1. Weak versus strong learning
PAC learning

Learning protocol
PAC learning

Learning protocol

- Get objects $x_1, \ldots, x_n \sim_{\text{iid}} P$, each paired with label $c(x_i)$. 

Probably approximately correct (PAC) learning (Valiant, 1984):

For any $\epsilon, \delta \in (0, 1)$ and any distribution $P$, learner requires $\text{poly}(1/\epsilon, 1/\delta, \ldots)$ sample size and time to produce hypothesis $h: X \rightarrow \{0, 1\}$ such that

$$\Pr_{x_1, \ldots, x_n \sim_{\text{iid}} P} \left[ \Pr_{x \sim P} (h(x) \neq c(x)) \leq \epsilon \right] \geq 1 - \delta.$$
PAC learning

Learning protocol

- Get objects $x_1, \ldots, x_n \sim \text{iid } P$, each paired with label $c(x_i)$.
- Using a procedure that takes input $(x_1, c(x_1)), \ldots, (x_n, c(x_n))$ and runs in time $t$, you pick a hypothesis $h : \mathcal{X} \to \{0, 1\}$. 

Probably approximately correct (PAC) learning (Valiant, 1984):

For any $\epsilon, \delta \in (0, 1)$ and any distribution $P$, learner requires $\text{poly}(1/\epsilon, 1/\delta, \ldots)$ sample size and time to produce hypothesis $h$ such that

$$\Pr_{x_1, \ldots, x_n \sim \text{iid } P} \left[ \Pr_{x \sim P} (h(x) \neq c(x)) \leq \epsilon \right] \geq 1 - \delta.$$ 

The probability of successful learning. 

Error rate.
PAC learning

Learning protocol

- Get objects $x_1, \ldots, x_n \sim_{\text{iid}} P$, each paired with label $c(x_i)$.
- Using a procedure that takes input $(x_1, c(x_1)), \ldots, (x_n, c(x_n))$ and runs in time $t$, you pick a hypothesis $h : \mathcal{X} \rightarrow \{0, 1\}$.
- Get quiz object $x \sim P$ (independent of all previous $x_i$).
Learning protocol

- Get objects $x_1, \ldots, x_n \sim_{\text{iid}} P$, each paired with label $c(x_i)$.
- Using a procedure that takes input $(x_1, c(x_1)), \ldots, (x_n, c(x_n))$ and runs in time $t$, you pick a hypothesis $h: \mathcal{X} \to \{0, 1\}$.
- Get quiz object $x \sim P$ (independent of all previous $x_i$).
- Evaluate: is $h(x) = c(x)$?
PAC learning

Learning protocol

- Get objects $x_1, \ldots, x_n \sim_{\text{iid}} P$, each paired with label $c(x_i)$.
- Using a procedure that takes input $(x_1, c(x_1)), \ldots, (x_n, c(x_n))$ and runs in time $t$, you pick a hypothesis $h: \mathcal{X} \rightarrow \{0, 1\}$.
- Get quiz object $x \sim P$ (independent of all previous $x_i$).
- Evaluate: is $h(x) = c(x)$?

Probably approximately correct (PAC) learning (Valiant, 1984):
For any $\epsilon, \delta \in (0, 1)$ and any distribution $P$, learner requires $\text{poly}(1/\epsilon, 1/\delta, \ldots)$ sample size and time to produce hypothesis $h$ such that

$$\Pr_{x_1, \ldots, x_n \sim_{\text{iid}} P} \left[ \Pr_{x \sim P} (h(x) \neq c(x)) \leq \epsilon \right] \geq 1 - \delta.$$
**Weak learning** (Kearns and Valiant, 1988):
There exists a constant $\epsilon_0 < 1/2$ such that, for any $\delta \in (0, 1)$ and any distribution $P$, learner requires $\text{poly}(1/\delta, \ldots)$ sample size and time to produce hypothesis $h$ such that

$$\Pr_{x^{(1)}, \ldots, x^{(n)} \sim_{\text{iid}} P} \left[ \Pr_{x \sim P} (h(x) \neq c(x)) \leq \epsilon_0 \right] \geq 1 - \delta.$$
**Weak learning** (Kearns and Valiant, 1988):
There exists a constant $\epsilon_0 < 1/2$ such that, for any $\delta \in (0, 1)$ and any distribution $P$, learner requires $\text{poly}(1/\delta, \ldots)$ sample size and time to produce hypothesis $h$ such that

$$\Pr_{x^{(1)}, \ldots, x^{(n)} \sim \text{iid } P} \left[ \Pr_{x \sim P} (h(x) \neq c(x)) \leq \epsilon_0 \right] \geq 1 - \delta.$$ 

Only need to be able to return a hypothesis that is better than guessing.
Weak learning versus strong learning

**Weak learning** (Kearns and Valiant, 1988):
There exists a constant $\epsilon_0 < 1/2$ such that, for any $\delta \in (0, 1)$ and any distribution $P$, learner requires $\text{poly}(1/\delta, \ldots)$ sample size and time to produce hypothesis $h$ such that

$$\Pr_{x^{(1)}, \ldots, x^{(n)} \sim \text{iid} P} \left[ \Pr_{x \sim P} (h(x) \neq c(x)) \leq \epsilon_0 \right] \geq 1 - \delta.$$ 

Only need to be able to return a hypothesis that is better than guessing.

**In practice**: Weak learning seems to be easier than **strong** (i.e., PAC) learning (e.g., “If $\geq 5\%$ of the e-mail characters are dollar signs, then it’s spam.”).
Weak learning versus strong learning

**Weak learning** (Kearns and Valiant, 1988):
There exists a constant $\epsilon_0 < 1/2$ such that, for any $\delta \in (0, 1)$ and any distribution $P$, learner requires $\text{poly}(1/\delta, \ldots)$ sample size and time to produce hypothesis $h$ such that

$$\Pr_{x^{(1)}, \ldots, x^{(n)} \sim \text{iid } P} \left[ \Pr_{x \sim P} (h(x) \neq c(x)) \leq \epsilon_0 \right] \geq 1 - \delta.$$  

Only need to be able to return a hypothesis that is better than guessing.

In practice: Weak learning seems to be easier than **strong** (i.e., PAC) learning (e.g., “If $\geq 5\%$ of the e-mail characters are dollar signs, then it’s spam.”).

Kearns and Valiant asked:

*If it is possible to weakly learn every concept from a concept class $C$, then is it possible to strongly learn every concept from $C$?*
**Boosting:** using an algorithm for weak learning (i.e., a *weak learner*) to achieve strong learning.
Boosting: using an algorithm for weak learning (i.e., a weak learner) to achieve strong learning.

Basic template for boosting algorithms:

Input: \((x_1, y_1), \ldots, (x_n, y_n) \in \mathcal{X} \times \{-1, +1\}\)

For \(t = 1, 2, \ldots, T\):

1. Choose distribution \(D_t\) over training examples.
2. Use weak learner with \(D_t\) to get weak hypothesis \(h_t\).

Return: single “ensemble” hypothesis based on \(h_1, \ldots, h_T\).
1988  Kearns and Valiant ask whether boosting is theoretically possible.
Boosting: history

1988  Kearns and Valiant ask whether boosting is theoretically possible.

1989  Schapire creates boosting algorithm; solves Kearns and Valiant's problem.
Boosting: history

1988 Kearns and Valiant ask whether boosting is theoretically possible.

1989 Schapire creates boosting algorithm; solves Kearns and Valiant's problem.

1990 Freund creates optimal boosting algorithm (Boost-by-majority).

Winner of 2004 ACM Paris Kanellakis Award: For their "seminal work and distinguished contributions [...] to the development of the theory and practice of boosting, a general and provably effective method of producing arbitrarily accurate prediction rules by combining weak learning rules"; specifically, for AdaBoost, which "can be used to significantly reduce the error of algorithms used in statistical analysis, spam filtering, fraud detection, optical character recognition, and market segmentation, among other applications".
Boosting: history

1988 Kearns and Valiant ask whether boosting is theoretically possible.

1989 Schapire creates boosting algorithm; solves Kearns and Valiant's problem.

1990 Freund creates optimal boosting algorithm (Boost-by-majority).

1988 Kearns and Valiant ask whether boosting is theoretically possible.
1989 Schapire creates boosting algorithm; solves Kearns and Valiant's problem.
1990 Freund creates optimal boosting algorithm (Boost-by-majority).
1995 Freund and Schapire create *AdaBoost*—a boosting algorithm with practical advantages over previous boosting algorithms.

Winner of 2004 ACM Paris Kanellakis Award:
For their "seminal work and distinguished contributions [...] to the development of the theory and practice of boosting, a general and provably effective method of producing arbitrarily accurate prediction rules by combining weak learning rules"; specifically, for AdaBoost, which "can be used to significantly reduce the error of algorithms used in statistical analysis, spam filtering, fraud detection, optical character recognition, and market segmentation, among other applications."
1988 Kearns and Valiant ask whether boosting is theoretically possible.

1989 Schapire creates boosting algorithm; solves Kearns and Valiant's problem.

1990 Freund creates optimal boosting algorithm (Boost-by-majority).


1995 Freund and Schapire create AdaBoost—a boosting algorithm with practical advantages over previous boosting algorithms.

Winner of 2004 ACM Paris Kanellakis Award:

*For their “seminal work and distinguished contributions [...] to the development of the theory and practice of boosting, a general and provably effective method of producing arbitrarily accurate prediction rules by combining weak learning rules”; specifically, for AdaBoost, which “can be used to significantly reduce the error of algorithms used in statistical analysis, spam filtering, fraud detection, optical character recognition, and market segmentation, among other applications”.*
AdaBoost

**input** Training examples $(\mathbf{x}_1, y_1), \ldots, (\mathbf{x}_n, y_n) \in \mathcal{X} \times \{-1, +1\}$

1: **initialize** $D_1(i) := 1/n$ for each $i = 1, \ldots, n$ (a probability distribution).
2: **for** $t = 1, \ldots, T$ **do**
3: Give $D_t$-weighted examples to WL; get back $h_t: \mathcal{X} \rightarrow \{-1, +1\}$.
4: Update weights:

\[
\begin{align*}
\gamma_t & := \sum_{i=1}^{n} D_t(i) \cdot 1\{h_t(\mathbf{x}_i) = y_i\} - \frac{1}{2} \\
\alpha_t & := \frac{1}{2} \ln \frac{1 + 2\gamma_t}{1 - 2\gamma_t} \quad \text{(weight of } h_t) \\
D_{t+1}(i) & := \frac{D_t(i) \exp \left( -\alpha_t \cdot y_i h_t(\mathbf{x}_i) \right)}{Z_t} \quad \text{for } i = 1, \ldots, n,
\end{align*}
\]

where $Z_t > 0$ is normalizer that makes $D_{t+1}$ a probability distribution.
5: **end for**
6: **return** Final hypothesis $\hat{h}(\mathbf{x}) := \text{sign} \left( \sum_{t=1}^{T} \alpha_t \cdot h_t(\mathbf{x}) \right)$.

(Let $\text{sign}(z) := 1$ if $z > 0$ and $\text{sign}(z) := -1$ if $z \leq 0$.)
Regarding $D_t$ as a distribution over training examples $(x_1, y_1), \ldots, (x_n, y_n)$,

$$\gamma_t = \Pr_{(x,y) \sim D_t} (h_t(x) = y) - \frac{1}{2} \quad \text{(i.e., accuracy minus 1/2).}$$
Regarding $D_t$ as a distribution over training examples $(x_1, y_1), \ldots, (x_n, y_n)$,

$$\gamma_t = \Pr_{(x,y) \sim D_t} (h_t(x) = y) - \frac{1}{2} \quad \text{(i.e., accuracy minus 1/2)}.$$

**Hypothesis weights** $\alpha_t = \frac{1}{2} \ln \frac{1+2\gamma_t}{1-2\gamma_t}$:
Interpretation

Regarding $D_t$ as a distribution over training examples $(x_1, y_1), \ldots, (x_n, y_n)$,

$$\gamma_t = \Pr_{(x, y) \sim D_t} (h_t(x) = y) - \frac{1}{2}$$

(i.e., accuracy minus $1/2$).

Hypothesis weights $\alpha_t = \frac{1}{2} \ln \frac{1 + 2\gamma_t}{1 - 2\gamma_t}$.

Example weights $D_{t+1}(i)$:

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

Regarding $D_t$ as a distribution over training examples $(x_1, y_1), \ldots, (x_n, y_n)$,

$$\gamma_t = \Pr_{(x, y) \sim D_t}(h_t(x) = y) - \frac{1}{2}$$

(i.e., accuracy minus $1/2$).

**Hypothesis weights** $\alpha_t = \frac{1}{2} \ln \frac{1 + 2\gamma_t}{1 - 2\gamma_t}$:

**Example weights** $D_{t+1}(i)$:

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

Which data get higher weight in $D_{t+1}$?
Example: AdaBoost with decision stumps

Weak learner: ERM with $\mathcal{H} =$ “decision stumps” on $\mathbb{R}^2$ (decision trees with one axis-aligned split). Straightforward to handle importance weights in ERM.

(Example from Figures 1.1 and 1.2 of Schapire & Freund text.)
Example: execution of AdaBoost

\[ D_1 \]

\[ h_1 \] \[ \gamma_1 = 0 \]
\[ \alpha_1 = 0.42 \]
\[ \gamma_2 = 0.29 \]
\[ \alpha_2 = 0.65 \]
\[ \gamma_3 = 0.36 \]
\[ \alpha_3 = 0.92 \]
Example: execution of AdaBoost

\[ D_1 \]

\[ h_1 \]

\[ \gamma_1 = 0.20, \alpha_1 = 0.42 \]
Example: execution of AdaBoost

\[ h_1 \]

\[ \gamma_1 = 0.20, \alpha_1 = 0.42 \]
Example: execution of AdaBoost

\[ D_1 \]

\[ D_2 \]

\[ h_1 \]

\[ \gamma_1 = 0.20, \alpha_1 = 0.42 \]

\[ h_2 \]

\[ \gamma_2 = 0.29, \alpha_2 = 0.65 \]
Example: execution of AdaBoost

\[ D_1 \]

\[ D_2 \]

\[ D_3 \]

\[ h_1 \]
\[ \gamma_1 = 0.20, \alpha_1 = 0.42 \]

\[ h_2 \]
\[ \gamma_2 = 0.29, \alpha_2 = 0.65 \]
Example: execution of AdaBoost

\begin{align*}
D_1 & \quad D_2 & \quad D_3 \\
\begin{array}{c}
\begin{array}{c}
+ \\
+ \\
+ \\
- \\
+ \\
- \\
+ \\
- \\
+ \\
- \\
\end{array} \\
\begin{array}{c}
+ \\
+ \\
+ \\
- \\
+ \\
- \\
+ \\
- \\
+ \\
- \\
\end{array} \\
\begin{array}{c}
+ \\
+ \\
+ \\
- \\
+ \\
- \\
+ \\
- \\
+ \\
- \\
\end{array} \\
\begin{array}{c}
+ \\
+ \\
+ \\
- \\
+ \\
- \\
+ \\
- \\
+ \\
- \\
\end{array}
\end{array} \\
\begin{array}{c}
\gamma_1 = 0.20, \quad \alpha_1 = 0.42 \\
\gamma_2 = 0.29, \quad \alpha_2 = 0.65 \\
\gamma_3 = 0.36, \quad \alpha_3 = 0.92
\end{array}
\end{align*}
Example: final hypothesis from AdaBoost

\[ \hat{h}(x) = \text{sign}(0.42 h_1(x) + 0.65 h_2(x) + 0.92 h_3(x)) \]

(Zero training error rate!)

\[ \gamma_1 = 0.20, \quad \alpha_1 = 0.42 \]

\[ \gamma_2 = 0.29, \quad \alpha_2 = 0.65 \]

\[ \gamma_3 = 0.36, \quad \alpha_3 = 0.92 \]
Example: final hypothesis from AdaBoost

Final hypothesis
\( \hat{h}(x) = \text{sign}(0.42h_1(x) + 0.65h_2(x) + 0.92h_3(x)) \)

(Zero training error rate!)
Test error rates of C4.5 and AdaBoost on several classification problems. Each point represents a single classification problem/dataset from UCI repository.

C4.5 = popular algorithm for learning decision trees.

(Figure 1.3 from Schapire & Freund text.)
Training error rate of final hypothesis

Recall $\gamma_t := \Pr_{(x,y) \sim D_t}(h_t(x) = y) - 1/2$.

Training error rate of final hypothesis from AdaBoost:

$$\frac{1}{n} \sum_{i=1}^{n} 1\{\hat{h}(x_i) \neq y_i\} \leq \exp \left(-2 \sum_{t=1}^{T} \gamma_t^2 \right).$$
Recall $\gamma_t := \Pr_{(x,y) \sim D_t} (h_t(x) = y) - 1/2$.

**Training error rate of final hypothesis from AdaBoost:**

$$\frac{1}{n} \sum_{i=1}^{n} 1\{\hat{h}(x_i) \neq y_i\} \leq \exp \left( -2 \sum_{t=1}^{T} \gamma_t^2 \right).$$

If average $\bar{\gamma}^2 := \frac{1}{T} \sum_{t=1}^{T} \gamma_t^2 > 0$, then training error rate is $\leq \exp \left( -2\bar{\gamma}^2 T \right)$. 

Recall $\gamma_t := \Pr_{(x,y) \sim D_t}(h_t(x) = y) - 1/2$.

**Training error rate of final hypothesis from AdaBoost:**

$$\frac{1}{n} \sum_{i=1}^{n} 1\{\hat{h}(x_i) \neq y_i\} \leq \exp \left(-2 \sum_{t=1}^{T} \gamma_t^2 \right).$$

If average $\bar{\gamma}^2 := \frac{1}{T} \sum_{t=1}^{T} \gamma_t^2 > 0$, then training error rate is $\leq \exp \left(-2\bar{\gamma}^2 T \right)$.

**“AdaBoost” = “Adaptive Boosting”**

Some $\gamma_t$ could be small, even negative—only care about overall average $\bar{\gamma}^2$. 


Recall $\gamma_t := \Pr_{(x,y) \sim D_t}(h_t(x) = y) - 1/2$.

**Training error rate of final hypothesis from AdaBoost:**

$$\frac{1}{n} \sum_{i=1}^{n} 1\{\hat{h}(x_i) \neq y_i\} \leq \exp \left( -2 \sum_{t=1}^{T} \gamma_t^2 \right).$$

If average $\bar{\gamma}^2 := \frac{1}{T} \sum_{t=1}^{T} \gamma_t^2 > 0$, then training error rate is $\leq \exp \left( -2\bar{\gamma}^2 T \right)$.

“AdaBoost” = “Adaptive Boosting”

Some $\gamma_t$ could be small, even negative—only care about overall average $\bar{\gamma}^2$.

**What about true error rate?**
Combining hypotheses

Let $\mathcal{H}$ be the *hypothesis class* used by the weak learner WL.
Combining hypotheses

Let $\mathcal{H}$ be the \textit{hypothesis class} used by the weak learner WL.

The hypothesis class used by AdaBoost is

$$
\mathcal{H}_T := \left\{ \mathbf{x} \mapsto \text{sign} \left( \sum_{t=1}^{T} \alpha_t h_t(\mathbf{x}) \right) : h_1, \ldots, h_T \in \mathcal{H}, \alpha_1, \ldots, \alpha_T \in \mathbb{R} \right\}
$$

i.e., thresholding linear combinations of $T$ hypotheses from $\mathcal{H}$. 

Combining hypotheses

Let $\mathcal{H}$ be the \textit{hypothesis class} used by the weak learner WL.

The hypothesis class used by AdaBoost is

$$\mathcal{H}_T := \left\{ x \mapsto \text{sign} \left( \sum_{t=1}^{T} \alpha_t h_t(x) \right) : h_1, \ldots, h_T \in \mathcal{H}, \alpha_1, \ldots, \alpha_T \in \mathbb{R} \right\}$$

i.e., thresholding linear combinations of $T$ hypotheses from $\mathcal{H}$.

\textbf{Fact}: VC dimension of $\mathcal{H}_T$ is $O(T \cdot \text{VC}(\mathcal{H}))$. 

Combining hypotheses

Let \( \mathcal{H} \) be the *hypothesis class* used by the weak learner WL.

The hypothesis class used by AdaBoost is

\[
\mathcal{H}_T := \left\{ \mathbf{x} \mapsto \text{sign} \left( \sum_{t=1}^{T} \alpha_t h_t(\mathbf{x}) \right) : h_1, \ldots, h_T \in \mathcal{H}, \alpha_1, \ldots, \alpha_T \in \mathbb{R} \right\}
\]

i.e., thresholding linear combinations of \( T \) hypotheses from \( \mathcal{H} \).

**Fact:** VC dimension of \( \mathcal{H}_T \) is \( O(T \cdot \text{VC}(\mathcal{H})) \).

**Theoretical guarantee:**
If WL uses hypothesis class \( \mathcal{H} \), then with high probability over choice of \((\mathbf{x}_1, y_1), \ldots, (\mathbf{x}_n, y_n) \sim_{iid} P\), AdaBoost returns hypothesis \( \hat{h} \in \mathcal{H}_T \) satisfying

\[
\Pr_{(\mathbf{x}, y) \sim P} (\hat{h}(\mathbf{x}) \neq y) \leq \exp \left( -2\gamma^2 T \right) + O \left( \sqrt{\frac{T \cdot \text{VC}(\mathcal{H})}{n}} \right).
\]

Training error rate
Combining hypotheses

Let $\mathcal{H}$ be the *hypothesis class* used by the weak learner WL.

The hypothesis class used by AdaBoost is

$$
\mathcal{H}_T := \left\{ \mathbf{x} \mapsto \text{sign} \left( \sum_{t=1}^{T} \alpha_t h_t(\mathbf{x}) \right) : h_1, \ldots, h_T \in \mathcal{H}, \alpha_1, \ldots, \alpha_T \in \mathbb{R} \right\}
$$

i.e., thresholding linear combinations of $T$ hypotheses from $\mathcal{H}$.

**Fact:** VC dimension of $\mathcal{H}_T$ is $O(T \cdot \text{VC}(\mathcal{H}))$.

**Theoretical guarantee:**
If WL uses hypothesis class $\mathcal{H}$, then with high probability over choice of $(\mathbf{x}_1, y_1), \ldots, (\mathbf{x}_n, y_n) \sim_{\text{iid}} P$, AdaBoost returns hypothesis $\hat{h} \in \mathcal{H}_T$ satisfying

$$
\Pr_{(\mathbf{x}, y) \sim P} (\hat{h}(\mathbf{x}) \neq y) \leq \exp \left( -2\bar{\gamma}^2 T \right) + O \left( \sqrt{\frac{T \cdot \text{VC}(\mathcal{H})}{n}} \right).
$$

**Training error rate**

Theory suggests danger of overfitting when $T$ is large relative to $n$. 
Combining hypotheses

Let $\mathcal{H}$ be the *hypothesis class* used by the weak learner WL.

The hypothesis class used by AdaBoost is

$$\mathcal{H}_T := \left\{ \mathbf{x} \mapsto \text{sign} \left( \sum_{t=1}^{T} \alpha_t h_t(\mathbf{x}) \right) : h_1, \ldots, h_T \in \mathcal{H}, \alpha_1, \ldots, \alpha_T \in \mathbb{R} \right\}$$

i.e., thresholding linear combinations of $T$ hypotheses from $\mathcal{H}$.

**Fact:** VC dimension of $\mathcal{H}_T$ is $O(T \cdot \text{VC}(\mathcal{H}))$.

**Theoretical guarantee:**
If WL uses hypothesis class $\mathcal{H}$, then with high probability over choice of $(\mathbf{x}_1, y_1), \ldots, (\mathbf{x}_n, y_n) \sim_{\text{iid}} P$, AdaBoost returns hypothesis $\hat{h} \in \mathcal{H}_T$ satisfying

$$\Pr_{(\mathbf{x}, y) \sim P} (\hat{h}(\mathbf{x}) \neq y) \leq \exp \left( -2\gamma^2 T \right) + O \left( \sqrt{\frac{T \cdot \text{VC}(\mathcal{H})}{n}} \right).$$

**Theory suggests danger of overfitting when $T$ is large relative to $n$.**
Indeed, this does happen sometimes . . . but often not!
A typical run of boosting

AdaBoost + C4.5 on “letters” dataset.

![Graph showing training and test error rates.

Error rate vs. number of rounds.]

- Training error rate is zero after just five rounds,
- but test error rate continues to decrease, even up to 1000 rounds!

(Figure 1.7 from Schapire & Freund text)
Boosting the margin

Final hypothesis from AdaBoost:

\[ \hat{f}(x) = \text{sign} \left( \frac{\sum_{t=1}^{T} \alpha_t \cdot h_t(x)}{\sum_{t=1}^{T} |\alpha_t|} \right), \quad x \in \mathcal{X}. \]

\[ g(x) \in [-1, +1] \]

Call \( y \cdot g(x) \in [-1, +1] \) the (normalized) margin achieved on example \((x, y)\). (Similar to, but not the same as, SVM margins.)
Boosting the margin

Final hypothesis from AdaBoost:

\[
\hat{f}(x) = \text{sign} \left( \frac{\sum_{t=1}^{T} \alpha_t \cdot h_t(x)}{\sum_{t=1}^{T} |\alpha_t|} \right), \quad x \in \mathcal{X}.
\]

\(g(x) \in [-1, +1]\)

Call \(y \cdot g(x) \in [-1, +1]\) the (normalized) margin achieved on example \((x, y)\).
(Similar to, but not the same as, SVM margins.)

Margins theory (Schapire, Freund, Bartlett, and Lee, 1998):

- Larger margins ⇒ better resistance to overfitting, independent of \(T\).
- AdaBoost tends to increase margins on training examples.
Boosting the margin

Final hypothesis from AdaBoost:

\[
\hat{f}(\mathbf{x}) = \text{sign} \left( \frac{\sum_{t=1}^{T} \alpha_t \cdot h_t(\mathbf{x})}{\sum_{t=1}^{T} |\alpha_t|} \right), \quad \mathbf{x} \in \mathcal{X}.
\]

\[g(\mathbf{x}) \in [-1, +1]\]

Call \(y \cdot g(\mathbf{x}) \in [-1, +1]\) the (normalized) margin achieved on example \((\mathbf{x}, y)\). (Similar to, but not the same as, SVM margins.)

**Margins theory** (Schapire, Freund, Bartlett, and Lee, 1998):

- Larger margins \(\Rightarrow\) better resistance to overfitting, independent of \(T\).
- AdaBoost tends to increase margins on training examples.

On “letters” dataset:

<table>
<thead>
<tr>
<th>(T = 5)</th>
<th>(T = 100)</th>
<th>(T = 1000)</th>
</tr>
</thead>
<tbody>
<tr>
<td>training error rate</td>
<td>0.0%</td>
<td>0.0%</td>
</tr>
<tr>
<td>test error rate</td>
<td>8.4%</td>
<td>3.3%</td>
</tr>
<tr>
<td>% margins (\leq 0.5)</td>
<td>7.7%</td>
<td>0.0%</td>
</tr>
<tr>
<td>min. margin</td>
<td>0.14</td>
<td>0.52</td>
</tr>
</tbody>
</table>
Implicit feature mapping based on hypothesis class $\mathcal{H}$:

$$x \mapsto \phi(x) := (h(x) : h \in \mathcal{H}) \in \{-1, +1\}^\mathcal{H}$$

(possibly infinite dimensional!).
Implicit feature mapping based on hypothesis class $\mathcal{H}$:

$$x \mapsto \phi(x) := (h(x) : h \in \mathcal{H}) \in \{-1, +1\}^\mathcal{H}$$

(possibly infinite dimensional!).

AdaBoost’s final hypothesis is a linear classifier in $\{-1, +1\}^\mathcal{H}$:

$$\hat{h}(x) = \text{sign} \left( \sum_{t=1}^{T} \alpha_t h_t(x) \right) = \text{sign} \left( \sum_{h \in \mathcal{H}} w_h h(x) \right) = \text{sign} \left( w^T \phi(x) \right)$$

where

$$w_f := \sum_{t=1}^{T} \alpha_t \cdot 1\{h_t = h\}, \quad h \in \mathcal{H}.$$
Exponential loss

AdaBoost is a particular “coordinate descent” algorithm for

$$\min_{\mathbf{w} \in \mathbb{R}^H} \frac{1}{n} \sum_{i=1}^{n} \exp \left( -y_i \phi(x_i)^\top \mathbf{w} \right).$$
Exponential loss

AdaBoost is a particular “coordinate descent” algorithm for

$$\min_{w \in \mathbb{R}^H} \frac{1}{n} \sum_{i=1}^{n} \exp \left(-y_i \phi(x_i)^T w \right).$$
Many variants of boosting:

- AdaBoost with different loss functions.
- Boosted decision trees = boosting + decision trees.
- Boosting algorithms for ranking and multi-class.
- Boosting algorithms that are robust to certain kinds of noise.
- ...

Many connections between boosting and other subjects:

- Game theory, online learning
- Geometry of information (replace $\| \cdot \|_2^2$ with relative entropy)
- Computational complexity
- ...

2. Face detection
Problem: Given an image, locate all of the faces.
Application: face detection

**Problem:** Given an image, locate all of the faces.

As a classification problem:

- Divide up images into patches (at varying scales, e.g., 24\times24, 48\times48).
- Learn classifier \( f: \mathcal{X} \rightarrow \mathcal{Y} \), where \( \mathcal{Y} = \{\text{face}, \text{not face}\} \).
Application: face detection

**Problem**: Given an image, locate all of the faces.

As a classification problem:

- Divide up images into patches (at varying scales, e.g., 24×24, 48×48).
- Learn classifier \( f : \mathcal{X} \rightarrow \mathcal{Y} \), where \( \mathcal{Y} = \{ \text{face, not face} \} \).

Many other things built on top of face detectors (e.g., face tracking, face recognizers).
**Application: face detection**

**Problem:** Given an image, locate all of the faces.

As a classification problem:

- Divide up images into patches (at varying scales, e.g., $24\times24$, $48\times48$).
- Learn classifier $f : \mathcal{X} \to \mathcal{Y}$, where $\mathcal{Y} = \{\text{face, not face}\}$.

Many other things built on top of face detectors (e.g., face tracking, face recognizers).

**Main problem:** how to make this **very fast**.
Face detector architecture by Viola & Jones (2001): major achievement in computer vision; detector actually usable in real-time.
Face detectors via AdaBoost [Viola & Jones, 2001]

**Face detector architecture by Viola & Jones (2001):** major achievement in computer vision; **detector actually usable in real-time.**

- Think of each image patch \((d \times d\)-pixel gray-scale) as a vector \(\mathbf{x} \in [0, 1]^{d^2}\).
Face detectors via AdaBoost [Viola & Jones, 2001]

Face detector architecture by Viola & Jones (2001): major achievement in computer vision; detector actually usable in real-time.

- Think of each image patch ($d \times d$-pixel gray-scale) as a vector $\mathbf{x} \in [0, 1]^{d^2}$.
- Used weak learning algorithm that picks linear classifiers $f_{\mathbf{w}, t}(\mathbf{x}) = \text{sign}(\langle \mathbf{w}, \mathbf{x} \rangle - t)$, where $\mathbf{w}$ has a very particular form:
Face detectors via AdaBoost [Viola & Jones, 2001]

**Face detector architecture by Viola & Jones (2001):** major achievement in computer vision; detector actually usable in real-time.

- Think of each image patch ($d \times d$-pixel gray-scale) as a vector $x \in [0, 1]^{d^2}$.
- Used weak learning algorithm that picks linear classifiers $f_{w,t}(x) = \text{sign}(\langle w, x \rangle - t)$, where $w$ has a very particular form:

$$\langle w, x \rangle = \text{average pixel value in black box} - \text{average pixel value in white box}$$
Face detectors via AdaBoost [Viola & Jones, 2001]

**Face detector architecture by Viola & Jones (2001):** major achievement in computer vision; detector actually usable in real-time.

- Think of each image patch \((d \times d\)-pixel gray-scale) as a vector \(x \in [0, 1]^{d^2}\).
- Used weak learning algorithm that picks linear classifiers
  \(f_{w,t}(x) = \text{sign}(\langle w, x \rangle - t)\), where \(w\) has a very particular form:

\[
\langle w, x \rangle = \text{average pixel value in black box} - \text{average pixel value in white box}
\]
Face detectors via AdaBoost [Viola & Jones, 2001]

Face detector architecture by Viola & Jones (2001): major achievement in computer vision; detector actually usable in real-time.

- Think of each image patch (\(d \times d\)-pixel gray-scale) as a vector \(x \in [0, 1]^{d^2}\).
- Used weak learning algorithm that picks linear classifiers 
  \(f_{w,t}(x) = \text{sign}(\langle w, x \rangle - t)\), where \(w\) has a very particular form:

\[
\langle w, x \rangle = \text{average pixel value in black box} - \text{average pixel value in white box}
\]
Face detector architecture by Viola & Jones (2001): major achievement in computer vision; detector actually usable in real-time.

- Think of each image patch ($d \times d$-pixel gray-scale) as a vector $x \in [0, 1]^{d^2}$.
- Used weak learning algorithm that picks linear classifiers $f_{w,t}(x) = \text{sign}(\langle w, x \rangle - t)$, where $w$ has a very particular form:

$$\langle w, x \rangle = \text{average pixel value in black box}$$
$$- \text{average pixel value in white box}$$
Face detectors via AdaBoost [Viola & Jones, 2001]

**Face detector architecture by Viola & Jones (2001):** major achievement in computer vision; detector actually usable in real-time.

- Think of each image patch ($d \times d$-pixel gray-scale) as a vector $x \in [0, 1]^{d^2}$.
- Used weak learning algorithm that picks linear classifiers $f_{w,t}(x) = \text{sign}(\langle w, x \rangle - t)$, where $w$ has a very particular form:

![Image of face patches and features](image-url)
Face detectors via AdaBoost [Viola & Jones, 2001]

Face detector architecture by Viola & Jones (2001): major achievement in computer vision; detector actually usable in real-time.

- Think of each image patch ($d \times d$-pixel gray-scale) as a vector $x \in [0, 1]^{d^2}$.
- Used weak learning algorithm that picks linear classifiers $f_{w,t}(x) = \text{sign}(\langle w, x \rangle - t)$, where $w$ has a very particular form:

![Diagram of face features](image)

- AdaBoost combines many “rules-of-thumb” of this form.
  - Very simple.
  - Extremely fast to evaluate via pre-computation.
Viola & Jones “integral image” trick

“Integral image” trick:

For every image, pre-compute

\[ s(r, c) = \text{sum of pixel values in rectangle from } (0, 0) \text{ to } (r, c) \]

(single pass through image).
“Integral image” trick:

For every image, pre-compute

\[ s(r, c) = \text{sum of pixel values in rectangle from } (0, 0) \text{ to } (r, c) \]

(single pass through image).

To compute inner product

\[ \langle w, x \rangle = \text{average pixel value in black box} \]
\[ - \text{average pixel value in white box} \]

just need to add and subtract a few \( s(r, c) \) values.
“Integral image” trick:

For every image, pre-compute

\[ s(r, c) = \text{sum of pixel values in rectangle from } (0, 0) \text{ to } (r, c) \]

(single pass through image).

To compute inner product

\[ \langle w, x \rangle = \text{average pixel value in black box} \]
\[ - \text{average pixel value in white box} \]

just need to add and subtract a few \( s(r, c) \) values.

⇒ Evaluating “rules-of-thumb” classifiers is extremely fast.
Viola & Jones cascade architecture

**Problem:** severe class imbalance (most patches don’t contain a face).
Problem: severe class imbalance (most patches don’t contain a face).

Solution: Train several classifiers (each using AdaBoost), and arrange in a special kind of decision list called a cascade:

\[
\begin{align*}
    f^{(1)}(x) &+ 1 \rightarrow f^{(2)}(x) &+ 1 \rightarrow f^{(3)}(x) \\
    -1 &
    -1 &
    -1 \\
    -1 &
    -1 &
    -1
\end{align*}
\]

- Each \( f^{(\ell)} \) is trained (using AdaBoost), adjust threshold (before passing through sign) to minimize false negative rate.
- Can make \( f^{(\ell)} \) in later stages more complex than in earlier stages, since most examples don’t make it to the end.
Viola & Jones cascade architecture

**Problem**: severe class imbalance (most patches don’t contain a face).

**Solution**: Train several classifiers (each using AdaBoost), and arrange in a special kind of *decision list* called a *cascade*:

\[
x \xrightarrow{f^{(1)}(x)} +1 \xrightarrow{f^{(2)}(x)} +1 \xrightarrow{f^{(3)}(x)} \cdots \xrightarrow{+1}
\]

- Each \( f^{(\ell)} \) is trained (using AdaBoost), adjust threshold (before passing through \texttt{sign}) to *minimize false negative rate*.
- Can make \( f^{(\ell)} \) in later stages more complex than in earlier stages, since most examples don’t make it to the end.

\[ \Rightarrow \text{(Cascade) classifier evaluation extremely fast.} \]
Viola & Jones detector: example results

Figure 10. Output of our face detector on a number of test images from the MIT+CMU test set.

6. Conclusions

We have presented an approach for face detection which minimizes computation time while achieving high detection accuracy. The approach was used to construct a face detection system which is approximately 15 times faster than any previous approach. Preliminary experiments, which will be described elsewhere, show that highly efficient detectors for other objects, such as pedestrians or automobiles, can also be constructed in this way.

This paper brings together new algorithms, representations, and insights which are quite generic and may well have broader application in computer vision and image processing.

The first contribution is a new technique for computing a rich set of image features using the integral image. In order to achieve true scale invariance, almost all face detection systems must operate on multiple image scales. The integral image, by eliminating the need to compute a multi-scale image pyramid, reduces the initial image processing required for face detection.
3. Bagging
**Bagging**

**Bagging** = **Bootstrap aggregating** (Leo Breiman, 1994).

**Input:** training data $S := \{(x_i, y_i)\}_{i=1}^{n}$ from $\mathcal{X} \times \{-1, +1\}$.

For $t = 1, 2, \ldots, T$:

1. Randomly pick $n$ examples with replacement from $S$
   $\rightarrow S^{(t)} := \{(x_i^{(t)}, y_i^{(t)})\}_{i=1}^{n}$ (a bootstrap sample).

2. Run learning algorithm on $S^{(t)}$
   $\rightarrow$ classifier $f_t$.

**Return** a majority vote classifier over $f_1, \ldots, f_T$. 
Aside: sampling with replacement

**Question:** if $n$ individuals are picked from a population of size $n$ *u.a.r. with replacement*, what is the probability that a given individual is *not* picked?

**Answer:** $(1 - \frac{1}{n})^n$

For large $n$: $\lim_{n \to \infty} (1 - \frac{1}{n})^n = 1/e \approx 0.3679$.

**Implications for bagging:**
- Each bootstrap sample contains about 63% of the data set.
- Remaining 37% can be used to estimate error rate of classifier trained on the bootstrap sample.
- Can average across bootstrap samples to get estimate of bagged classifier's error rate (sort of).
Aside: sampling with replacement

**Question**: if \( n \) individuals are picked from a population of size \( n \) *u.a.r. with replacement*, what is the probability that a given individual is *not* picked?

**Answer**: 

\[
\left(1 - \frac{1}{n}\right)^n
\]
Aside: sampling with replacement

**Question:** if $n$ individuals are picked from a population of size $n$ *u.a.r. with replacement*, what is the probability that a given individual is *not* picked?

**Answer:**

$$\left(1 - \frac{1}{n}\right)^n$$

For large $n$:

$$\lim_{n \to \infty} \left(1 - \frac{1}{n}\right)^n = \frac{1}{e} \approx 0.3679.$$
Aside: sampling with replacement

**Question:** if $n$ individuals are picked from a population of size $n$ *u.a.r. with replacement*, what is the probability that a given individual is *not* picked?

**Answer:**

$$\left(1 - \frac{1}{n}\right)^n$$

For large $n$:

$$\lim_{n \to \infty} \left(1 - \frac{1}{n}\right)^n = \frac{1}{e} \approx 0.3679.$$  

**Implications for bagging:**

- Each bootstrap sample contains about 63% of the data set.
- Remaining 37% can be used to estimate error rate of classifier trained on the bootstrap sample.
- Can average across bootstrap samples to get estimate of bagged classifier’s error rate (sort of).
Aside: sampling with replacement

**Question:** if \( n \) individuals are picked from a population of size \( n \) *u.a.r. with replacement*, what is the probability that a given individual is *not* picked?

**Answer:**

\[
\left( 1 - \frac{1}{n} \right)^n
\]

For large \( n \):

\[
\lim_{n \to \infty} \left( 1 - \frac{1}{n} \right)^n = \frac{1}{e} \approx 0.3679.
\]

**Implications for bagging:**

- Each bootstrap sample contains about 63% of the data set.
Aside: sampling with replacement

**Question:** if \( n \) individuals are picked from a population of size \( n \) u.a.r. with replacement, what is the probability that a given individual is *not* picked?

**Answer:**

\[
\left(1 - \frac{1}{n}\right)^n
\]

For large \( n \):

\[
\lim_{n \to \infty} \left(1 - \frac{1}{n}\right)^n = \frac{1}{e} \approx 0.3679.
\]

**Implications for bagging:**

- Each bootstrap sample contains about 63% of the data set.
- Remaining 37% can be used to estimate error rate of classifier trained on the bootstrap sample.
Aside: sampling with replacement

**Question:** if $n$ individuals are picked from a population of size $n$ *u.a.r. with replacement*, what is the probability that a given individual is *not* picked?

**Answer:**

$$ \left( 1 - \frac{1}{n} \right)^n $$

For large $n$:

$$ \lim_{n \to \infty} \left( 1 - \frac{1}{n} \right)^n = \frac{1}{e} \approx 0.3679. $$

**Implications for bagging:**

- Each bootstrap sample contains about 63% of the data set.
- Remaining 37% can be used to estimate error rate of classifier trained on the bootstrap sample.
- Can average across bootstrap samples to get estimate of *bagged classifier’s* error rate (sort of).
**Random Forests** (Leo Breiman, 2001).

**Input**: training data $S := \{(x_i, y_i)\}_{i=1}^n$ from $\mathbb{R}^d \times \{-1, +1\}$.

For $t = 1, 2, \ldots, T$:

1. Randomly pick $n$ examples with replacement from $S$  
   $$S^{(t)} := \{(x_i^{(t)}, y_i^{(t)})\}_{i=1}^n$$ (a bootstrap sample).

2. Run variant of decision tree learning algorithm on $S^{(t)}$, where each split is chosen by only considering a random subset of $\sqrt{d}$ features (rather than all $d$ features)  
   $$\rightarrow \text{decision tree classifier } f_t.$$

**Return** a majority vote classifier over $f_1, \ldots, f_T$. 
Why do Random Forests work well?

▶ Usually, decision trees are grown to perfectly fit the bootstrap sample. (Overfitting?)
▶ The hope is that sampling with replacement from training data is similar to sampling new training data from $P$. Distribution $P$ multiple IID samples from $P$: $S_1, S_2, \ldots$ Empirical distribution $P$ multiple IID samples from $P^n$: $S^{(1)}, S^{(2)}, \ldots$
▶ Combining trees in majority-vote "reduces variance" (helps to prevent overfitting).
Why do Random Forests work well?

- Usually, decision trees are grown to perfectly fit the bootstrap sample. (Overfitting???)
Why do Random Forests work well?

- Usually, decision trees are grown to perfectly fit the bootstrap sample. (Overfitting???)
- The hope is that sampling with replacement from training data is similar to sampling new training data from $P$.

<table>
<thead>
<tr>
<th>Distribution $P$</th>
<th>multiple IID samples from $P$: $S_1, S_2, \ldots$</th>
</tr>
</thead>
<tbody>
<tr>
<td>Empirical distribution $P_n$</td>
<td>multiple IID samples from $P_n$: $S^{(1)}, S^{(2)}, \ldots$</td>
</tr>
</tbody>
</table>
Why do Random Forests work well?

- Usually, decision trees are grown to perfectly fit the bootstrap sample. (Overfitting???)
- The hope is that sampling with replacement from training data is similar to sampling new training data from $P$.

<table>
<thead>
<tr>
<th>Distribution $P$</th>
<th>multiple IID samples from $P$: $S_1, S_2, \ldots$</th>
</tr>
</thead>
<tbody>
<tr>
<td>Empirical distribution $P_n$</td>
<td>multiple IID samples from $P_n$: $S^{(1)}, S^{(2)}, \ldots$</td>
</tr>
</tbody>
</table>

- Combining trees in majority-vote “reduces variance” (helps to prevent overfitting).
Key takeaways

1. Weak and strong learning.
2. AdaBoost algorithm; concept of margins in boosting.
3. Interpreting AdaBoost’s final classifier as a linear classifier, and interpreting AdaBoost as a coordinate descent algorithm.
4. Structure of decision lists / cascades.
5. Concept of bootstrap samples; bagging and random forests.