substack{\uparrow \\ \text{For a cleaner form later}}}{\frac{1}{2}}[\alpha_1 h_1(\boldsymbol{x}) + \ldots, +\alpha_m h_m(\boldsymbol{x})]$$

$$E = \sum_{i=1}^{N} e^{-y^{(i)} H_m(\boldsymbol{x}^{(i)})} = \sum_{i=1}^{N} e^{-y^{(i)}[H_{m-1}(\boldsymbol{x}^{(i)}) + \frac{1}{2}\alpha_m h_m(\boldsymbol{x}^{(i)})]}$$

$$= \sum_{i=1}^{N} \underset{\substack{\uparrow \\ \text{Suppose it is fixed at stage } m}}{e^{-y^{(i)} H_{m-1}(\boldsymbol{x}^{(i)})}} e^{-\frac{1}{2}\alpha_m y^{(i)} h_m(\boldsymbol{x}^{(i)})} = \sum_{i=1}^{N} \underbrace{w_m^{(i)}}_{e^{-y^{(i)} H_{m-1}(\boldsymbol{x}^{(i)})}} \underset{\substack{\uparrow \\ \text{Should be optimized at stage } m \text{ by seeking } h_m(\boldsymbol{x}) \text{ and } \alpha_m}}{e^{-\frac{1}{2}\alpha_m y^{(i)} h_m(\boldsymbol{x}^{(i)})}}$$

Weighted exponential loss

$$E = \sum_{i=1}^{N} w_m^{(i)} e^{-\frac{1}{2}\alpha_m y^{(i)} h_m(\boldsymbol{x}^{(i)})}$$

- Sequentially adds a new component trained on reweighted training samples.
- $w_m^{(i)}$: history of classification of $\boldsymbol{x}^{(i)}$ by H_{m-1}
  - Loss weighted towards mistakes
- Iteration m optimization:
  - Choose the new component, $h_m = h(\boldsymbol{x}; \boldsymbol{\theta}_m)$
  - And the vote that optimizes the weighted exponential loss, α_m.

Minimizing loss: finding h_m

$$E = \sum_{i=1}^{N} w_m^{(i)} e^{-\frac{1}{2}\alpha_m y^{(i)} h_m(\boldsymbol{x}^{(i)})}$$

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

$$= (e^{\frac{\alpha_m}{2}} - e^{\frac{-\alpha_m}{2}}) \underbrace{\left(\sum_{y^{(i)}\neq h_m(\boldsymbol{x}^{(i)})} w_m^{(i)} \right)}_{J_m = \sum_{i=1}^{N} w_m^{(i)} \times I\left(y^{(i)} \neq h_m(\boldsymbol{x}^{(i)})\right)} + e^{\frac{-\alpha_m}{2}} \left(\sum_{i=1}^{N} w_m^{(i)} \right)$$

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

Minimizing loss: finding α_m

$$\frac{\partial E}{\partial \alpha_m} = 0$$

$$\Longrightarrow \frac{1}{2}\left(e^{\frac{\alpha_m}{2}} + e^{\frac{-\alpha_m}{2}}\right)\left(\sum_{y^{(i)} \neq h_m(\boldsymbol{x}^{(i)})} w_m^{(i)}\right) - \frac{1}{2} e^{\frac{-\alpha_m}{2}} \left(\sum_{i=1}^{N} w_m^{(i)}\right) = 0$$

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

$$\epsilon_m = \frac{\sum_{i=1}^{N} w_m^{(i)} I\left(y^{(i)} \neq h_m(\boldsymbol{x}^{(i)})\right)}{\sum_{i=1}^{N} w_m^{(i)}}, \qquad \alpha_m = \ln\left(\frac{1-\epsilon_m}{\epsilon_m}\right)$$

Updating weights

- Updating weights in AdaBoost algorithm:

$$w_i^{m+1} = w_i^m e^{-\frac{1}{2}\alpha_m y^{(i)} h_m(\boldsymbol{x}^{(i)})}$$

$$\xRightarrow{y^{(i)} h_m(\boldsymbol{x}^{(i)}) = 1 - 2I(y^{(i)} \neq h_m(\boldsymbol{x}^{(i)}))} w_i^{m+1} = w_i^m \underbrace{e^{-\frac{1}{2}\alpha_m}}_{\text{Independent of } i \text{ and can be ignored}} e^{\alpha_m I(y^{(i)} \neq h_m(\boldsymbol{x}^{(i)}))}$$

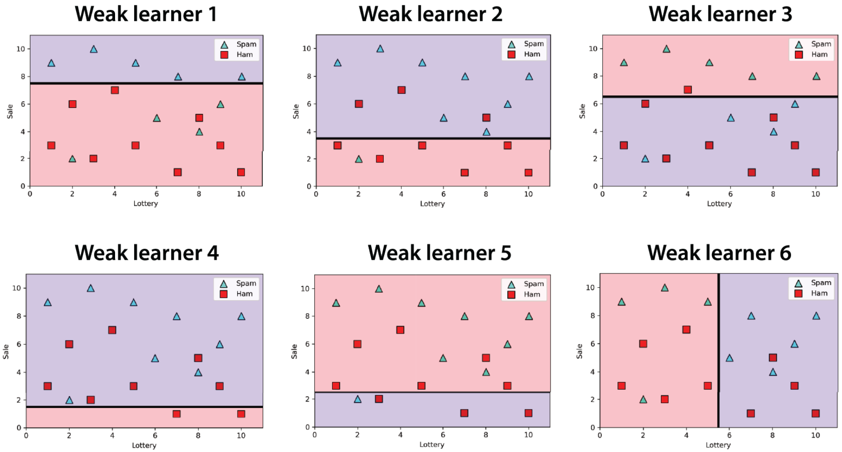
$$\Longrightarrow w_i^{m+1} = w_i^m e^{\alpha_m I(y^{(i)} \neq h_m(\boldsymbol{x}^{(i)}))}$$

Summary

Algorithm 5 AdaBoost Summary

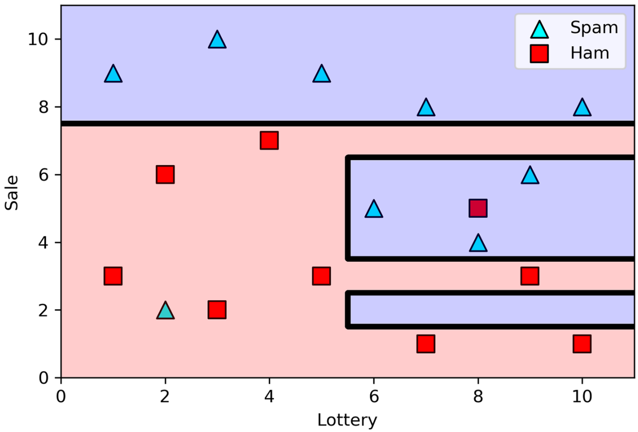
for $i = 1$ to N **do**
  Initialize the data weight $w_1^{(i)} = \frac{1}{N}$
end for
for $m = 1$ to M **do**
  Find a classifier $h_m(\boldsymbol{x})$ by minimzing the weighted error function
  Find the normalized weighted error of $h_m(\boldsymbol{x})$ as ϵ_m
  Compute the new component weight as α_m
  Update example weights for the next iteration $w_{m+1}^{(i)}$
end for
Combined classifier $\hat{y} = \text{sign}(H_M(\boldsymbol{x}))$ where $H_M(\boldsymbol{x}) = \sum_{m=1}^{M} \alpha_m h_m(\boldsymbol{x})$

Example



Adopted from [4]

Example, Cont.



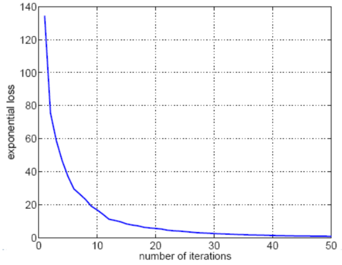
Adopted from [4]

Exponential loss properties

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

$$H_m(x) = \frac{1}{2}[\hat{\alpha}_1 h(\boldsymbol{x}; \hat{\boldsymbol{\theta}}_1) + \cdots + \hat{\alpha}_m h(\boldsymbol{x}; \hat{\boldsymbol{\theta}}_m)]$$

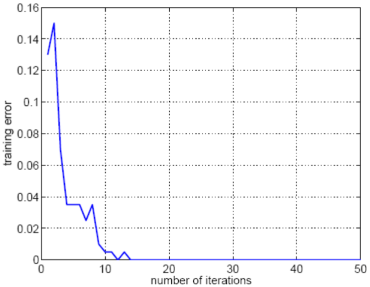
- Thus, **lower exponential loss** over training data is guaranteed.



Adopted from [6]

Training error properties

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

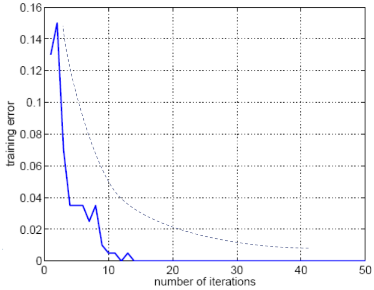


Adopted from [6]

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., $\epsilon_m < 0.5$)

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

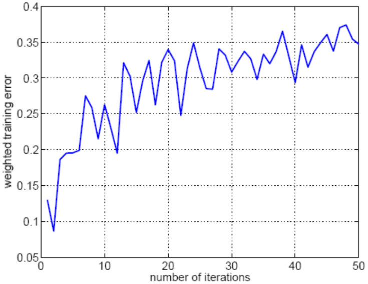


Adopted from [6]

Weighted error properties

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

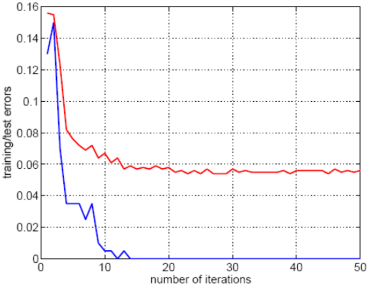
$$\epsilon_m = \frac{\sum_{i=1}^{N} w_m^{(i)} I\left(y^{(i)} \neq h_m(\boldsymbol{x}^{(i)})\right)}{\sum_{i=1}^{N} w_m^{(i)}}$$



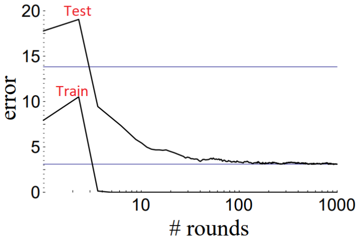
Adopted from [6]

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.



Adopted from [6]



Adopted from [3]

Typical behavior

- **Exponential loss** goes **strictly down**.
- **Training error** of H goes **down**.
- Weighted error $\boldsymbol{\epsilon_m}$ goes **up** $\implies$ votes $\boldsymbol{\alpha_m}$ go **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 bias	Reduces bias and variance (focus on bias)

1. Introduction
2. Bagging
3. Random Forest
4. Boosting
5. AdaBoost
6. Comparison
7. **References**

[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.