

1 Introduction

2 Bagging

3 Random Forest

4 Boosting

5 AdaBoost

6 Comparison

7 References

1 Introduction

Condorcet's jury theorem

Ensemble learning

Ensemble methods

2 Bagging

③ Random Forest

4 Boosting

5 AdaBoost

⑥ Comparison

7 References

1 Introduction

Condorcet's jury theorem

Ensemble learning

Ensemble methods

2 Bagging

③ Random Forest

4 Boosting

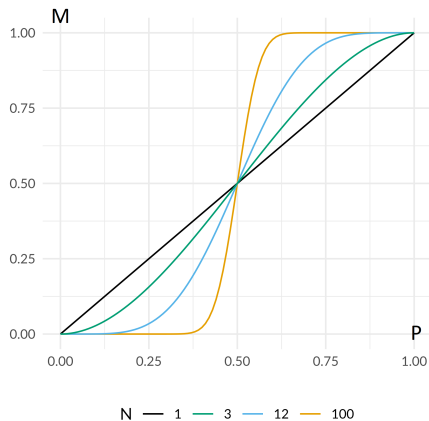
5 AdaBoost

⑥ Comparison

7 References

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 $N \rightarrow \infty$, then $M \rightarrow 1$
 - How?



Adopted from Wikipedia

1 Introduction

Condorcet's jury theorem

Ensemble learning

Ensemble methods

2 Bagging

③ Random Forest

4 Boosting

5 AdaBoost

6 Comparison

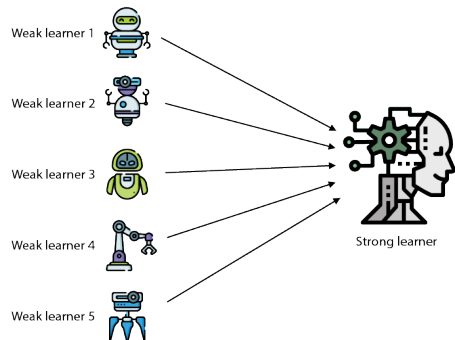
7 References

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]

1 Introduction

Condorcet's jury theorem

Ensemble learning

Ensemble methods

2 Bagging

③ Random Forest

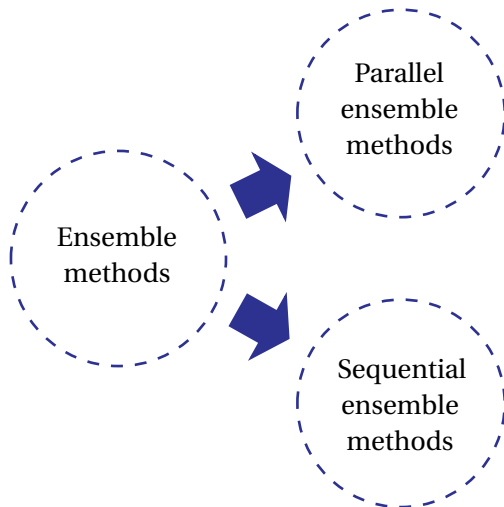
4 Boosting

5 AdaBoost

6 Comparison

7 References

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

1 Introduction

2 Bagging

Basic idea & algorithm

Decision tree (quick review)

③ Random Forest

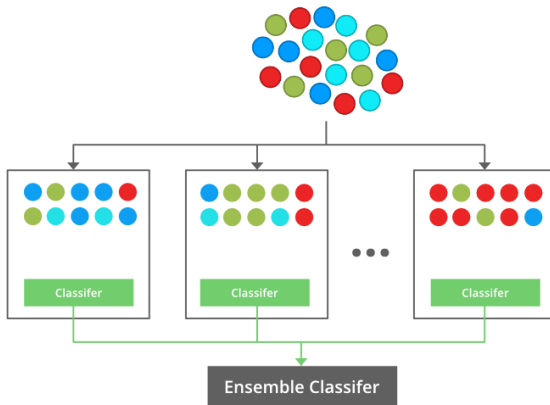
4 Boosting

5 AdaBoost

⑥ Comparison

7 References

Basic idea, Cont.



Original Data

Bootstrapping

Aggregating

Bagging

Adopted 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 N -face dice N times*
- 4: Train a model h_t using D_t and add it to the ensemble
- 5: **end for**
- 6: $H(x) = \text{sign}(\sum_{t=1}^M h_t(\mathbf{x}))$
 ▷ *Aggregate models by voting for classification or by averaging for regression*

1 Introduction

2 Bagging

Basic idea & algorithm

Decision tree (quick review)

③ Random Forest

4 Boosting

5 AdaBoost

⑥ Comparison

7 References

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 2 Constructing DT

```

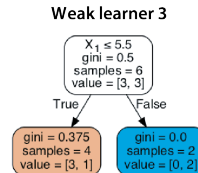
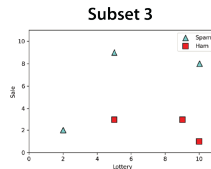
1: procedure FINDTREE( $S, A$ ) ▷ Input:  $S$  (samples),  $A$  (attributes)
2:   if  $A$  is empty or all labels in  $S$  are the same then
3:     status  $\leftarrow$  leaf
4:     class  $\leftarrow$  most common class in  $S$ 
5:   else
6:     status  $\leftarrow$  internal
7:      $a \leftarrow$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

```


- A set of navigation icons typically found in Beamer presentations, including symbols for back, forward, search, and other slide controls.

Why decision trees?

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

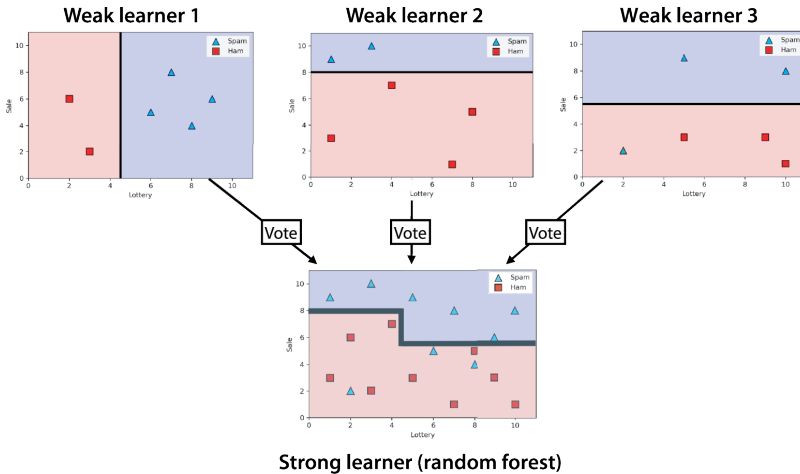


Adopted from [4]

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.

Example

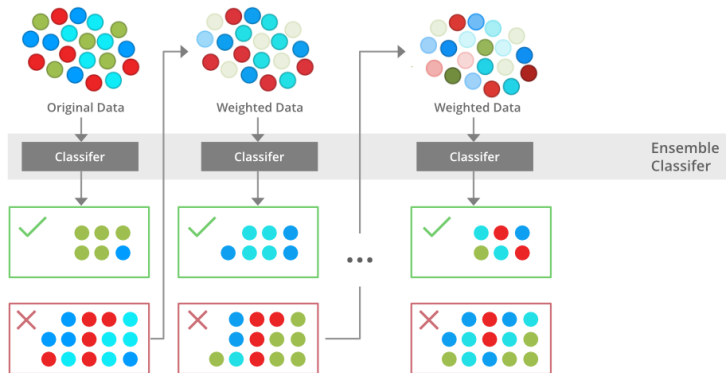


Adopted from [4]

7 References

- 1 Introduction
- 2 Bagging
- 3 Random Forest
- 4 Boosting**
 - Motivation & basic idea
 - Algorithm
- 5 AdaBoost
- 6 Comparison
- 7 References

Basic idea, Cont.



Adopted from GeeksForGeeks

- ## 7 References

35 / 70

- Basic idea & example
- Algorithm
- Loss function
- Summary & example
- Properties

6 Comparison

1 Introduction

2 Bagging

③ Random Forest

4 Boosting

5 AdaBoost

Basic idea & example

Algorithm

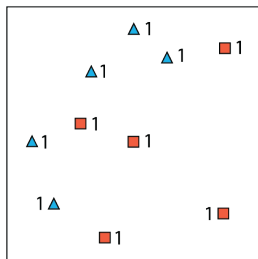
Loss function

Summary & example

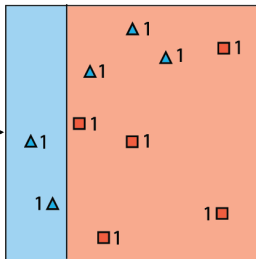
Properties

6 Comparison

Example

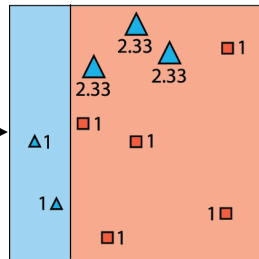


Add a weight of 1 to every point.



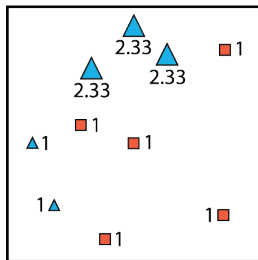
Fit a weak learner.
Correct: 7
Incorrect: 3

Adopted from [4]

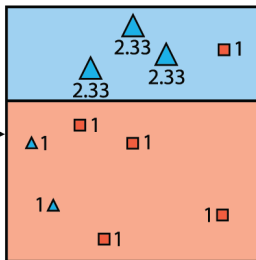


Rescale misclassified points by $7/3$.

Example, Cont.

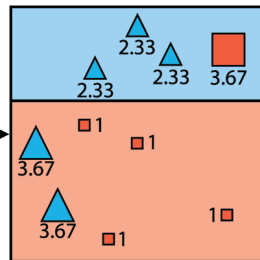


The rescaled dataset



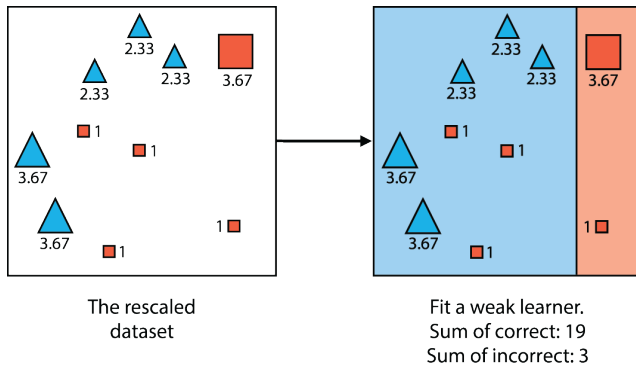
Fit a weak learner.
Sum of correct: 11
Sum of incorrect: 3

Adopted from [4]



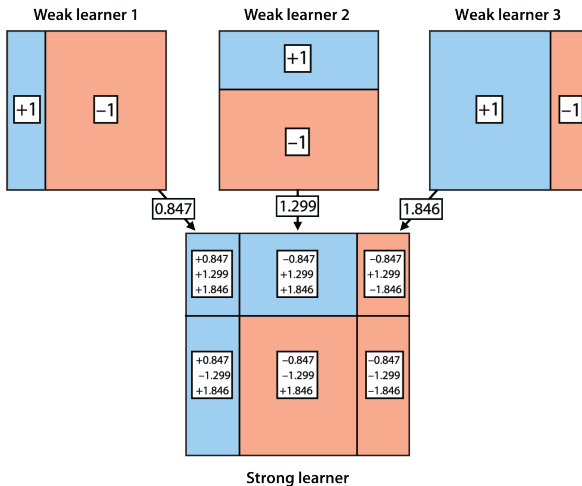
Rescale misclassified points by $11/3$.

Example, Cont.



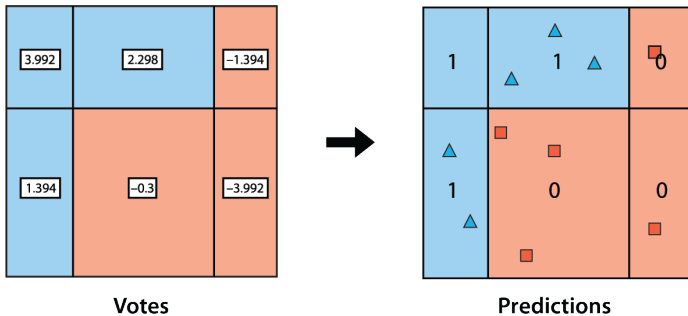
Adopted from [4]

Example, Cont.



Adopted from [4]

Example, Cont.



Adopted from [4]

1 Introduction

2 Bagging

3 Random Forest

4 Boosting

5 AdaBoost

Basic idea & example

Algorithm

Loss function

Summary & example

Properties

6 Comparison

- ## Algorithm

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

1 Introduction

2 Bagging

3 Random Forest

4 Boosting

5 AdaBoost

Basic idea & example

Algorithm

Loss function

Summary & example

Properties

6 Comparison

Loss function

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

$$H_m(\mathbf{x}) = \alpha_1 h_1(\mathbf{x}) + \dots + \alpha_m h_m(\mathbf{x})$$

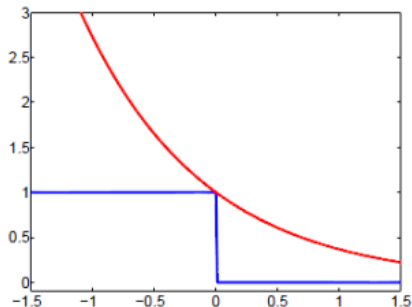
- Many options for the loss function
 - AdaBoost is equivalent to using the following **exponential loss**

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

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

Why the exponential loss?

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



Adopted from [2]

Calculations

- Consider adding the m -th component:

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

↑

For a cleaner form later

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

Suppose it is fixed at stage m

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

Weighted exponential loss

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

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

Minimizing loss: finding h_m

$$\begin{aligned}
 E &= \sum_{i=1}^N w_m^{(i)} e^{-\frac{1}{2} \alpha_m y^{(i)} h_m(\mathbf{x}^{(i)})} \\
 &= e^{\frac{-\alpha_m}{2}} \left(\sum_{y^{(i)} = h_m(\mathbf{x}^{(i)})} w_m^{(i)} \right) + e^{\frac{\alpha_m}{2}} \left(\sum_{y^{(i)} \neq h_m(\mathbf{x}^{(i)})} w_m^{(i)} \right) \\
 &= (e^{\frac{\alpha_m}{2}} - e^{\frac{-\alpha_m}{2}}) \underbrace{\left(\sum_{y^{(i)} \neq h_m(\mathbf{x}^{(i)})} w_m^{(i)} \right)}_{J_m} + e^{\frac{-\alpha_m}{2}} \left(\sum_{i=1}^N w_m^{(i)} \right)
 \end{aligned}$$

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

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

Minimizing loss: finding α_m

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

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

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

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

Updating weights

- Updating weights in AdaBoost algorithm:

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

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

↑

Independent of i and can be ignored

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

1 Introduction

2 Bagging

3 Random Forest

4 Boosting

5 AdaBoost

Basic idea & example

Algorithm

Loss function

Summary & example

Properties

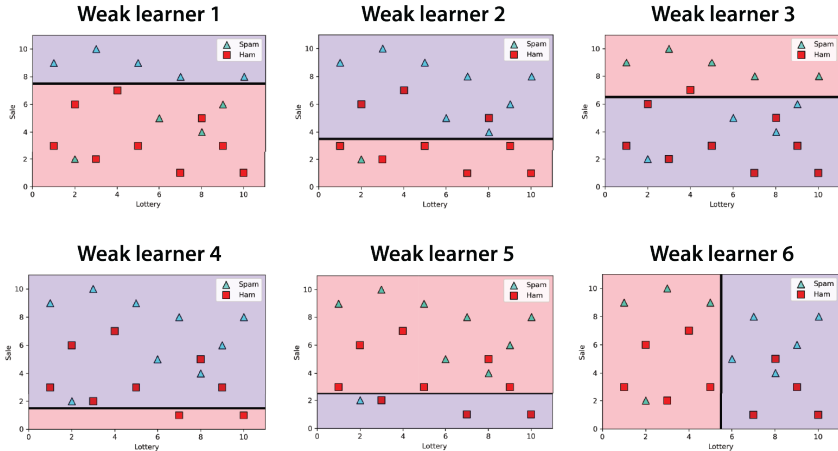
6 Comparison

Summary

Algorithm 5 AdaBoost Summary

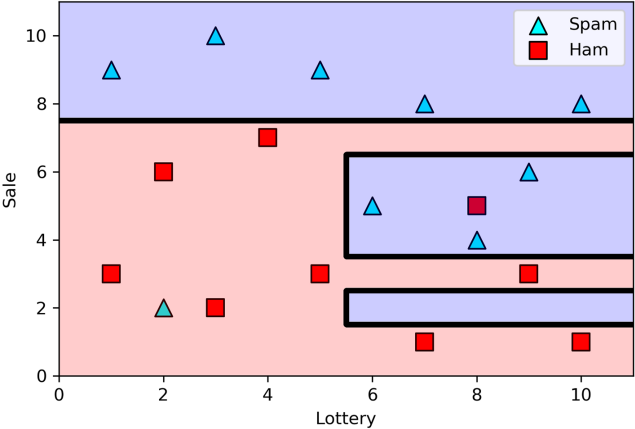
- 1: **for** $i = 1$ to N **do**
 - 2: Initialize the data weight $w_1^{(i)} = \frac{1}{N}$
 - 3: **end for**
 - 4: **for** $m = 1$ to M **do**
 - 5: Find a classifier $h_m(\mathbf{x})$ by minimizing the weighted error function
 - 6: Find the normalized weighted error of $h_m(\mathbf{x})$ as ϵ_m
 - 7: Compute the new component weight as α_m
 - 8: Update example weights for the next iteration $w_{m+1}^{(i)}$
 - 9: **end for**
 - 10: Combined classifier $\hat{y} = \text{sign}(H_M(\mathbf{x}))$ where $H_M(\mathbf{x}) = \sum_{m=1}^M \alpha_m h_m(\mathbf{x})$
-

Example



Adopted from [4]

Example, Cont.



Adopted from [4]

1 Introduction

2 Bagging

3 Random Forest

4 Boosting

5 AdaBoost

Basic idea & example

Algorithm

Loss function

Summary & example

Properties

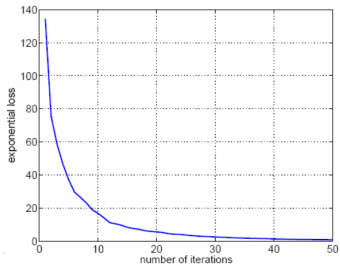
6 Comparison

Exponential loss properties

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

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

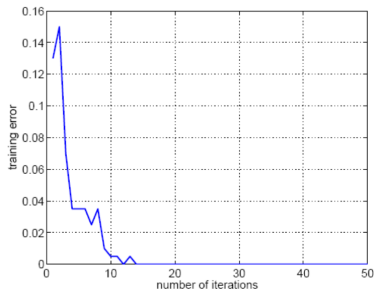
- Thus, **lower exponential loss** over training data is guaranteed.



Adopted from [6]

Training error properties

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

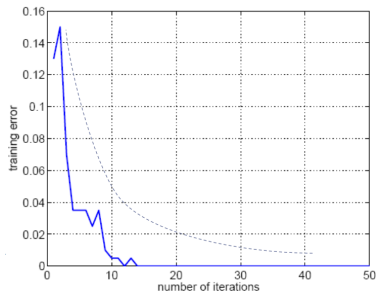


Adopted from [6]

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., $\epsilon_m < 0.5$)

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

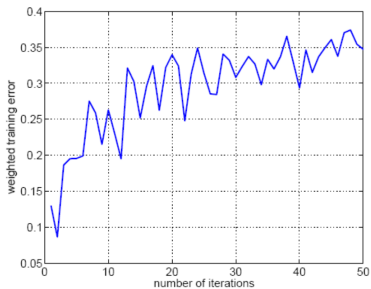


Adopted from [6]

Weighted error properties

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

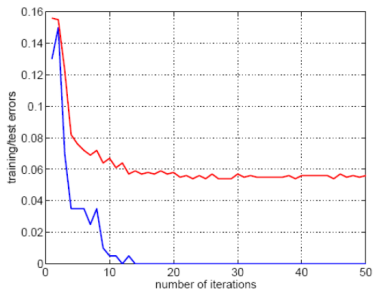
$$\epsilon_m = \frac{\sum_{i=1}^N w_m^{(i)} I(y^{(i)} \neq h_m(\mathbf{x}^{(i)}))}{\sum_{i=1}^N w_m^{(i)}}$$



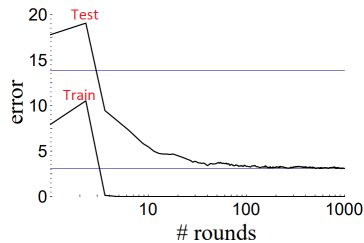
Adopted from [6]

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 ϵ_m goes **up** \implies votes α_m go **down**.
- **Test error decreases** even after a flat training error.

- A set of navigation icons typically found in Beamer presentations, including symbols for back, forward, search, and other slide controls.

- A set of navigation icons typically found in Beamer presentations, including symbols for back, forward, search, and other slide controls.

Contributions

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

- [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.