Boosting and Ensembles

Machine Learning

Some slides based on materials by Yoav Freund, Rob Schapire, Dan Roth, Tommi Jakkola and others

Boosting and Ensembles

- What is boosting?
- AdaBoost
- Ensemble methods

Boosting and Ensembles

- What is boosting?
- AdaBoost
- Ensemble methods

Boosting

A general learning approach for constructing a *strong learner*, given a collection of (possibly infinite) weak learners

Historically: An answer to a question in the context of the PAC theory

The Strength of Weak Learnability

ROBERT E. SCHAPIRE 1989-90

Practically useful

Boosting is a way to create a strong learner using only weak learners (also known as “rules of thumb”)

An *Ensemble method*

- A class of learning algorithms that composes classifiers using other classifiers as building blocks
- Boosting has stronger theoretical guarantees than other ensemble methods

Example: How may I help you?

Goal: Automatically categorize type of phone call requested by a phone customer

"I'd like to know my account balance please" → `Balances`

"When do you open on Monday?" → `Hours`

"I am unable to login to my account on the app" → `OnlineServices`

Example: How may I help you?

Goal: Automatically categorize type of phone call requested by a phone customer

"I'd like to know my account balance please" → `Balances`

"When do you open on Monday?" → `Hours`

"I am unable to login to my account on the app" → `OnlineServices`

Important observation

- *Rules of thumb* are often correct
  - Eg: If *login* occurs in the utterance, then predit `OnlineServices`
- But hard to find a single prediction rule that covers all cases

One boosting approach

- Select a small subset of examples
- Derive a rough rule of thumb

- Sample a second subset of examples
- Derive a second rule of thumb

- Repeat T times…

- Combine rules of thumb into a single prediction rule

One boosting approach

- Select a small subset of examples
- Derive a rough rule of thumb

- Sample a second subset of examples
- Derive a second rule of thumb

- Repeat T times…

- Combine rules of thumb into a single prediction rule

Need to specify:
1. How to select these subsets?
2. How to combine these rules of thumb?

One boosting approach

- Select a small subset of examples
- Derive a rough rule of thumb

- Sample a second subset of examples
- Derive a second rule of thumb

- Repeat T times…

- Combine rules of thumb into a single prediction rule

Need to specify:

1. How to select these subsets?
2. How to combine these rules of thumb?

Boosting: A general method for converting rough rules of thumb into accurate classifiers

Boosting: The formal problem setup

- Strong PAC algorithm

  For any distribution over examples,
  for every $\epsilon > 0$, for every $\delta > 0$,
  given a polynomial number of random examples
  finds a hypothesis with error $\leq \epsilon$ with probability $\geq 1 - \delta$

Boosting: The formal problem setup

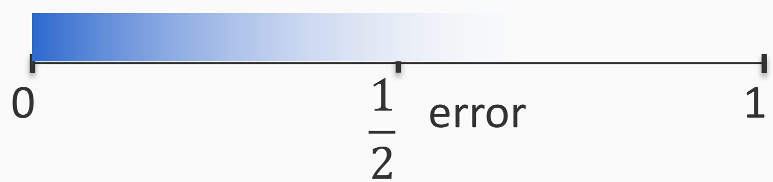
- Strong PAC algorithm

  For any distribution over examples,

  for every $\epsilon > 0$, for every $\delta > 0$,

  given a polynomial number of random examples

  finds a hypothesis with error $\leq \epsilon$ with probability $\geq 1 - \delta$



Boosting: The formal problem setup

- Strong PAC algorithm

  For any distribution over examples,
  for every $\epsilon > 0$, for every $\delta > 0$,
  given a polynomial number of random examples
  finds a hypothesis with error $\leq \epsilon$ with probability $\geq 1 - \delta$

- Weak PAC algorithm
  - Same, but only for $\epsilon > \frac{1}{2} - \gamma$ for some small γ

    E.g. if $\gamma = 0.01$, the error ϵ should be more than $0.5 - 0.01 = 0.49$

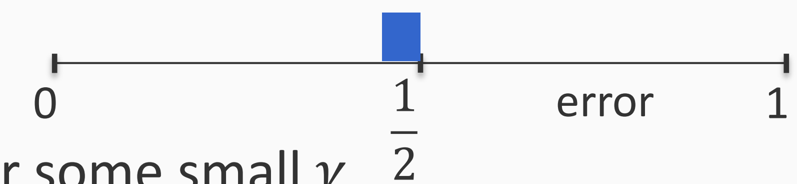
    Assuming that the labels are equally possible, this error is only *slightly* better than chance

Boosting: The formal problem setup

- Strong PAC algorithm

  For any distribution over examples,
  for every $\epsilon > 0$, for every $\delta > 0$,
  given a polynomial number of random examples
  finds a hypothesis with error $\leq \epsilon$ with probability $\geq 1 - \delta$

- Weak PAC algorithm



  – Same, but only for $\epsilon > \frac{1}{2} - \gamma$ for some small γ

E.g. if $\gamma = 0.01$, the error ϵ should be more than $0.5 - 0.01 = 0.49$

Assuming that the labels are equally possible, this error is only *slightly* better than chance

Boosting: The formal problem setup

- Strong PAC algorithm

  For any distribution over examples,
  for every $\epsilon > 0$, for every $\delta > 0$,
  given a polynomial number of random examples
  finds a hypothesis with error $\leq \epsilon$ with probability $\geq 1 - \delta$

- Weak PAC algorithm
  - Same, but only for $\epsilon > \frac{1}{2} - \gamma$ for some small γ

- Question [Kearns and Valiant '88]:
  - Does weak learnability imply strong learnability?

    That is, if we have a weak PAC algorithm for a concept class, is the concept class learnable in the strong sense?

History: Early boosting algorithms

History: Early boosting algorithms

- [Schapire '89]
  - First provable boosting algorithm
  - Call weak learner three times on three modified distributions
  - Get slight boost in accuracy
  - Apply recursively

History: Early boosting algorithms

- [Schapire '89]
  - First provable boosting algorithm
  - Call weak learner three times on three modified distributions
  - Get slight boost in accuracy
  - Apply recursively
- [Freund '90]
  - "Optimal" algorithm that "boosts by majority"

History: Early boosting algorithms

- [Schapire ’89]
  - First provable boosting algorithm
  - Call weak learner three times on three modified distributions
  - Get slight boost in accuracy
  - Apply recursively
- [Freund ’90]
  - “Optimal” algorithm that “boosts by majority”
- [Drucker, Schapire & Simard ’92]
  - First experiments using boosting
  - Limited by practical drawbacks

History: Early boosting algorithms

- [Schapire '89]
  - First provable boosting algorithm
  - Call weak learner three times on three modified distributions
  - Get slight boost in accuracy
  - Apply recursively
- [Freund '90]
  - "Optimal" algorithm that "boosts by majority"
- [Drucker, Schapire & Simard '92]
  - First experiments using boosting
  - Limited by practical drawbacks
- [Freund & Schapire '95]
  - Introduced the *AdaBoost* algorithm
  - Strong practical advantages over previous boosting algorithms

History: Early boosting algorithms

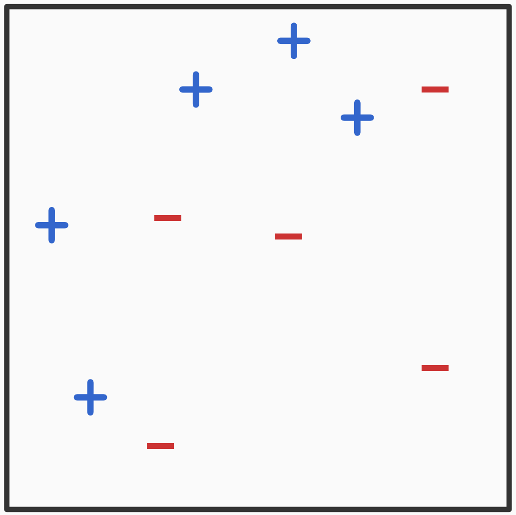
- [Schapire ’89]
  - First provable boosting algorithm
  - Call weak learner three times on three modified distributions
  - Get slight boost in accuracy
  - Apply recursively
- [Freund ’90]
  - “Optimal” algorithm that “boosts by majority”
- [Drucker, Schapire & Simard ’92]
  - First experiments using boosting
  - Limited by practical drawbacks
- [Freund & Schapire ’95]
  - Introduced the *AdaBoost* algorithm
  - Strong practical advantages over previous boosting algorithms

- AdaBoost was followed by a huge number of papers and practical applications
  - And a Gödel prize in 2003 for Freund and Schapire

Boosting and Ensembles

- What is boosting?
- AdaBoost
  - Intuition
  - The algorithm
  - Why does it work
- Ensemble methods

A toy example

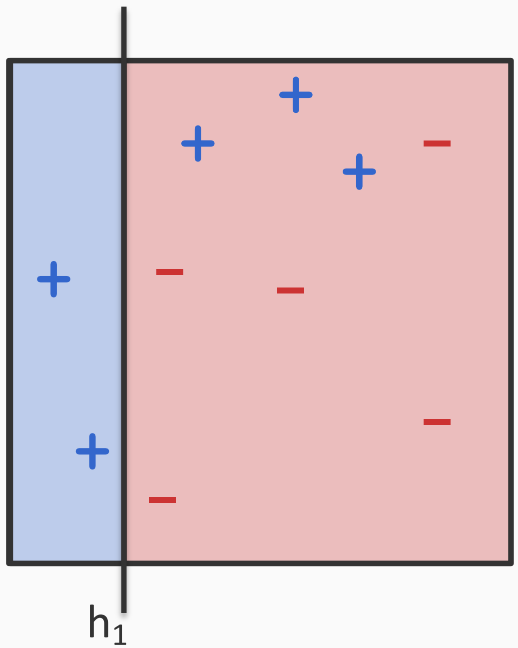


Our weak learner: An axis parallel line

Or

Initially all examples are equally important

A toy example

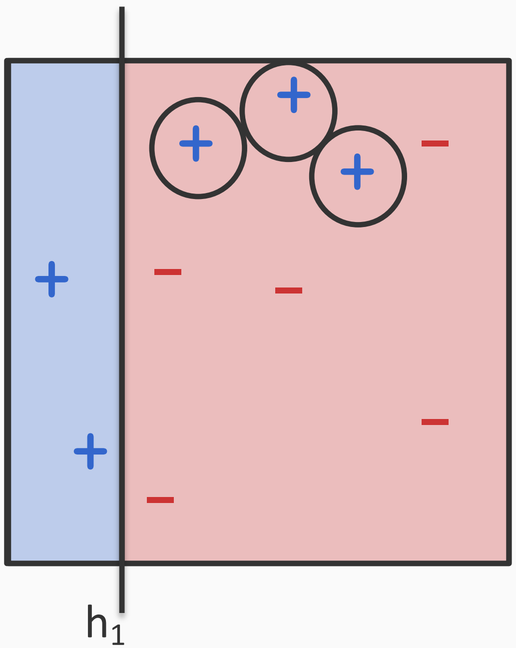


Our weak learner: An axis parallel line

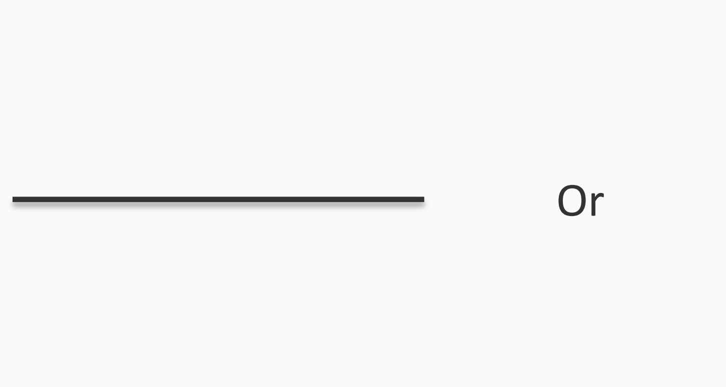
Initially all examples are equally important

h_1 = The best classifier on this data

A toy example



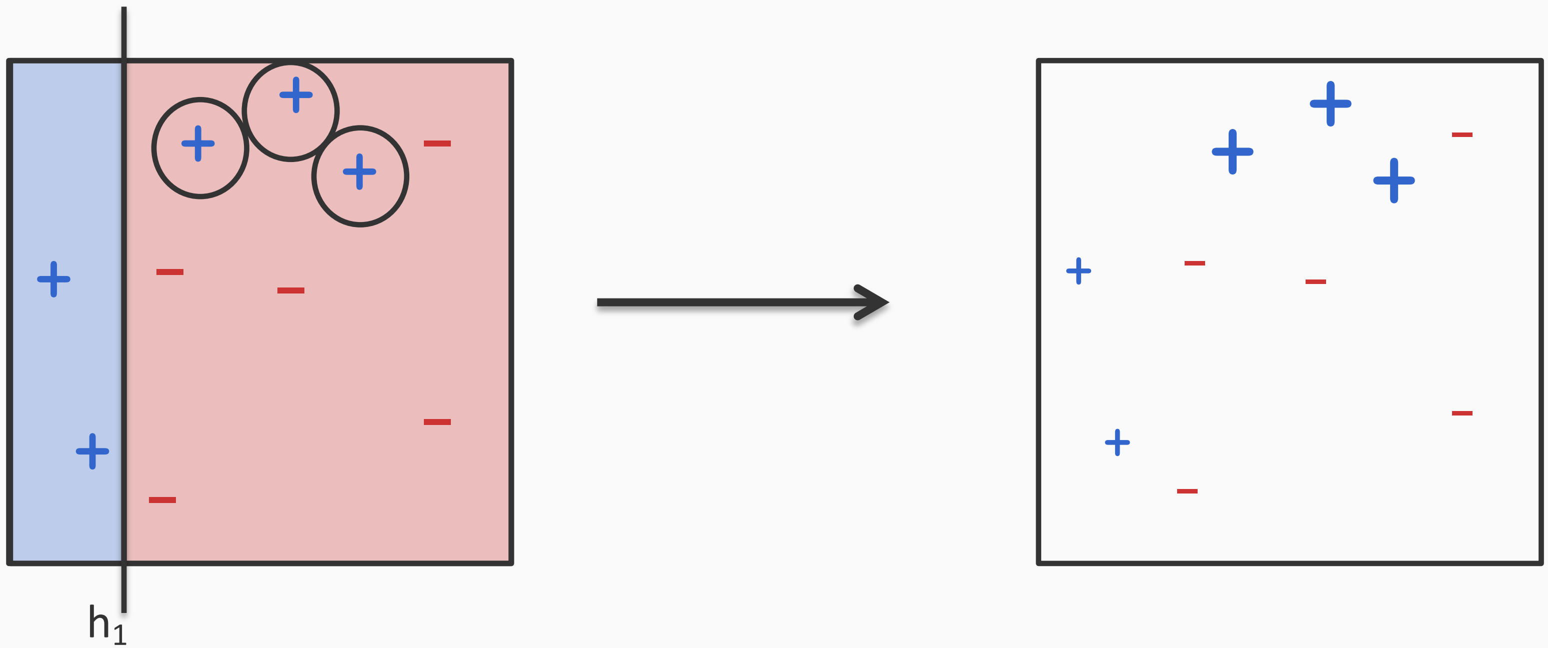
Our weak learner: An axis parallel line



Initially all examples are equally important

h_1 = The best classifier on this data
Clearly there are mistakes. Error ϵ_1 = 0.3

A toy example

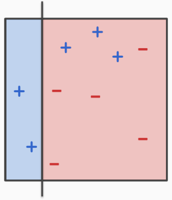


Initially all examples are equally important

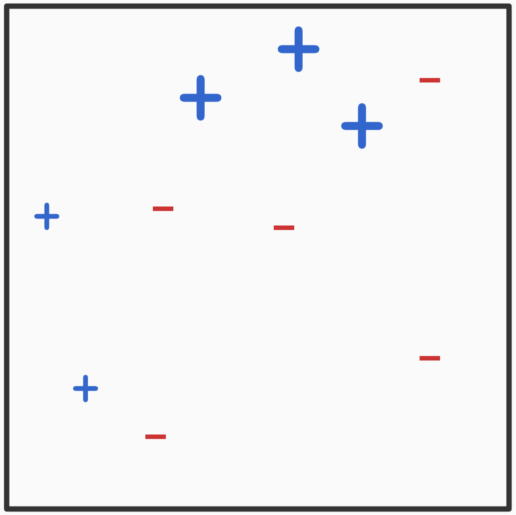
h_1 = The best classifier on this data
Clearly there are mistakes. Error ϵ_1 = 0.3

For the next round, increase the importance of the examples with mistakes and down-weight the examples that h_1 got correctly

A toy example

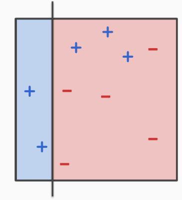


h_1

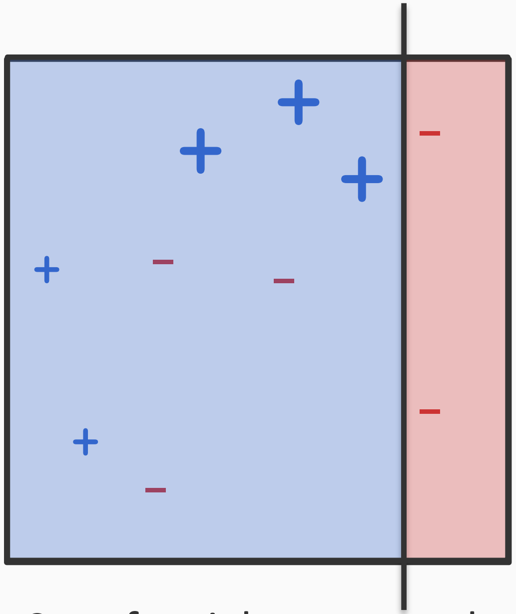


D_t = Set of weights at round t, one for each example. Think “How much should the weak learner care about this example in its choice of the classifier?”

A toy example

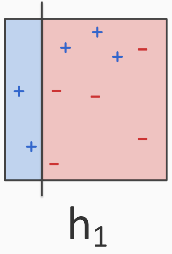


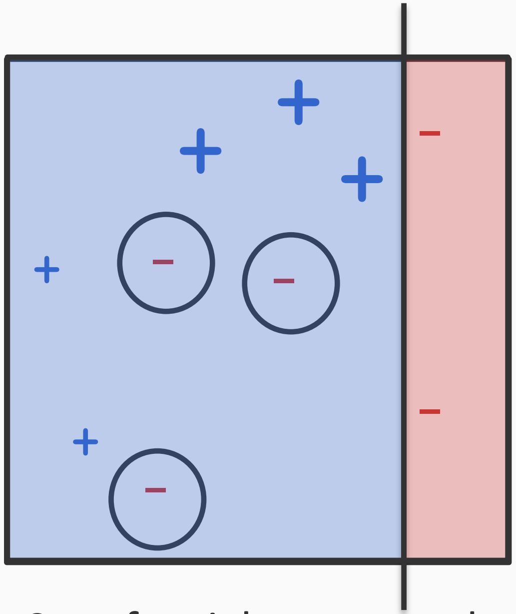
h_1



D_t = Set of weights at round t, one for each example. Think "How much should the weak learner care about this example in its choice of the classifier?"

A toy example

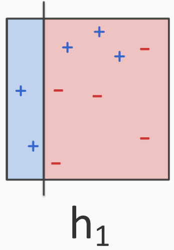


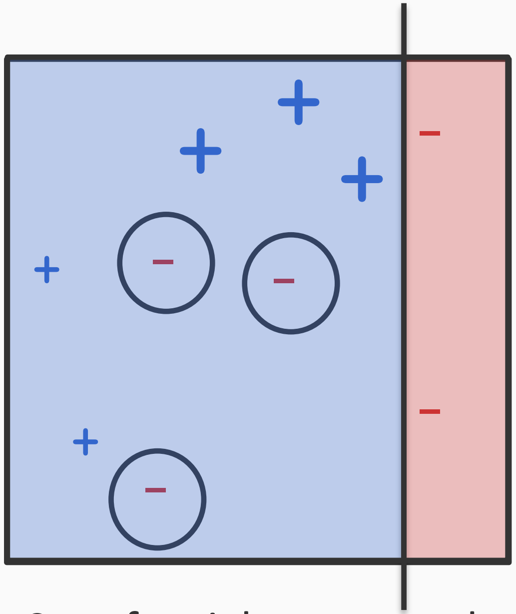


D_t = Set of weights at round t, one for each example. Think "How much should the weak learner care about this example in its choice of the classifier?"

h_1 = A classifier learned on this data. *Has an error ϵ_2 = 0.21*

A toy example



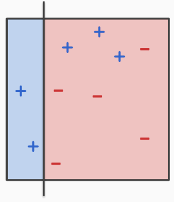


D_t = Set of weights at round t, one for each example. Think "How much should the weak learner care about this example in its choice of the classifier?"

h_2 = A classifier learned on this data. *Has an error ϵ_2 = 0.21*

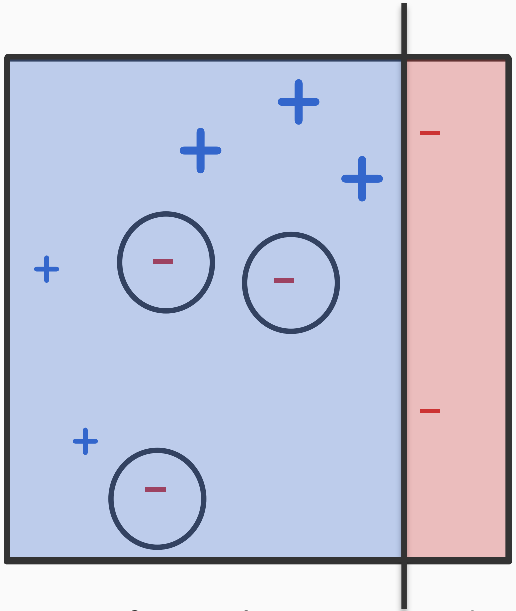
Why not 0.3? Because while computing error, we will weight each example x_i by its $D_t(i)$

A toy example



$$\epsilon_t = \frac{1}{2} - \frac{1}{2}\left(\sum_{i=1}^{m} D_t(i) y_i h(x_i)\right)$$

Why is this a reasonable definition?

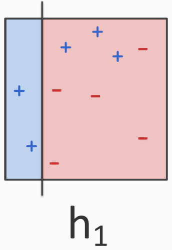


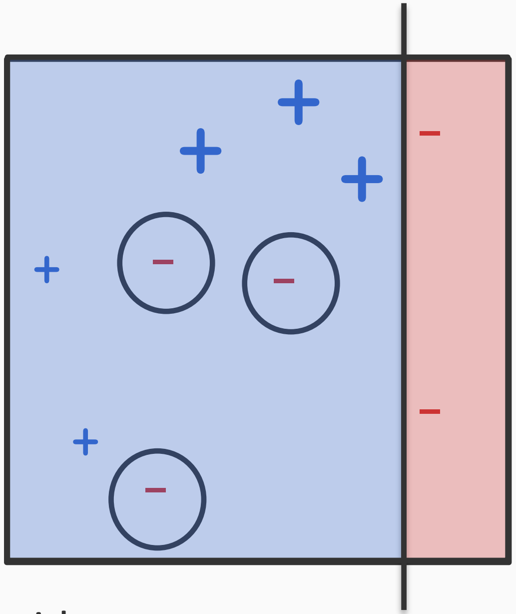
D_t = Set of weights at round t, one for each example. Think "How much should the weak learner care about this example in its choice of the classifier?"

h_2 = A classifier learned on this data. *Has an error ϵ_2 = 0.21*

Why not 0.3? Because while computing error, we will weight each example x_i by its $D_t(i)$

A toy example





$$\epsilon_t = \frac{1}{2} - \frac{1}{2}\left(\sum_{i=1}^{m} D_t(i) y_i h(x_i)\right)$$

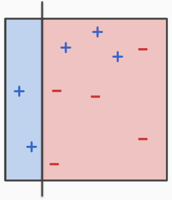
Why is this a reasonable definition?

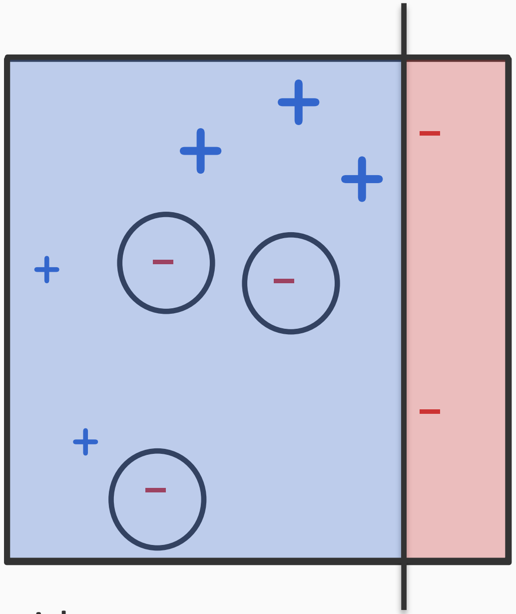
Consider two cases

Case 1: When $y \neq h(x)$

Case 2: When $y = h(x)$

A toy example





$$\epsilon_t = \frac{1}{2} - \frac{1}{2}\left(\sum_{i=1}^{m} D_t(i)y_i h(x_i)\right)$$

Why is this a reasonable definition?

Consider two cases

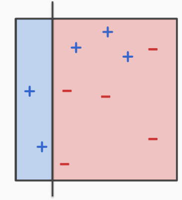
Case 1: When $y \neq h(x)$

we have $y_i h(x_i) = -1$

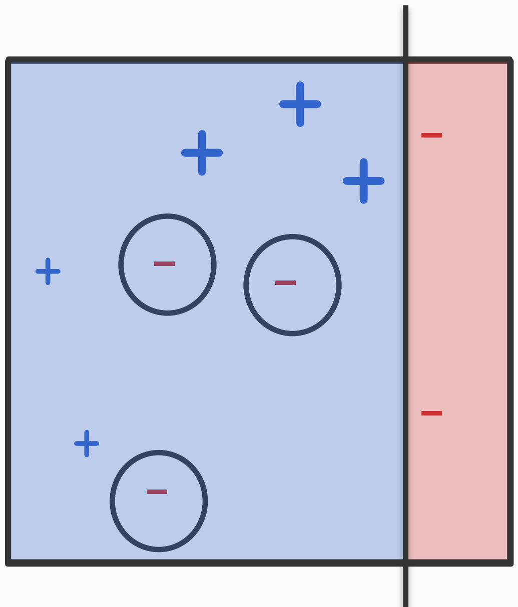
Case 2: When $y = h(x)$

we have $y_i h(x_i) = +1$

A toy example



h_1



Consider two cases

Case 1: When $y \neq h(x)$

we have $y_i h(x_i) = -1$

Case 2: When $y = h(x)$

we have $y_i h(x_i) = +1$

$$\epsilon_t = \frac{1}{2} - \frac{1}{2}\left(\sum_{i=1}^{m} D_t(i) y_i h(x_i)\right)$$

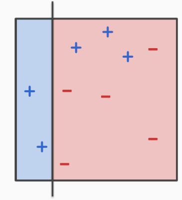
Why is this a reasonable definition?

$$\epsilon_t = \sum_{\substack{i \\ \text{where} \\ y_i \neq h(x_i)}} D_t(i)$$

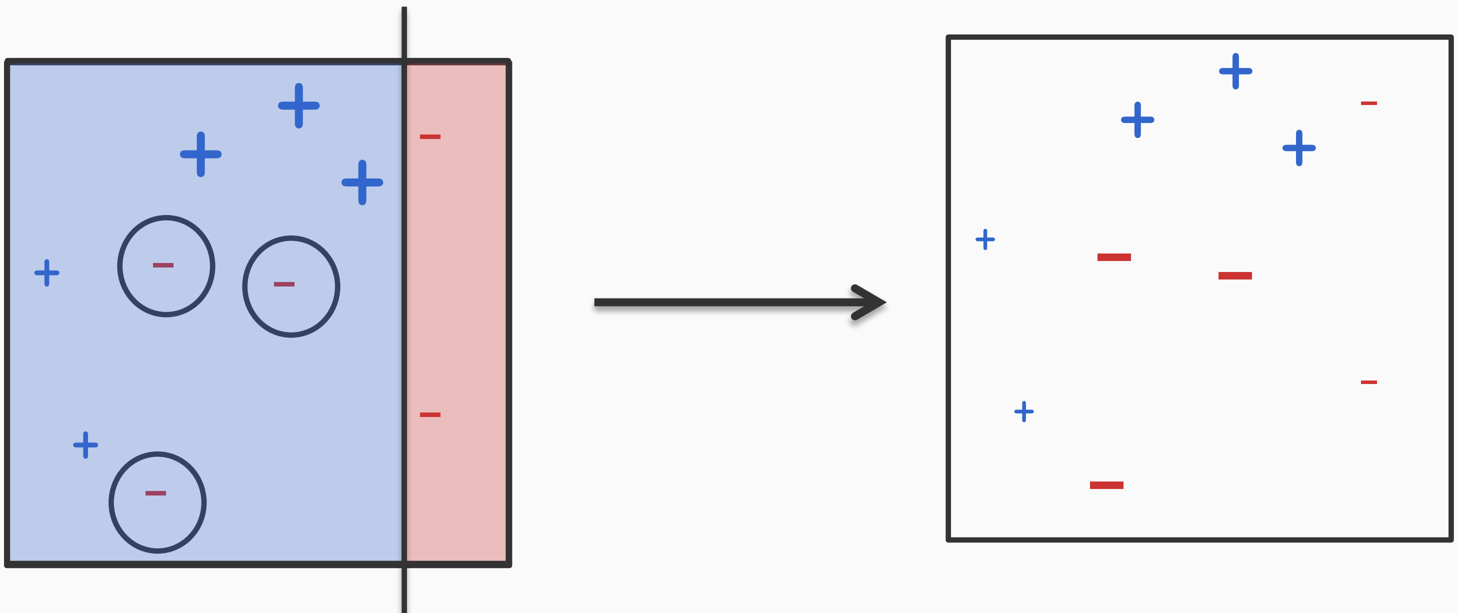
Represents the total error, but each example only contributes to the extent that it is important

Exercise: Show this

A toy example



h_1

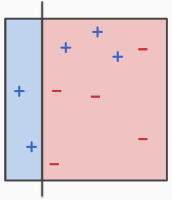


D_t = Set of weights at round t, one for each example. Think "How much should the weak learner care about this example in its choice of the classifier?"

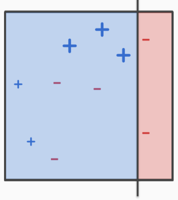
h_2 = A classifier learned on this data. *Has an error $\epsilon_2 = 0.21$*

For the next round, increase the importance of the mistakes and down-weight the examples that h_2 got correctly

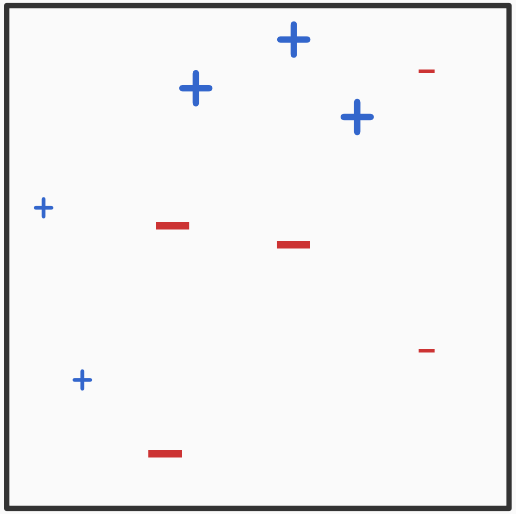
A toy example



h_1

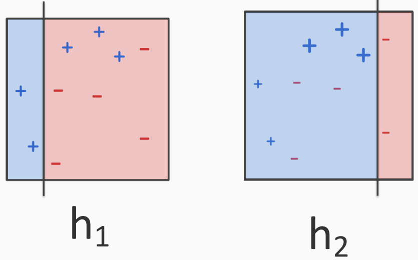


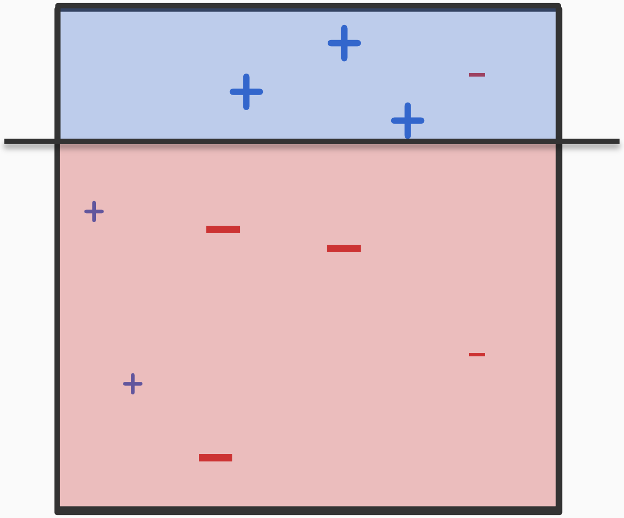
h_2



D_t = Set of weights at round t, one for each example. Think "How much should the weak learner care about this example in its choice of the classifier?"

A toy example



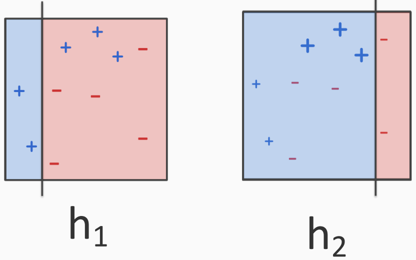


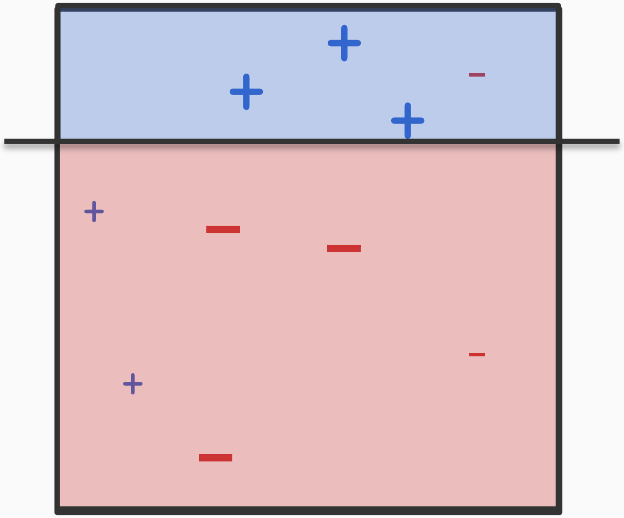
$$\epsilon_t = \frac{1}{2} - \frac{1}{2}\left(\sum_{i=1}^{m} D_t(i)\; y_i h(x_i)\right)$$

D_t = Set of weights at round t, one for each example. Think "How much should the weak learner care about this example in its choice of the classifier?"

h_2 = A classifier learned on this data. *Has an error ϵ_3= 0.14*

A toy example





$$\epsilon_t = \frac{1}{2} - \frac{1}{2}\left(\sum_{i=1}^{m} D_t(i)\; y_i h(x_i)\right)$$

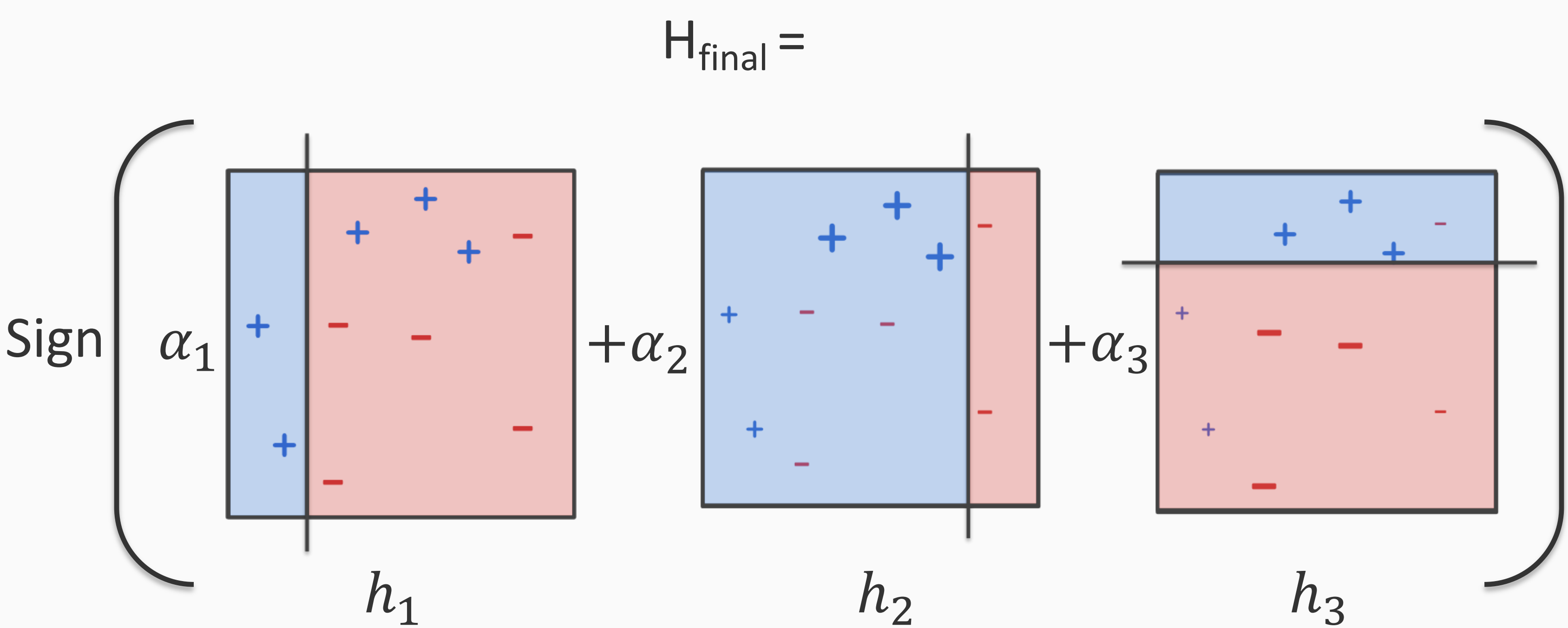
D_t = Set of weights at round t, one for each example. Think "How much should the weak learner care about this example in its choice of the classifier?"

h_2 = A classifier learned on this data. *Has an error ϵ_3= 0.14*

Why not 0.3? Because while computing error, we will weight each example x_i by its $D_t(i)$

A toy example

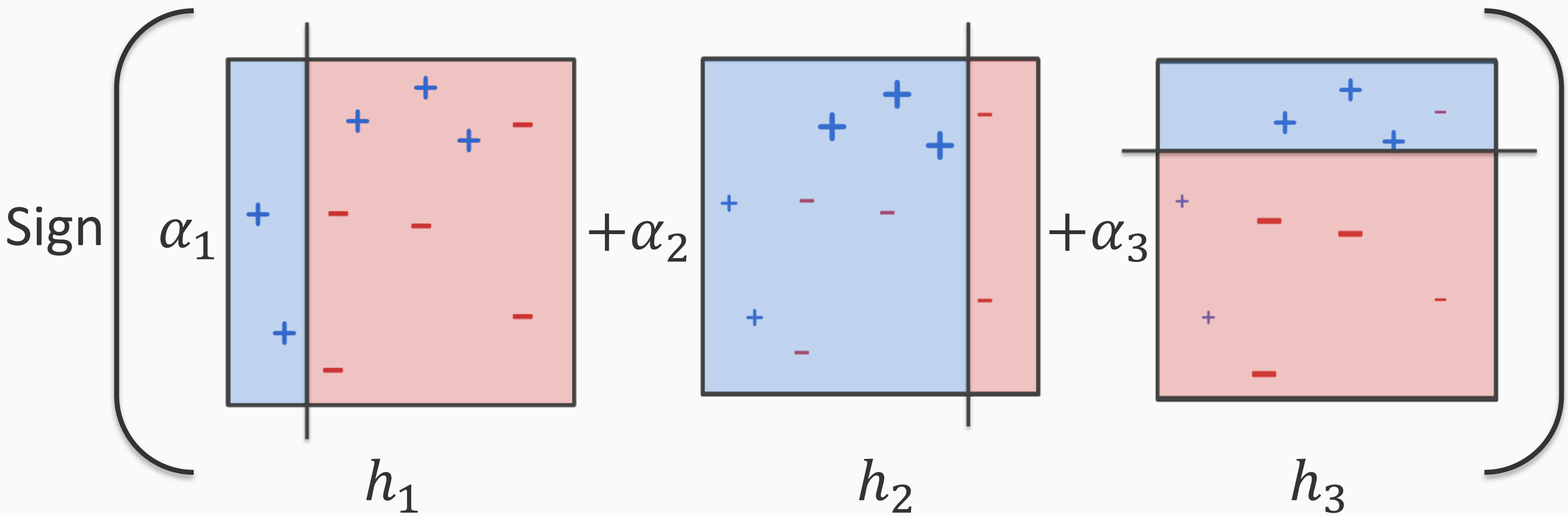
The final hypothesis is a combination of all the h_i's we have seen so far



A toy example

The final hypothesis is a combination of all the h_i's we have seen so far

H_{final} =



Think of the α values as the vote for each weak classifier and the boosting algorithm has to somehow specify them

An outline of Boosting

Given a training set $(x_1, y_1), (x_2, y_2), \cdots (x_m, y_m)$

- Instances $x_i \in X$ labeled with $y_i \in \{-1, +1\}$

- For t = 1, 2, $\cdots$, T:
  - Construct a distribution D_t on $\{1, 2, \cdots, m\}$
  - Find a weak hypothesis (rule of thumb) h_t such that it has a small *weighted* error ϵ_t
- Construct a final output H_{final}

An outline of Boosting

Given a training set $(x_1, y_1), (x_2, y_2), \cdots (x_m, y_m)$

– Instances $x_i \in X$ labeled with $y_i \in \{-1, +1\}$

- For t = 1, 2, ⋯, T:
  - Construct a distribution D_t on $\{1, 2, \cdots, m\}$
  - Find a weak hypothesis (rule of thumb) h_t such that it has a small *weighted* error ϵ_t
- Construct a final output H_{final}

Need to specify these two to get a complete algorithm

AdaBoost: Constructing D_t

We have m examples

D_t is a set of weights over the examples

$$D_t(1), D_t(2), \cdots, D_t(m)$$

At every round, the weak learner looks for hypotheses h_t that emphasizes examples that have a higher D_t

AdaBoost: Constructing D_t

Initially (t = 1), use the uniform distribution over all examples

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

AdaBoost: Constructing D_t

Initially (t = 1), use the uniform distribution over all examples

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

After t rounds

- What we have
  - D_t and the hypothesis h_t that was learned
  - The ϵ_t of that hypothesis on the training data

AdaBoost: Constructing D_t

Initially (t = 1), use the uniform distribution over all examples

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

After t rounds

- What we have
  - D_t and the hypothesis h_t that was learned
  - The ϵ_t of that hypothesis on the training data

- What we want from the $(t+1)^{th}$ round
  - Find a hypothesis so that examples that were incorrect in the previous round are correctly predicted by the new one
  - That is, increase the importance of misclassified examples and decrease the importance of correctly predicted ones

AdaBoost: Constructing D_t

Initially ($t = 1$), use the uniform distribution over all examples

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

After t rounds, we have some D_t and a hypothesis h_t that the weak learner produced

Create D_{t+1} as follows:

$$D_{t+1}(i) \quad = \quad \frac{D_t(i)}{Z_t} \begin{cases} e^{-\alpha_t} & \text{if } y_i = h_t(x_i) \\ e^{\alpha_t} & \text{if } y_i \neq h_t(x_i) \end{cases}$$

AdaBoost: Constructing D_t

Initially (t = 1), use the uniform distribution over all examples

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

After t rounds, we have some D_t and a hypothesis h_t that the weak learner produced

Create D_{t+1} as follows:

$$D_{t+1}(i) = \frac{D_t(i)}{Z_t} \begin{cases} e^{-\alpha_t} & \text{if } y_i = h_t(x_i) \\ e^{\alpha_t} & \text{if } y_i \neq h_t(x_i) \end{cases}$$

Demote correctly predicted examples (because, as we will see, $\alpha_t > 0$)

Promote incorrectly predicted examples (because, as we will see, $\alpha_t > 0$)

AdaBoost: Constructing D_t

Initially ($t = 1$), use the uniform distribution over all examples

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

After t rounds, we have some D_t and a hypothesis h_t that the weak learner produced

Create D_{t+1} as follows:

$$\begin{aligned} D_{t+1}(i) &= \frac{D_t(i)}{Z_t} \begin{cases} e^{-\alpha_t} & \text{if } y_i = h_t(x_i) \\ e^{\alpha_t} & \text{if } y_i \neq h_t(x_i) \end{cases} \\ &= \frac{D_t(i)}{Z_t} \cdot \exp\left(-\alpha_t \cdot y_i h_t(x_i)\right) \end{aligned}$$

AdaBoost: Constructing D_t

After t rounds, we have some D_t and a hypothesis h_t that the weak learner produced

Create D_{t+1} as follows:

$$\begin{aligned} D_{t+1}(i) &= \frac{D_t(i)}{Z_t} \begin{cases} e^{-\alpha_t} & \text{if } y_i = h_t(x_i) \\ e^{\alpha_t} & \text{if } y_i \neq h_t(x_i) \end{cases} \\ &= \frac{D_t(i)}{Z_t} \cdot \exp\left(-\alpha_t \cdot y_i h_t(x_i)\right) \end{aligned}$$

AdaBoost: Constructing D_t

After t rounds, we have some D_t and a hypothesis h_t that the weak learner produced

Create D_{t+1} as follows:

$$
\begin{aligned}
D_{t+1}(i) &= \frac{D_t(i)}{Z_t} \begin{cases} e^{-\alpha_t} & \text{if } y_i = h_t(x_i) \\ e^{\alpha_t} & \text{if } y_i \neq h_t(x_i) \end{cases} \\
&= \frac{D_t(i)}{Z_t} \cdot \exp\left(-\alpha_t \cdot y_i h_t(x_i)\right)
\end{aligned}
$$

Z_t: A normalization constant. Ensures that the weights D_{t+1} add up to 1

Exercise: How should we compute the value of Z_t?

AdaBoost: Constructing D_t

After t rounds, we have some D_t and a hypothesis h_t that the weak learner produced

Create D_{t+1} as follows:

$$
\begin{aligned}
D_{t+1}(i) &= \frac{D_t(i)}{Z_t} \begin{cases} e^{-\alpha_t} & \text{if } y_i = h_t(x_i) \\ e^{\alpha_t} & \text{if } y_i \neq h_t(x_i) \end{cases} \\
&= \frac{D_t(i)}{Z_t} \cdot \exp\left(-\alpha_t \cdot y_i h_t(x_i)\right)
\end{aligned}
$$

Z_t: A normalization constant. Ensures that the weights D_{t+1} add up to 1

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

Since $\epsilon_t < \frac{1}{2}$, the value of $\alpha_t > 0$

AdaBoost: Constructing D_t

After t rounds, we have some D_t and a hypothesis h_t that the weak learner produced

Create D_{t+1} as follows:

$$
\begin{aligned}
D_{t+1}(i) &= \frac{D_t(i)}{Z_t} \begin{cases} e^{-\alpha_t} & \text{if } y_i = h_t(x_i) \\ e^{\alpha_t} & \text{if } y_i \neq h_t(x_i) \end{cases} \\
&= \frac{D_t(i)}{Z_t} \cdot \exp\left(-\alpha_t \cdot y_i h_t(x_i)\right)
\end{aligned}
$$

Z_t: A normalization constant. Ensures that the weights D_{t+1} add up to 1

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

Eventually, the classifier h_t gets a vote of α_t in the final classifier

An outline of Boosting

Given a training set $(x_1, y_1), (x_2, y_2), \cdots (x_m, y_m)$

– Instances $x_i \in X$ labeled with $y_i \in \{-1, +1\}$

- For t = 1, 2, ⋯, T:
  ✓ $\boxed{\text{Construct a distribution } D_t \text{ on } \{1, 2, \cdots, m\}}$
  – Find a weak hypothesis (rule of thumb) h_t such that it has a small *weighted* error ϵ_t
- $\boxed{\text{Construct a final output } H_{final}}$

Need to specify these two to get a complete algorithm

The final hypothesis

- After T rounds, we have
  - T weak classifiers $h_1, h_2, \cdots h_T$
  - T values of α_t

- Recall that each weak classifier is takes an example x and produces a -1 or a +1

- Define the final hypothesis H_{final} as

$$H_{final}(x) = \text{sgn}\left(\sum_t \alpha_t h_t(x)\right)$$

AdaBoost: The full algorithm

Given a training set $(x_1, y_1), (x_2, y_2), \cdots (x_m, y_m)$
Instances $x_i \in X$ labeled with $\mathrm{y}_i \in \{-1, +1\}$

T: a parameter to the learner

1. Initialize $D_1(i) = \frac{1}{m}$ for each example indexed by $i \in \{1,2, \cdots, m\}$

AdaBoost: The full algorithm

Given a training set $(x_1, y_1), (x_2, y_2), \cdots (x_m, y_m)$

Instances $x_i \in X$ labeled with $\mathrm{y}_i \in \{-1, +1\}$

T: a parameter to the learner

1. Initialize $D_1(i) = \frac{1}{m}$ for each example indexed by $i \in \{1,2,\cdots,m\}$
2. For t = 1, 2, $\cdots$ T:
   - Find a classifier h_t whose *weighted classification error* is better than chance

AdaBoost: The full algorithm

Given a training set $(x_1, y_1), (x_2, y_2), \cdots (x_m, y_m)$
Instances $x_i \in X$ labeled with $\mathrm{y}_i \in \{-1, +1\}$

T: a parameter to the learner

1. Initialize $D_1(i) = \frac{1}{m}$ for each example indexed by $i \in \{1,2, \cdots, m\}$
2. For t = 1, 2, ⋯ T:
   - Find a classifier h_t whose *weighted classification error* is better than chance
   - Compute its vote

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

AdaBoost: The full algorithm

Given a training set $(x_1, y_1), (x_2, y_2), \cdots (x_m, y_m)$
Instances $x_i \in X$ labeled with $y_i \in \{-1, +1\}$

T: a parameter to the learner

1. Initialize $D_1(i) = \frac{1}{m}$ for each example indexed by $i \in \{1, 2, \cdots, m\}$
2. For t = 1, 2, ⋯ T:
   - Find a classifier h_t whose *weighted classification error* is better than chance
   - Compute its vote

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

   - Update the values of the weights for the training examples

$$D_{t+1}(i) = \frac{D_t(i)}{Z_t} \cdot \exp(-\alpha_t \cdot y_i h_t(x_i))$$

AdaBoost: The full algorithm

Given a training set $(x_1, y_1), (x_2, y_2), \cdots (x_m, y_m)$
Instances $x_i \in X$ labeled with $\mathrm{y}_i \in \{-1, +1\}$

T: a parameter to the learner

1. Initialize $D_1(i) = \frac{1}{m}$ for each example indexed by $i \in \{1,2, \cdots, m\}$
2. For t = 1, 2, ⋯ T:
   - Find a classifier h_t whose *weighted classification error* is better than chance
   - Compute its vote

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

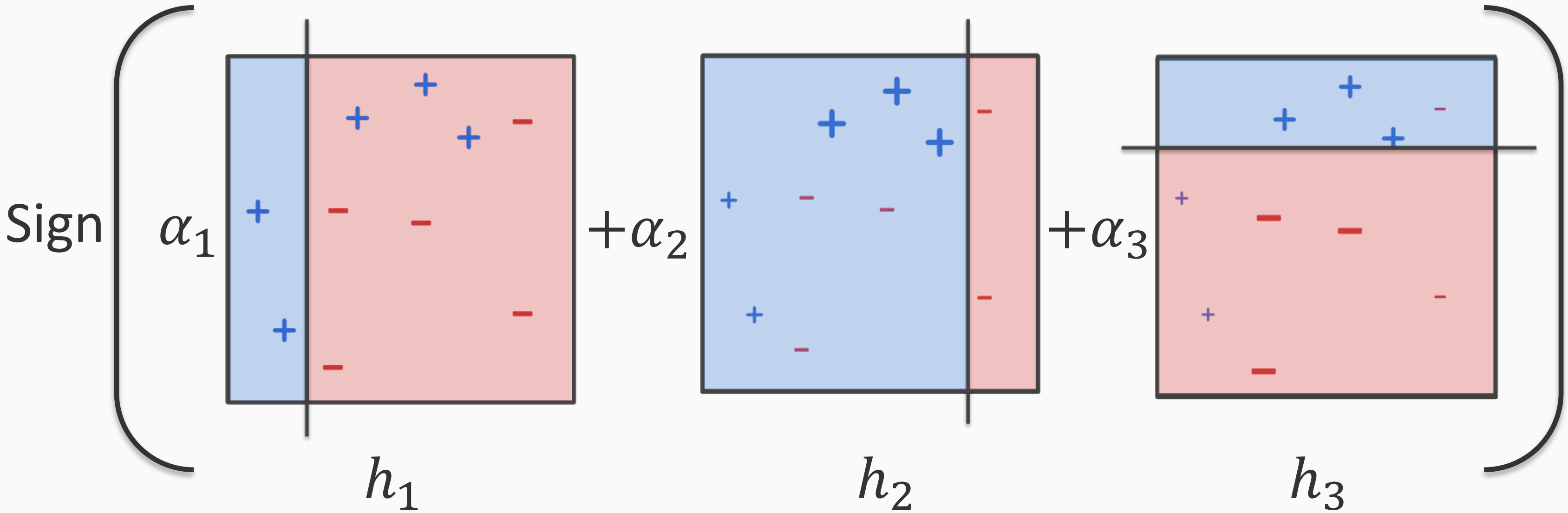
   - Update the values of the weights for the training examples

$$D_{t+1}(i) = \frac{D_t(i)}{Z_t} \cdot \exp\big(-\alpha_t \cdot y_i h_t(x_i)\big)$$

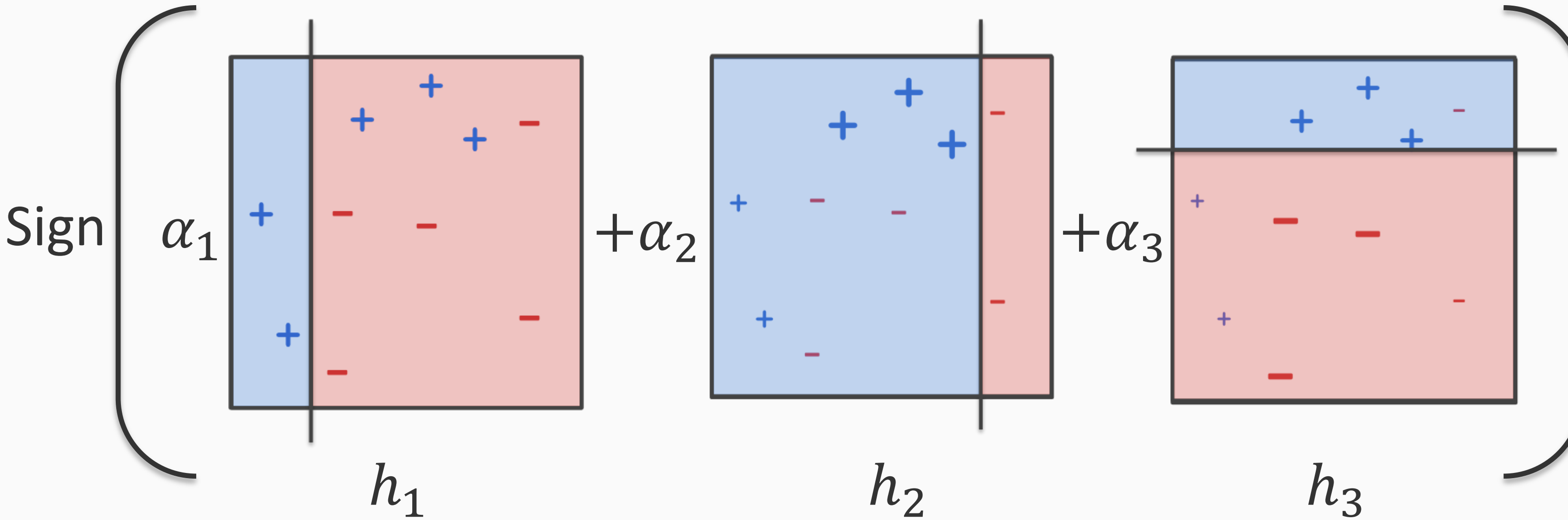
3. Return the final hypothesis that predicts labels as

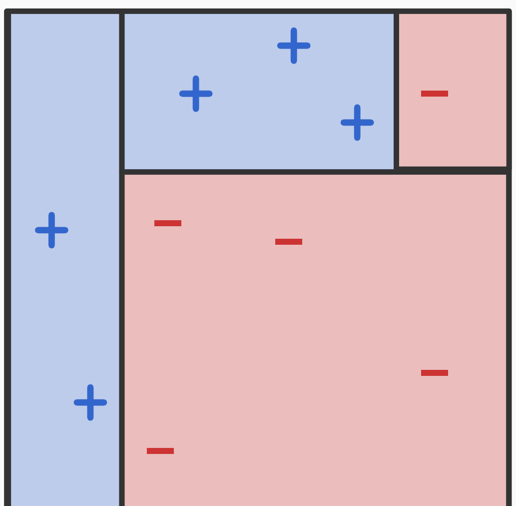
$$H_{final}(x) = \operatorname{sgn}\left(\sum_{t=1}^{T} \alpha_t h_t(x) \right)$$

Back to the toy example



Back to the toy example





Analyzing the training error

Theorem:

- Run AdaBoost for T rounds
- Let $\epsilon_t = \frac{1}{2} - \gamma_t$
- Let $0 < \gamma_t \leq \gamma$ for all t
- Then,

$$\text{Training error}\left(H_{final}\right) \leq e^{-2\gamma^2 T}$$

Analyzing the training error

Theorem:

- Run AdaBoost for T rounds
- Let $\epsilon_t = \frac{1}{2} - \gamma_t$ We have a weak learner
- Let $0 < \gamma_t \leq \gamma$ for all t
- Then,

$$\text{Training error}\left(H_{final}\right) \leq e^{-2\gamma^2 T}$$

Analyzing the training error

Theorem:

- Run AdaBoost for T rounds
- Let $\epsilon_t = \frac{1}{2} - \gamma_t$
- Let $0 < \gamma_t \leq \gamma$ for all t
- Then,

$$\text{Training error}\left(H_{final}\right) \leq e^{-2\gamma^2 T}$$

We have a weak learner

As T increases, the training error drops exponentially

Analyzing the training error

Theorem:

- Run AdaBoost for T rounds
- Let $\epsilon_t = \frac{1}{2} - \gamma_t$
- Let $0 < \gamma_t \leq \gamma$ for all t
- Then,

$$\text{Training error}(H_{final}) \leq e^{-2\gamma^2 T}$$

We have a weak learner

As T increases, the training error drops exponentially

Proof is simple, see pointer on website

Analyzing the training error

Theorem:

- Run AdaBoost for T rounds
- Let $\epsilon_t = \frac{1}{2} - \gamma_t$
- Let $0 < \gamma_t \leq \gamma$ for all t
- Then,

$$\text{Training error}(H_{final}) \leq e^{-2\gamma^2 T}$$

Proof is simple, see pointer on website

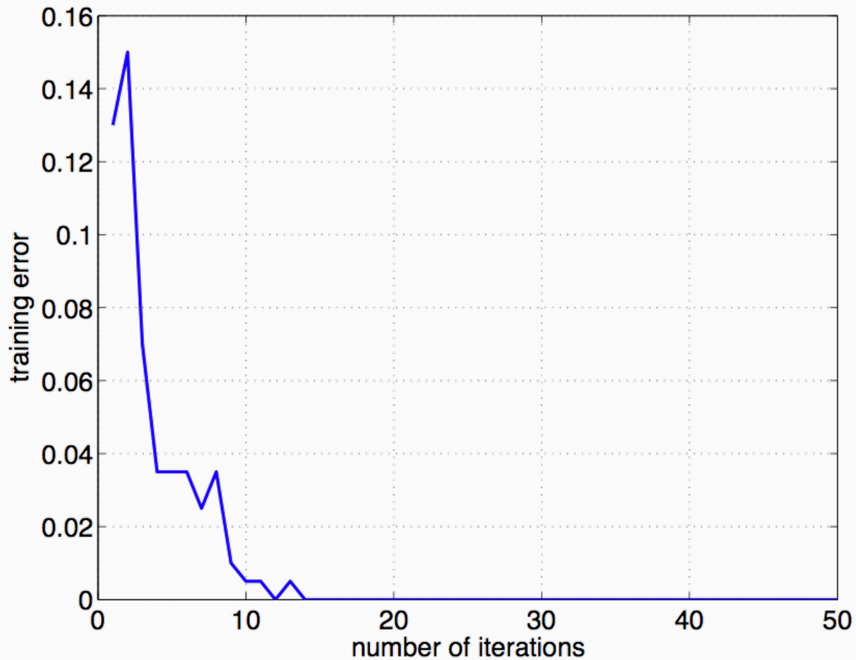
We have a weak learner

As T increases, the training error drops exponentially

Is it sufficient to upper bound the training error?

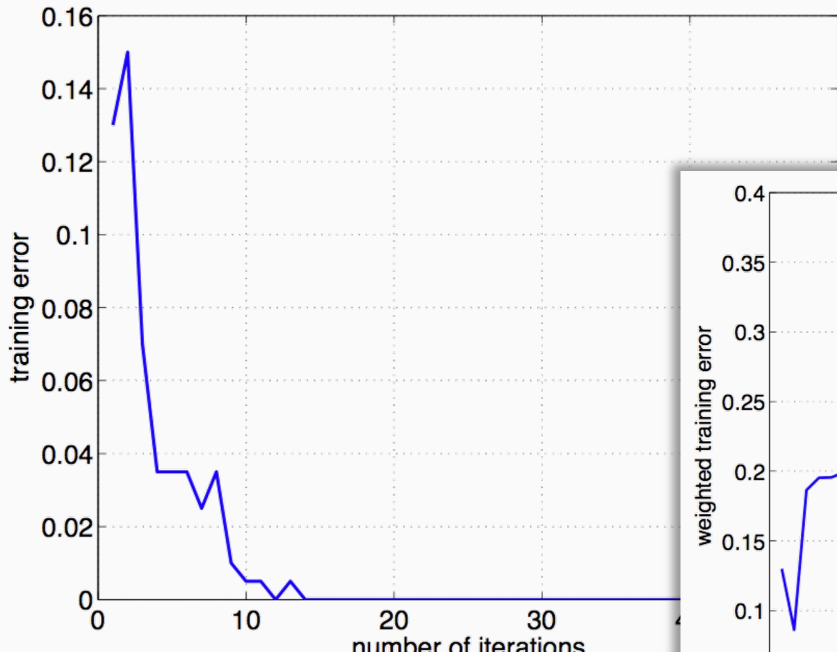
Adaboost: Training error

The training error of the combined classifier decreases exponentially fast if the errors of the weak classifiers (the ϵ_t) are strictly better than chance

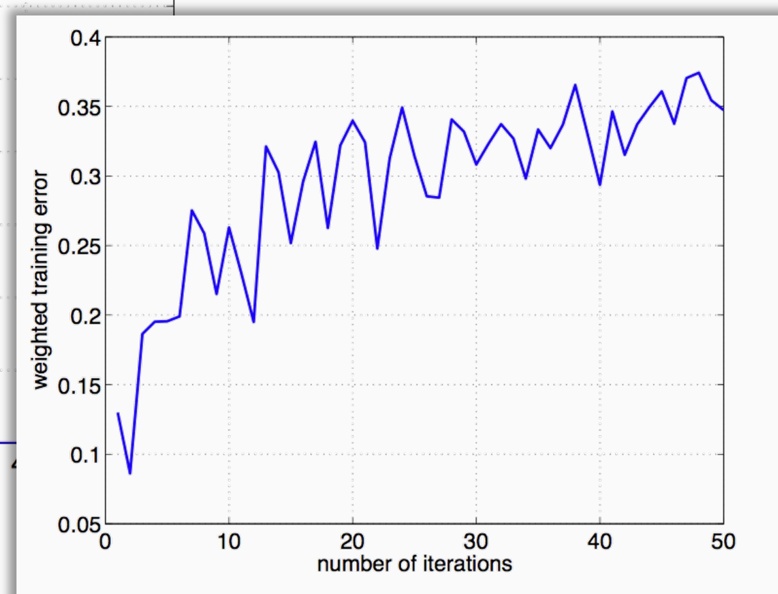


Adaboost: Training error

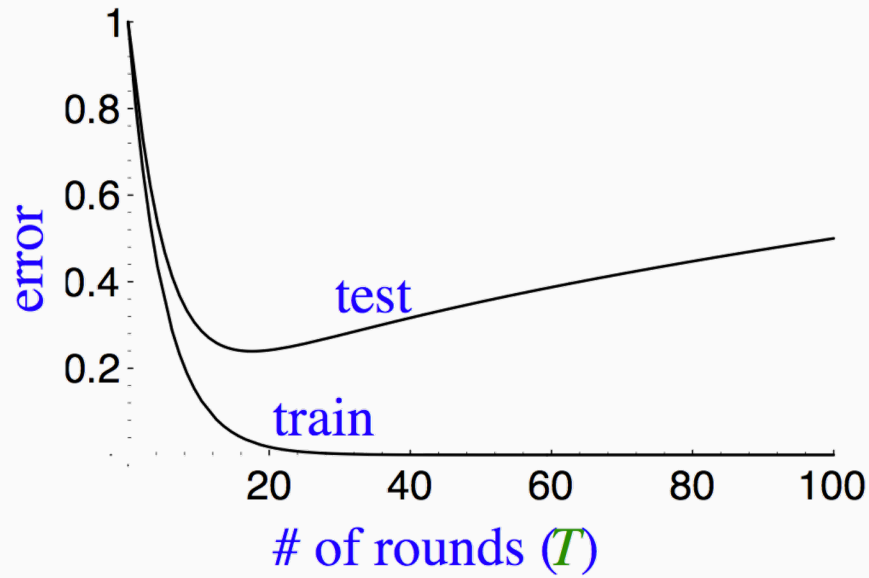
The training error of the combined classifier decreases exponentially fast if the errors of the weak classifiers (the ϵ_t) are strictly better than chance



The individual classifier errors (ϵ_t) tend to increase



What about the test error?



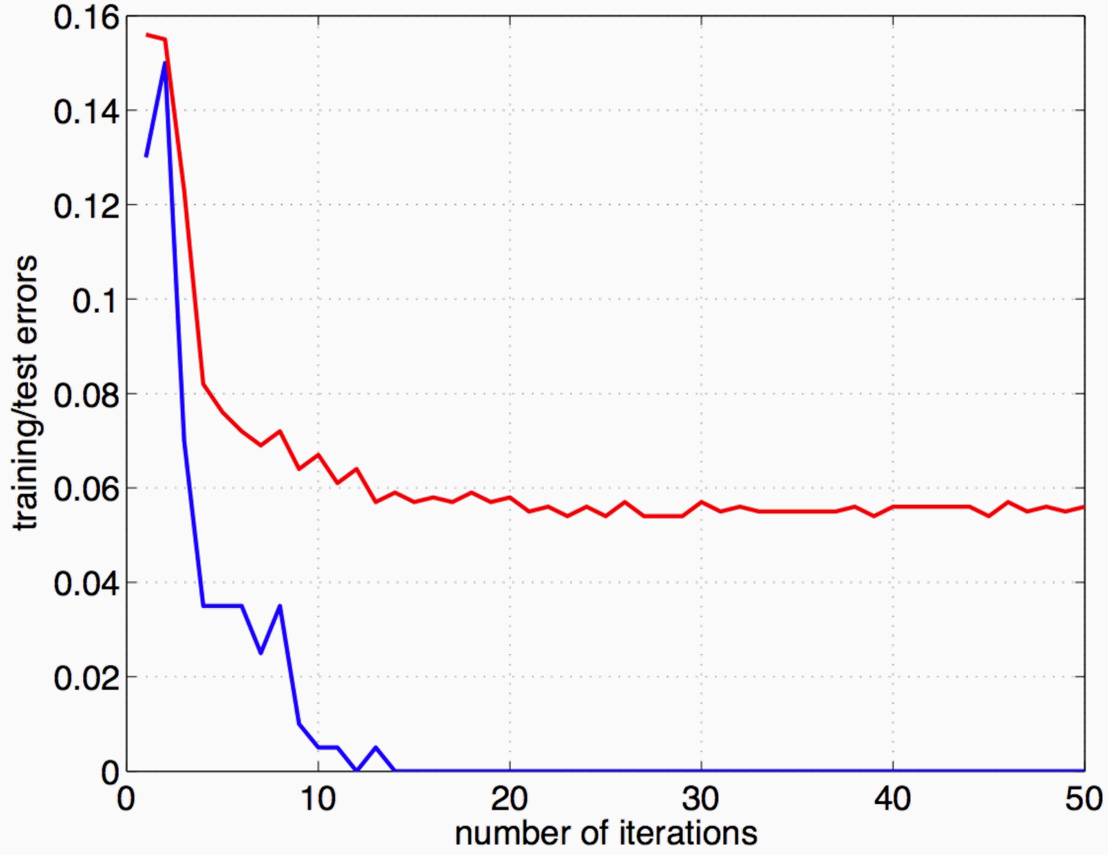
What the theory tells us:

Training error will keep decreasing or reach zero (the AdaBoost theorem)

Test error will increase after the H_{final} becomes too "complex"

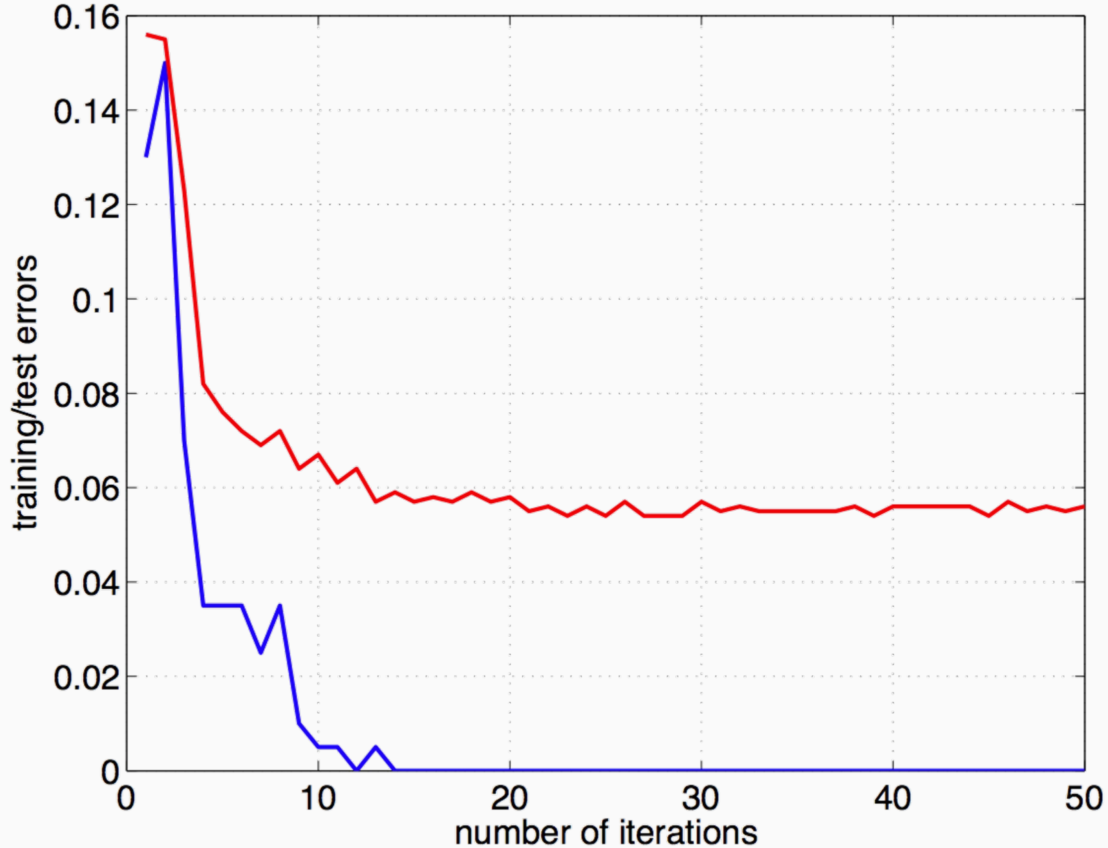
- Think about Occam's razor and overfitting

In practice



0.16
0.14
0.12
0.1
0.08
0.06
0.04
0.02
0
training/test errors
0
10
20
30
40
50
number of iterations

In practice



Strange observation: Test error may decrease even after training error has hit zero! Why? (One possible explanation in [Schapire, Freund, Bartlett, Lee, 1997])

AdaBoost: Summary

- What is good about it
  - Simple, fast and only one additional parameter to tune (T)
  - Use it with any weak learning algorithm
    - Which means that we only need to look for classifiers that are slightly better than chance
- Caveats
  - Performance often depends on dataset and the weak learners
  - Can fail if the weak learners are too complex (overfitting)
  - Can fail if the weak learners are too weak (underfitting)

- Empirical evidence [Caruana and Niculescu-Mizil, 2006] that boosted decision stumps are the best approach to try if you have a small number of features (no more than hundreds)

Boosting and Ensembles

- What is boosting?
- AdaBoost
- Ensemble methods
  - Boosting, Bagging and Random Forests

Ensemble methods

- In general, meta algorithms that combine the output of multiple classifiers
- Often tend to be empirically robust
- Eg: The winner of the $1 million Netflix prize in 2009 was a giant ensemble

Boosting

- Initialization:
  - Weigh all training samples equally
- Iteration Step:
  - Train model on weighted train set
  - Compute weighted error of model on train set
  - Increase weights on training cases model gets wrong
- Typically requires 100's to 1000's of iterations
- Return final model:
  - Carefully weighted prediction of each model

Boosting: Different Perspectives

- Boosting is a maximum-margin method (Schapire et al. 1998, Rosset et al. 2004)
  - Trades lower margin on easy cases for higher margin on harder cases
- Boosting is an additive logistic regression model (Friedman, Hastie and Tibshirani 2000)
  - Tries to fit the logit of the true conditional probabilities
- Boosting is an *equalizer* (Breiman 1998, Friedman, Hastie, Tibshirani 2000)
  - Weighted proportion of the number of times an example is misclassified by base learners tends to be the same for all training cases
- Boosting is a linear classifier, but does not give well calibrated probability estimate.

Bagging

Short for ***B**ootstrap* ***agg**regat**ing*** [Breiman, 1994]

- Given a training set with m examples
- Repeat t = 1, 2, ⋯, m:
  - Draw m’ (< m) samples *with replacement* from the training set
  - Train a classifier (any classifier) C_i

- Construct final classifier by taking votes from each C_i

Bagging

Short for ***B**ootstrap **agg**regat**ing***

- A method for generating multiple versions of a predictor and using these to get an aggregated predictor.
  - Averages over the versions when predicting a numerical outcome (regression)
  - Does a plurality vote when predicting a class (classification)

Bagging

Short for ***B**ootstrap **agg**regat**ing***

- A method for generating multiple versions of a predictor and using these to get an aggregated predictor.
  - Averages over the versions when predicting a numerical outcome (regression)
  - Does a plurality vote when predicting a class (classification)
- The multiple versions are constructed by making bootstrap replicates of the learning set and using these as training sets
  - That is, use samples of the data, with replacement

Bagging

Short for ***B**ootstrap **agg**regat**ing***

- A method for generating multiple versions of a predictor and using these to get an aggregated predictor.
  - Averages over the versions when predicting a numerical outcome (regression)
  - Does a plurality vote when predicting a class (classification)

- The multiple versions are constructed by making bootstrap replicates of the learning set and using these as training sets
  - That is, use samples of the data, with replacement

- Tests on real and simulated data sets using classification and regression trees and subset selection in linear regression show that bagging can give substantial gains in accuracy

Bagging

Short for ***Bootstrap aggregating***

- A method for generating multiple versions of a predictor and using these to get an aggregated predictor.
  - Averages over the versions when predicting a numerical outcome (regression)
  - Does a plurality vote when predicting a class (classification)

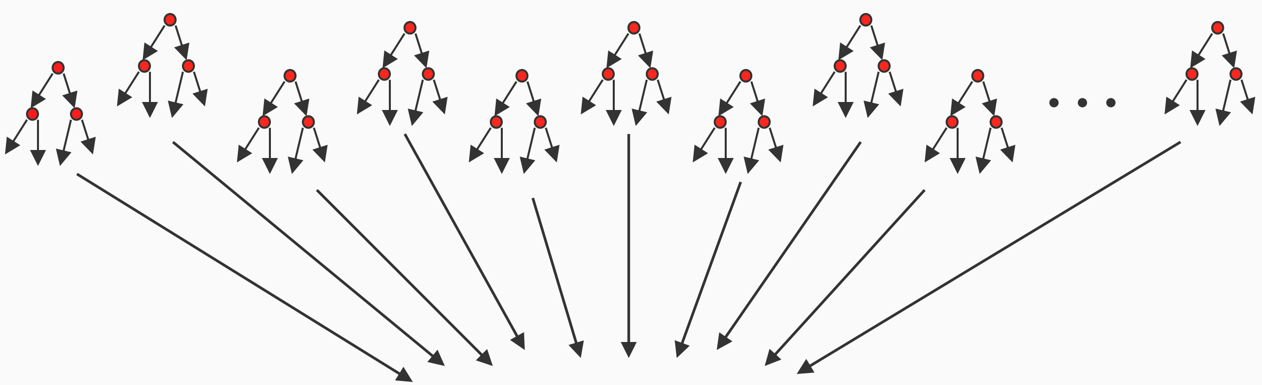
- The multiple versions are constructed by making bootstrap replicates of the learning set and using these as training sets
  - That is, use samples of the data, with replacement

- Tests on real and simulated data sets using classification and regression trees and subset selection in linear regression show that bagging can give substantial gains in accuracy

- Instability of the prediction method: If perturbing the training set can cause significant changes in the learned classifier *then* bagging can improve accuracy

Example: Bagged Decision Trees

- Draw T bootstrap samples of data
- Train trees on each sample to produce T trees
- Average prediction of trees on out-of-bag samples

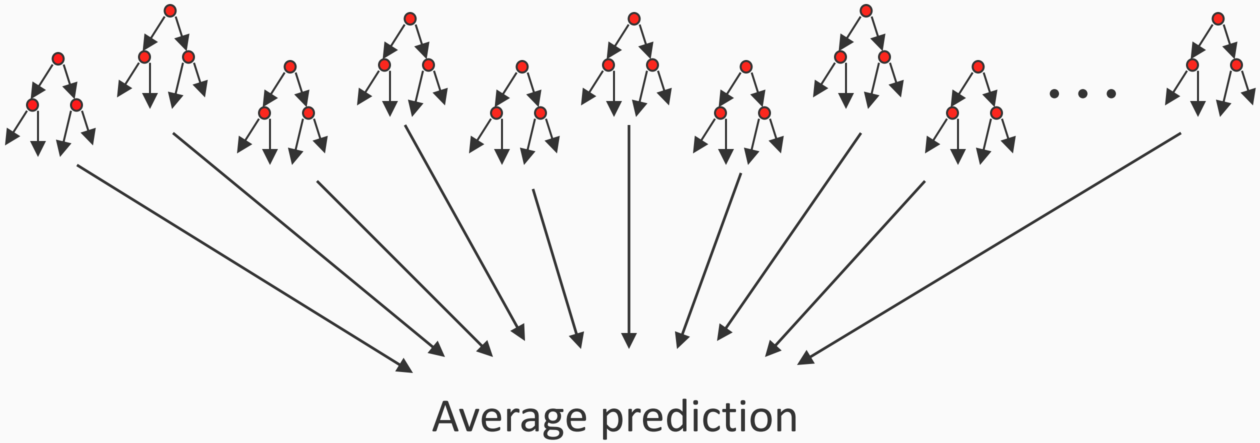


Average prediction

$$\frac{0.23\ +\ 0.19\ +\ 0.34\ +\ 0.22\ +\ 0.26\ +\ \ldots\ +\ 0.31}{\text{number of trees}} = 0.24$$

Random Forests (Bagged Trees++)

- Draw T (possibly ***1000s***) bootstrap samples of data
- ***Draw sample of available attributes at each split***
- Train trees on each sample+attribute set to produce T trees
- Average prediction of trees on out-of-bag samples



$$\frac{0.23 + 0.19 + 0.34 + 0.22 + 0.26 + \ldots + 0.31}{\text{number of trees}} = 0.24$$

Boosting and Ensembles: What have we seen?

- What is boosting?
  - Does weak learnability imply strong learnability?
- AdaBoost
  - Intuition
  - The algorithm
  - Why does it work
- Ensemble methods
  - Boosting, Bagging and Random Forests