Ensemble Learning

Machine Learning
Fall 2023

The slides are mainly from Vivek Srikumar
Ensemble Learning: Boosting and Bagging

- What is boosting?
- AdaBoost
- Bagging methods
Ensemble Learning: Boosting and Bagging

• What is boosting?

• AdaBoost

• Bagging methods
Boosting

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

• Historically: An answer to a theoretical question in PAC learning

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
Practically useful

- Face detector
- Hand-written letter recognition
- Text classification, news tagging
- Information retrieval
- Motion capture
- DNA sequence analysis
- ...

![Faces found](image.png)

![Handwritten letters](image2.png)
Boosting: The formal problem setup

• **Strong** PAC algorithm
  For any distribution over examples, for every \( \varepsilon > 0, \delta > 0, \)
given a polynomially many random examples
finds a hypothesis with error \( \leq \varepsilon \) with probability \( \geq 1 - \delta \)
Boosting: The formal problem setup

• **Strong** PAC algorithm
  For any distribution over examples, for every \( \varepsilon > 0, \delta > 0 \), given a polynomially many random examples finds a hypothesis with error \( \leq \varepsilon \) with probability \( \geq 1 - \delta \)

• **Weak** PAC algorithm
  – Same, but only for \( \varepsilon \geq \frac{1}{2} - \gamma \)
Boosting: The formal problem setup

- **Strong** PAC algorithm
  For any distribution over examples, for every $\varepsilon > 0$, $\delta > 0$, given a polynomially many random examples finds a hypothesis with error $\leq \varepsilon$ with probability $\geq 1 - \delta$

- **Weak** PAC algorithm
  - Same, but only for $\varepsilon \geq \frac{1}{2} - \gamma$

- **Question** [Kearns and Valiant ’88]:
  - Does weak learnability lead to strong learnability?
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 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 for Freund and Schapire
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 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 for Freund and Schapire
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 *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 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
Ensemble Learning

• What is boosting?

• AdaBoost
  – Intuition
  – The algorithm
  – Why does it work

• Bagging methods
A toy example

Initially all examples are equally important

Our weak learner: An axis parallel line

Or
A toy example

Initially all examples are equally important

$h_1 =$ The best classifier on this data

Our weak learner: An axis parallel line

Or
A toy example

Initially all examples are equally important

$h_1 = \text{The best classifier on this data}$

Clearly there are mistakes. Error $\epsilon_1 = 0.3$
A toy example

Initially all examples are equally important

$h_1 = \text{The best classifier on this data}$

Clearly there are mistakes. Error $\epsilon_1 = 0.3$

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

\[ h_1 \]

\[ D_t = \text{Set of weights at round } t, \text{ one for each example.} \text{ indicates } \text{“How much should the weak learner care about this example in its choice of the classifier?”} \]

\[ \sum_{i=1}^{m} D_t(i) = 1 \]
A toy example

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

\[ \sum_{i=1}^{m} D_t(i) = 1 \]
A toy example

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

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

$$\sum_{i=1}^{m} D_t(i) = 1$$
A toy example

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

\[ h_2 = \text{A classifier learned on this data. Has an error } \varepsilon_2 = 0.21 \]

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

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

\[ h_2 = A \text{ classifier learned on this data. } Has \text{ an error } \epsilon_2 = 0.21 \]

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

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

Why is this a reasonable definition?

\[ \sum_{i=1}^{m} D_t(i) = 1 \]
A toy example

Consider two cases

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

**Case 2:** When \( y = h(x) \)

Why is this a reasonable definition?

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

\[
\sum_{i=1}^{m} D_t(i) = 1
\]
A toy example

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 $y_i h(x_i) = +1$

Why is this a reasonable definition?

\[
\epsilon_t = \frac{1}{2} - \frac{1}{2} \left( \sum_{i=1}^{m} D_t(i) y_i h(x_i) \right)
\]
A toy example

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 $y_i h(x_i) = +1$

Why is this a reasonable definition?

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

$$\epsilon_t = \sum_i D_t(i)$$

where $y_i \neq h(x_i)$

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

Exercise: Show this
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 $\varepsilon_2 = 0.21$*

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

$D_t =$ 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

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

\[ h_3 = \text{A classifier learned on this data. } \text{Has an error } \epsilon_3 = 0.14 \]
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_3 = A classifier learned on this data. Has an error $\epsilon_3 = 0.14$

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

$$H_{\text{final}} = \alpha_1 + \alpha_2 + \alpha_3$$
A toy example

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

$$H_{\text{final}} = \alpha_1 h_1 + \alpha_2 h_2 + \alpha_3 h_3$$

Think of the $\alpha$ 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 \((x_1, y_1), \ldots, (x_m, y_m)\)

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

**• For** \(t = 1, 2, \ldots, T\):
  - Construct a set of sample weights \(D_t\) on \(\{1, 2, \ldots, m\}\)
  - Find a **weak hypothesis** \(h_t\) such that it has a small **weighted** error \(\epsilon_t\)

**• Construct a final hypothesis** \(H_{\text{final}}\)
An outline of Boosting

Given a training set \((x_1, y_1), \ldots, (x_m, y_m)\)

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

- For \(t = 1, 2, \ldots, T\):
  - Construct a set of sample weights \(D_t\) on \(\{1, 2, \ldots, m\}\)
  - Find a **weak hypothesis** \(h_t\) such that it has a small *weighted* error \(\varepsilon_t\)

- **Construct a final hypothesis** \(H_{final}\)

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

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

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 error $\epsilon_t$ 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 error $\epsilon_t$ 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:

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

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

$$= \frac{D_t(i)}{Z_t} \cdot \exp\left(-\alpha_t \cdot y_i h_t(x_i)\right)$$
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:

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

$$= \frac{D_t(i)}{Z_t} \cdot \exp \left( -\alpha_t \cdot y_i h_t(x_i) \right)$$

- Demote correctly predicted examples (if $\alpha_t > 0$)
- Promote incorrectly predicted examples (if $\alpha_t > 0$)
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:

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

$$= \frac{D_t(i)}{Z_t} \cdot \exp(-\alpha_t \cdot y_i h_t(x_i))$$
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:

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

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

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

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

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

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

$Z_t$: 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:

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

$$= \frac{D_t(i)}{Z_t} \cdot \exp \left( -\alpha_t \cdot y_i h_t(x_i) \right)$$

$Z_t$: 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 $\alpha_t$ in the final classifier.
An outline of Boosting

Given a training set \((x_1, y_1), \ldots, (x_m, y_m)\)

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

- For \(t = 1, 2, \ldots, T\):
  - Construct a distribution \(D_t\) on \(\{1, 2, \ldots, m\}\)
  - Find a weak hypothesis (rule of thumb) \(h_t\) such that it has a small weighted error \(\varepsilon_t\)

- Construct a final output \(H_{\text{final}}\)

Need to specify these two to get a complete algorithm
The final hypothesis

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

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

- Define the final hypothesis $H_{\text{final}}$ as

$$H_{\text{final}}(x) = \text{sgn} \left( \sum_{t} \alpha_t h_t(x) \right)$$
AdaBoost: The full algorithm

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

1. Initialize \(D_1(i) = 1/m\) for all \(i = 1, 2, \ldots, m\)
2. For \(t = 1, 2, \ldots T:\)
AdaBoost: The full algorithm

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

1. Initialize \(D_1(i) = 1/m\) for all \(i = 1, 2, \cdots, m\)

2. For \(t = 1, 2, \cdots T\):
   1. Find a classifier \(h_t\) whose \textit{weighted classification error} is better than chance

\(T\): a parameter to the learner
AdaBoost: The full algorithm

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

1. Initialize \(D_1(i) = 1/m\) for all \(i = 1, 2, \cdots, m\)

2. For \(t = 1, 2, \cdots, T\):
   1. Find a classifier \(h_t\) whose *weighted classification error* is better than chance
   2. 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 \((x_1, y_1), \ldots, (x_m, y_m)\)
Instances \(x_i \in X\) labeled with \(y_i \in \{-1, +1\}\)

1. Initialize \(D_1(i) = 1/m\) for all \(i = 1, 2, \ldots, m\)

2. For \(t = 1, 2, \ldots, T\):
   1. Find a classifier \(h_t\) whose \textit{weighted classification error} is better than chance
   2. Compute its vote
      \[
      \alpha_t = \frac{1}{2} \ln \left( \frac{1 - \epsilon_t}{\epsilon_t} \right)
      \]
   3. Update the values of the weights for the training examples
      \[
      D_{t+1}(i) = \frac{D_t(i)}{Z_t} \cdot \exp\left( -\alpha_t \cdot y_i h_t(x_i) \right)
      \]
AdaBoost: The full algorithm

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

1. Initialize \(D_1(i) = 1/m\) for all \(i = 1, 2, \ldots, m\)

2. For \(t = 1, 2, \ldots, T\):
   1. Find a classifier \(h_t\) whose \textit{weighted classification error} is better than chance
   2. Compute its vote
      \[
      \alpha_t = \frac{1}{2} \ln \left( \frac{1 - \epsilon_t}{\epsilon_t} \right)
      \]
   3. Update the values of the weights for the training examples
      \[
      D_{t+1}(i) = \frac{D_t(i)}{Z_t} \cdot \exp \left( -\alpha_t \cdot y_i h_t(x_i) \right)
      \]

3. Return the final hypothesis
   \[
   H_{\text{final}}(x) = \text{sgn} \left( \sum_t \alpha_t h_t(x) \right)
   \]
Back to the toy example

\[ H_{\text{final}} = \alpha_1 + \alpha_2 + \alpha_3 \]
Back to the toy example

\[ H_{\text{final}} = \alpha_1 + \alpha_2 + \alpha_3 \]
Hint of Implementation with decision trees

• How to compute information gain for weighted examples? View them as fraction examples!

<table>
<thead>
<tr>
<th></th>
<th>T</th>
<th>H</th>
<th>W</th>
<th>Play?</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>S</td>
<td>H</td>
<td>H</td>
<td>W</td>
</tr>
<tr>
<td>2</td>
<td>S</td>
<td>H</td>
<td>H</td>
<td>S</td>
</tr>
<tr>
<td>3</td>
<td>O</td>
<td>H</td>
<td>H</td>
<td>W</td>
</tr>
<tr>
<td>4</td>
<td>R</td>
<td>M</td>
<td>H</td>
<td>W</td>
</tr>
<tr>
<td>5</td>
<td>R</td>
<td>C</td>
<td>N</td>
<td>W</td>
</tr>
</tbody>
</table>

Round 1:

Sample 1 2 3 4 5
- - + + +

$D_1$ 1/5 1/5 1/5 1/5 1/5

$p_+ = 1/5 + 1/5 + 1/5 = 3/5$, $p_- = 2/5$, entropy = ...

Round 2:

Sample 1 2 3 4 5
- - + + +

$D_2$ 2/8 1/8 1/8 3/8 1/8

$p_+ = 1/8 + 3/8 + 1/8 = 5/8$, $p_- = 3/8$, entropy = ...
Analyzing the training error

Theorem:

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

\[
\text{Training error}(H_{final}) \leq e^{-2\gamma^2 T}
\]
Analyzing the training error

Theorem:

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

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

Theorem:

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

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

**Theorem:**

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

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

- We have a weak learner
- Assuming non-zero lower bound for all $t$, which may not always be the case in practice
- 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 $0 < \gamma \leq \gamma_t$ for all $t$
- Then,

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

Proof is simple, see pointer on course website

[link](https://www.cs.utah.edu/~zhe/teach/cs6350-lectures.html)
Analyzing the training error

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

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

We have a weak learner
Assuming non-zero lower bound for all t, which may not always be the case in practice
As T increases, the training error drops exponentially
As long as we run enough # of iterations, we can perfectly fit the training data!

Proof is simple, see pointer on course website

https://www.cs.utah.edu/~zhe/teach/cs6350-lectures.html
Adaboost: Training error

The training error of the combined classifier decreases exponentially fast if the errors of the weak classifiers (the $\epsilon_t$) 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 $\epsilon_t$) are strictly better than chance.
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 $H_{\text{final}}$ becomes too “complex”
  - Think about Occam’s razor and overfitting
In practice
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
  - Can fail if the weak learners are too weak

- **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)
Ensemble Learning

• What is boosting?

• AdaBoost

• Bagging methods
  – Bagging and Random Forests
Bagging 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
Bagging

Short for Bootstrap aggregating [Breiman, 1994]

• Given a training set with m examples

• Repeat t = 1, 2, ⋯, T:
  – Draw m’ (≤ m) samples uniformly with replacement from the training set
  – Train a classifier (any classifier) C_t

• Construct final classifier by taking votes from each C_t
Bagging

Short for *Bootstrap aggregating* [Breiman, 1994]

- Given a training set with m examples
- Repeat t = 1, 2, ..., T:
  - Draw m' (≤ m) samples *uniformly with replacement* from the training set
  - Train a classifier (any classifier) C_i

There could be duplications! When m’=m, one set of m’ samples is called a **bootstrap sample**!

Dataset = \{x_1, x_2, x_3, x_4, x_5\}, m = 5, m’ = 5
\{x_1, x_1, x_2, x_4, x_2\}  (**bootstrap sample 1**)  
\{x_2, x_2, x_3, x_4, x_5\}  (**bootstrap sample 2**)  
\{x_1, x_3, x_5, x_5, x_4\}  (**bootstrap sample 3**)  
...

...
Bagging

Short for *Bootstrap aggregating* [Breiman, 1994]

- Given a training set with $m$ examples
- Repeat $t = 1, 2, \ldots, T$:
  - Draw $m' \leq m$ samples *uniformly 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 **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)
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 repetition
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 repetition

- 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 repetition

- 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
Bagged Decision Trees

• Draw T bootstrap sample sets of data
• Train trees on each sample → T trees
• Average prediction of trees on out-of-bag samples

Average prediction

\[
\frac{0.23 + 0.19 + 0.34 + 0.22 + 0.26 + \ldots + 0.31}{\# \text{ Trees}} = 0.24
\]
Bagged Decision Trees (pseudo code)

• Given a training set with m examples
• Repeat $t = 1, 2, \ldots, T$:
  – Draw m samples *uniformly with replacement* from the training set (i.e. a bootstrap sample), denoted by $X_t, Y_t$
  – Learn a decision tree $C_t$ from $X_t, Y_t$, using ID3 or CART or others (no need for pruning or depth/width limitation)
• Prediction: Vote or average T predictions for classification or regression, respectively.
Why it works? (bias + variance decomposition)

• Intuitively, more predictors ➔ more reliable/robust
• Brief sketch of theoretical justification
  – Denote the training data by D, the learned predictor by h, the target function by f.
  – Given a new test example \( x^* \) (fixed and constant)
  – Let us analyze the squared error averaged over all possible training data (i.e., expectation taken over training set D’s distribution)

\[
E_D \left[ f(x^*) - h(x^*) \right]^2
\]
Why it works? (bias + variance decomposition)

\[ E_D \left[ f(x^*) - h(x^*) \right]^2 \]

\[ = E_D \left[ f(x^*) - E_D[h(x^*)] + E_D[h(x^*)] - h(x^*) \right]^2 \]

\[ = E_D \left[ f(x^*) - E_D[h(x^*)] \right]^2 + E_D \left[ E_D[h(x^*)] - h(x^*) \right]^2 \]

\[ + E_D \left[ (f(x^*) - E_D[h(x^*)]) \cdot (E_D[h(x^*)] - h(x^*)) \right] \cdot 2 \]
Why it works? (bias + variance decomposition)

\[
ED \left[ f(x^*) - h(x^*) \right]^2 \\
= ED \left[ f(x^*) - ED[h(x^*)] + ED[h(x^*)] - h(x^*) \right]^2 \\
= ED \left[ f(x^*) - ED[h(x^*)] \right]^2 + ED \left[ ED[h(x^*)] - h(x^*) \right]^2 \\
+ ED \left[ (f(x^*) - ED[h(x^*)]) \cdot (ED[h(x^*)] - h(x^*)) \right] = 0 \quad \text{Why?}
\]
Why it works? (bias + variance decomposition)

\[
E_D\left[ f(x^*) - h(x^*) \right]^2
= E_D\left[ f(x^*) - E_D[h(x^*)] + E_D[h(x^*)] - h(x^*) \right]^2
= E_D\left[ f(x^*) - E_D[h(x^*)] \right]^2 + E_D\left[ E_D[h(x^*)] - h(x^*) \right]^2
+ E_D\left[ (f(x^*) - E_D[h(x^*)]) \cdot (E_D[h(x^*)] - h(x^*)) \right]

= E_D\left[ f(x^*) - E_D[h(x^*)] \right]^2 + E_D\left[ E_D[h(x^*)] - h(x^*) \right]^2
= \left( f(x^*) - E_D[h(x^*)] \right)^2 + \text{Var}_D[h(x^*)]
\]
Why it works? (bias + variance decomposition)

\[
E_D\left[ f(x^*) - h(x^*) \right]^2
\]

\[
= E_D\left[ f(x^*) - E_D[h(x^*)] + E_D[h(x^*)] - h(x^*) \right]^2
\]

\[
= E_D\left[ f(x^*) - E_D[h(x^*)] \right]^2 + E_D\left[ E_D[h(x^*)] - h(x^*) \right]^2
+ E_D\left[ (f(x^*) - E_D[h(x^*)]) \cdot (E_D[h(x^*)] - h(x^*)) \right]
\]

\[
= E_D\left[ f(x^*) - E_D[h(x^*)] \right]^2 + E_D\left[ E_D[h(x^*)] - h(x^*) \right]^2
\]

\[
= \left( f(x^*) - E_D[h(x^*)] \right)^2 + \text{Var}_D[h(x^*)]
\]

Bias

Variance

Trade-off

The more complex the model, the better it can fit the training data \(\rightarrow\) often the smaller the bias, but the larger the variance! Why: a small change of training data will lead to completely different learning result and prediction!
Why it works? (bias + variance decomposition)

\[ E_D \left[ f(x^*) - h(x^*) \right]^2 \]

\[ = \left( f(x^*) - E_D[h(x^*)] \right)^2 + \text{Var}_D[h(x^*)] \]

Trade-off

Bias

Variance

- A decision tree tends to overfit, so has a small bias, but large variance!
- Bagging averages a set of decision trees, to maintains similar bias but largely reduces variance!
Why it works? (bias + variance decomposition)

\[
ED \left[ f(x^*) - h(x^*) \right]^2
\]

\[
= (f(x^*) - ED[h(x^*)])^2 + Var_D[h(x^*)]
\]

Bias

Variance

- A decision tree tends to overfit, so has a small bias, but large variance!
- Bagging averages a set of decision trees, to maintains similar bias but largely reduces variance!

Note: for other types of error, say, 0-1, a similar bias+variance decomposition can be derived!


Problem of bagged trees

- When a small set of features are strongly indicative, they will be **consistently selected** by most of the trees, so the bagged trees are **strongly correlated**, and the variance cannot be effectively reduced.

\[
ED[f(x^*) - h(x^*)]^2 = (f(x^*) - ED[h(x^*)])^2 + Var_D[h(x^*)]
\]

Trade-off

- Bias
- Variance
Random Forests (Bagged Trees++)

- Draw T bootstrap samples of data
- **Draw a subset of available attributes at each split**
- Train trees on each sample/attribute set → T trees
- Average prediction of trees on out-of-bag samples

\[
\text{Average prediction} = \frac{0.23 + 0.19 + 0.34 + 0.22 + 0.26 + \ldots + 0.31}{\# \text{ Trees}} = 0.24
\]
Random Forests (Pseudo Code)

- Given a training set with m examples, feature set F
- Repeat t = 1, 2, ..., T:
  - Draw m samples uniformly with replacement from the training set (i.e. a bootstrap sample), denoted by $X_t$, $Y_t$
  - Call $\text{RandTreeLearn}(X_t, Y_t, F)$ to Learn a decision tree $C_t$ from $X_t$, $Y_t$ (no need for pruning or depth/width limitation)
- Prediction: Vote or average T predictions for classification or regression, respectively.

- $\text{RandTreeLearn}(X_t, Y_t, A)$
  - At each node:
    - Randomly sample a very small subset of A, denoted by G ($|G| < |A|$)
    - Select the best feature in G to perform split (with gain based on entropy, gini-index, ...)
    - $\text{RandTreeLearn}(X_t', Y_t', A\{\text{selected feature}\})$
Ensemble Learning: What have we seen?

• What is boosting?
  – Does weak learnability imply strong learnability?

• AdaBoost
  – Intuition
  – The algorithm
  – Why does it work

• Bagging methods
  – Bagging and Random Forests