1. Motivation
Learning

What is learning?

Note:
- Some skills/knowledge that humans exhibit appear to be preprogrammed, but others appear to be acquired by other means.
- It is this latter acquisition process that we call learning.

Note:
- Human brain is a computational device, subject to limitations shared by other computational devices.
- Information that can be acquired from the world is also limited.
What is learning?

- Some skills/knowledge that humans exhibit appear to be preprogrammed, but others appear to be acquired by other means.
What is learning?

- Some skills/knowledge that humans exhibit appear to be preprogrammed, but others appear to be acquired by other means.
- It is this latter acquisition process that we call learning.
What is learning?

- Some skills/knowledge that humans exhibit appear to be preprogrammed, but others appear to be acquired by other means.
- It is this latter acquisition process that we call learning.

Why is learning possible?
Learning

What is learning?

- Some skills/knowledge that humans exhibit appear to be preprogrammed, but others appear to be acquired by other means.
- It is this latter acquisition process that we call learning.

Why is learning possible?

Note:
- Human brain is a computational device, subject to limitations shared by other computational devices.
Learning

What is learning?

- Some skills/knowledge that humans exhibit appear to be preprogrammed, but others appear to be acquired by other means.
- It is this latter acquisition process that we call learning.

Why is learning possible?

Note:

- Human brain is a computational device, subject to limitations shared by other computational devices.
- Information that can be acquired from the world is also limited.
What is learning?

- Some skills/knowledge that humans exhibit appear to be preprogrammed, but others appear to be acquired by other means.
- It is this latter acquisition process that we call learning.

Why is learning possible with limited computational and information resources?

**Note:**

- Human brain is a computational device, subject to limitations shared by other computational devices.
- Information that can be acquired from the world is also limited.
Goals of learning theory

1. Provide a rigorous theory to explain the phenomenon of learning.
2. Determine what is learnable and what is not learnable.
3. Provide guidance for design of computational devices that need to perform learning.
Goals of learning theory

1. Provide a rigorous theory to explain the phenomenon of learning.
Goals of learning theory

1. Provide a rigorous theory to explain the phenomenon of learning.
2. Determine what is learnable and what is not learnable.
Goals of learning theory

1. Provide a rigorous theory to explain the phenomenon of learning.
2. Determine what is learnable and what is not learnable.
3. Provide guidance for design of computational devices that need to perform learning.
4. ...
Goals of learning theory

1. Provide a rigorous theory to explain the phenomenon of learning.
2. Determine what is learnable and what is not learnable.
3. Provide guidance for design of computational devices that need to perform learning. → algorithms for machine learning
4. ...
2. A basic model of learning
Concept learning

Basic “thing” to be learned: concept $c$. 
Basic “thing” to be learned: \textit{concept} $c$.

- Every object is either a \textit{positive example} of $c$, or is a \textit{negative example} of $c$. 
Concept learning

Basic “thing” to be learned: *concept* $c$.

- Every object is either a *positive example* of $c$, or is a *negative example* of $c$.
- Objects are elements of the *input space* (a.k.a. *domain*) $\mathcal{X}$. 

▶ Types of animals: $\mathcal{X} =$ all present-day animals on Earth, $c =$ land mammals $=$ \{giraffe, lion, ...\}.

▶ Images of handwritten digits: $\mathcal{X} =$ \{0, 1\}_28 × 28 (i.e., \{0, 1\}_{28 \times 28} binary arrays), $c =$ set of arrays that depict numeral “4”.

▶ Can concepts be learned from its positive and negative examples?

▶ What does it mean to learn the concept?

▶ How should these examples be chosen? And how many are needed?
Basic “thing” to be learned: concept $c$.

- Every object is either a positive example of $c$, or is a negative example of $c$.
- Objects are elements of the input space (a.k.a. domain) $\mathcal{X}$, and therefore can think of a concept $c$ as
  - a subset of the input space, $c \subseteq \mathcal{X}$,
Basic “thing” to be learned: concept $c$.

- Every object is either a positive example of $c$, or is a negative example of $c$.
- Objects are elements of the input space (a.k.a. domain) $\mathcal{X}$, and therefore can think of a concept $c$ as
  - a subset of the input space, $c \subseteq \mathcal{X}$, or
  - a $\{0, 1\}$-valued function $c: \mathcal{X} \rightarrow \{0, 1\}$, where $c(x) = 1$ iff $x$ is a positive example of the concept.
Concept learning

Basic “thing” to be learned: *concept* $c$.

- Every object is either a *positive example* of $c$, or is a *negative example* of $c$.
- Objects are elements of the *input space* (a.k.a. *domain*) $\mathcal{X}$, and therefore can think of a concept $c$ as
  - a subset of the input space, $c \subseteq \mathcal{X}$, or
  - a $\{0, 1\}$-valued function $c: \mathcal{X} \to \{0, 1\}$, where $c(x) = 1$ iff $x$ is a positive example of the concept.

- **Types of animals:**
  $\mathcal{X} =$ all present-day animals on Earth,
  $c =$ land mammals $= \{\text{giraffe, lion, ...} \}$. 

\[ 6 / 25 \]
Basic “thing” to be learned: *concept* \( c \).

▶ Every object is either a *positive example* of \( c \), or is a *negative example* of \( c \).

▶ Objects are elements of the *input space* (a.k.a. *domain*) \( \mathcal{X} \), and therefore can think of a concept \( c \) as
  
  ▶ a subset of the input space, \( c \subseteq \mathcal{X} \), or
  ▶ a \( \{0, 1\} \)-valued function \( c: \mathcal{X} \rightarrow \{0, 1\} \), where \( c(x) = 1 \) iff \( x \) is a positive example of the concept.

▶ **Types of animals:**
  
  \( \mathcal{X} \) = all present-day animals on Earth,
  
  \( c = \) land mammals = \{giraffe, lion, ...\}.

▶ **Images of handwritten digits:**
  
  \( \mathcal{X} = \{0, 1\}^{28 \times 28} \) (i.e., \( 28 \times 28 \) binary arrays),
  
  \( c = \) set of arrays that depict numeral “4”.
Concept learning

Basic “thing” to be learned: concept $c$.

- Every object is either a positive example of $c$, or is a negative example of $c$.
- Objects are elements of the input space (a.k.a. domain) $\mathcal{X}$, and therefore can think of a concept $c$ as
  - a subset of the input space, $c \subseteq \mathcal{X}$, or
  - a $\{0, 1\}$-valued function $c: \mathcal{X} \to \{0, 1\}$, where $c(x) = 1$ iff $x$ is a positive example of the concept.

- Types of animals:
  $\mathcal{X} =$ all present-day animals on Earth,
  $c =$ land mammals $= \{\text{giraffe, lion, } \ldots \}$.

- Images of handwritten digits:
  $\mathcal{X} = \{0, 1\}^{28 \times 28}$ (i.e., $28 \times 28$ binary arrays),
  $c =$ set of arrays that depict numeral “4”.

- Can concepts be learned from its positive and negative examples?
Basic “thing” to be learned: concept $c$.

- Every object is either a positive example of $c$, or is a negative example of $c$.
- Objects are elements of the input space (a.k.a. domain) $\mathcal{X}$, and therefore can think of a concept $c$ as
  - a subset of the input space, $c \subseteq \mathcal{X}$, or
  - a $\{0, 1\}$-valued function $c: \mathcal{X} \to \{0, 1\}$, where $c(x) = 1$ iff $x$ is a positive example of the concept.

- **Types of animals:**
  $\mathcal{X} =$ all present-day animals on Earth,
  $c =$ land mammals $=$ \{giraffe, lion, ...\}.

- **Images of handwritten digits:**
  $\mathcal{X} = \{0, 1\}^{28 \times 28}$ (i.e., $28 \times 28$ binary arrays),
  $c =$ set of arrays that depict numeral “4”.

- Can concepts be learned from its positive and negative examples?
  - What does it mean to learn the concept?
Concept learning

Basic “thing” to be learned: concept $c$.

- Every object is either a positive example of $c$, or is a negative example of $c$.
- Objects are elements of the input space (a.k.a. domain) $\mathcal{X}$, and therefore can think of a concept $c$ as
  - a subset of the input space, $c \subseteq \mathcal{X}$, or
  - a $\{0, 1\}$-valued function $c: \mathcal{X} \rightarrow \{0, 1\}$, where $c(x) = 1$ iff $x$ is a positive example of the concept.

- **Types of animals:**
  $\mathcal{X} =$ all present-day animals on Earth,
  $c =$ land mammals $= \{\text{giraffe, lion, . . .} \}.$

- **Images of handwritten digits:**
  $\mathcal{X} = \{0, 1\}^{28 \times 28}$ (i.e., $28 \times 28$ binary arrays),
  $c =$ set of arrays that depict numeral “4”.

- Can concepts be learned from its positive and negative examples?
  - What does it mean to learn the concept?
  - How should these examples be chosen? And how many are needed?
Learning a concept

You have learned a concept $c$ if you can correctly answer questions about it:

“Is $x$ a positive example of $c$?”
You have *learned a concept* $c$ if you can correctly answer questions about it:

"Is $x$ a positive example of $c$?"

- If $x$ was seen before, then just have to have good memory.
Learning a concept

You have *learned a concept* \( c \) if you can correctly answer questions about it:

“*Is \( x \) a positive example of \( c \)?”

- If \( x \) was seen before, then just have to have good memory.
- If \( x \) was not seen before, then must *generalize* from examples you’ve seen.
Learning a concept

You have *learned a concept* $c$ if you can correctly answer questions about it:

“Is $x$ a positive example of $c$?”

- If $x$ was seen before, then just have to have good memory.
- If $x$ was not seen before, then must *generalize* from examples you’ve seen.

We are primarily interested in such cases.
Learning a concept

You have *learned a concept* $c$ if you can correctly answer questions about it:

“Is $x$ a positive example of $c$?”

- If $x$ was seen before, then just have to have good memory.
- If $x$ was not seen before, then must *generalize* from examples you’ve seen.

  *We are primarily interested in such cases.*

- We’ll pick quiz object $x$ randomly: $x \sim P$.

  ($P$ is probability distribution on $\mathcal{X}$ that is unknown to you.)
You have *learned a concept* $c$ if you can correctly answer questions about it:

> “Is $x$ a positive example of $c$?”

- If $x$ was seen before, then just have to have good memory.
- If $x$ was not seen before, then must *generalize* from examples you’ve seen.

  We are primarily interested in such cases.

- We’ll pick quiz object $x$ randomly: $x \sim P$.

  ($P$ is probability distribution on $\mathcal{X}$ that is unknown to you.)

- Should answer correctly for most objects (e.g., $P$-mass $1 - \epsilon$).
You have *learned a concept* \( c \) if you can correctly answer questions about it:

“*Is \( x \) a positive example of \( c \)?”

- If \( x \) was seen before, then just have to have good memory.
- If \( x \) was not seen before, then must *generalize* from examples you’ve seen.

  *We are primarily interested in such cases.*

- We’ll pick quiz object \( x \) randomly: \( x \sim P \).
  
  \((P \text{ is probability distribution on } \mathcal{X} \text{ that is unknown to you.})\)

- Should answer correctly for most objects (e.g., \( P \text{-mass } 1 - \epsilon \)).

Thus, reasonable to learn about \( c \) by seeing objects drawn randomly from \( P \), e.g., \( x^{(1)}, \ldots, x^{(n)} \sim_{\text{iid}} P \), each paired with their “label” \( c(x^{(i)}) \).

(You are promised that it is the same \( P \) used to pick the quiz object.)
What is successful learning?

Learning protocol
What is successful learning?

Learning protocol

- Get objects $x^{(1)}, \ldots, x^{(n)} \sim_{iid} P$, each paired with label $c(x^{(i)})$. 

Efficient PAC learning: sample size $n$ and running time $t$ are polynomial in $1/\epsilon$, $1/\delta$, and the representation sizes of the objects, concept, and hypothesis.
What is successful 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 $h: \mathcal{X} \rightarrow \{0, 1\}$.

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

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

- $\epsilon \in (0, 1)$: error rate parameter.
- $\delta \in (0, 1)$: confidence parameter.

Efficient PAC learning: sample size $n$ and running time $t$ are polynomial in $1/\epsilon$, $1/\delta$, and the representation sizes of the objects, concept, and hypothesis.
What is successful learning?

Learning protocol

- Get objects $x^{(1)}, \ldots, x^{(n)} \sim_{iid} P$, 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 $x^{(i)}$).
What is successful 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 \(h(x) = c(x)\)?
What is successful learning?

Learning protocol

- Get objects $x^{(1)}, \ldots, x^{(n)} \sim_{iid} P$, 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):

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

- $\epsilon \in (0, 1)$: error rate parameter.
- $\delta \in (0, 1)$: confidence parameter.
What is successful 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 $h: X \rightarrow \{0, 1\}$.
- 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):

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

- $\epsilon \in (0, 1)$: error rate parameter.
- $\delta \in (0, 1)$: confidence parameter.

*Efficient PAC learning*: sample size $n$ and running time $t$ are polynomial in $1/\epsilon$, $1/\delta$, and the representation sizes of the objects, concept, and hypothesis.
For which \textit{concept classes} \( C \) is it possible to efficiently PAC-learn every \( c \in C \) for arbitrary distribution \( P \)?
3. Which concept classes are efficiently PAC learnable?
Concept class #1: monotone conjunctions

- $X = \{0, 1\}^d$, the Boolean hypercube.
  Each object is described by $d$ Boolean attributes $x = (x_1, \ldots, x_d)$.

- $C =$ monotone conjunctions.
  Conjunctions of literals that appear in positive form.
Concept class #1: monotone conjunctions

- $\mathcal{X} = \{0, 1\}^d$, the *Boolean hypercube*.
  Each object is described by $d$ Boolean attributes $\mathbf{x} = (x_1, \ldots, x_d)$.
  - E.g., $x_1 = \text{“has stripes”}$, $x_2 = \text{“eats plants”}$, $x_3 = \text{“lives in water”}$.

- $\mathcal{C} = \text{monotone conjunctions}$.  
  Conjunctions of literals that appear in *positive* form.
  - E.g., $c(\mathbf{x}) = x_1 \land x_2 \land x_3$. 

What is a learning procedure for monotone conjunctions? How many examples does it need, and what is its time complexity?
Concept class #1: monotone conjunctions

▶ $\mathcal{X} = \{0, 1\}^d$, the *Boolean hypercube*.
Each object is described by $d$ Boolean attributes $\mathbf{x} = (x_1, \ldots, x_d)$.

▶ E.g., $x_1 = \text{"has stripes"}$, $x_2 = \text{"eats plants"}$, $x_3 = \text{"lives in water"}$.

▶ $\mathcal{C} =$ *monotone conjunctions*.
Conjunctions of literals that appear in *positive* form.

▶ E.g., $c(\mathbf{x}) = x_1 \land x_2 \land x_{13}$.

What is a learning procedure for monotone conjunctions? How many examples does it need, and what is its time complexity?
Learn monotone conjunction

input Examples \((x^{(1)}, c(x^{(1)})), \ldots, (x^{(n)}, c(x^{(n)})) \in \{0, 1\}^d \times \{0, 1\}\).
output Hypothesis \(h\) (a monotone conjunction).

1: Start with \(h = x_1 \land \cdots \land x_d\).
2: for \(i = 1, \ldots, n\) do
3: If \(c(x^{(i)}) = 1\), then remove all variables \(x_j\) from \(h\) s.t. \(x_j^{(i)} = 0\).
4: end for
5: return \(h\).
Algorithm for monotone conjunction

Learn monotone conjunction

input Examples \((x^{(1)}, c(x^{(1)})), \ldots, (x^{(n)}, c(x^{(n)})) \in \{0, 1\}^d \times \{0, 1\}\).
output Hypothesis \(h\) (a monotone conjunction).

1: Start with \(h = x_1 \land \cdots \land x_d\).
2: for \(i = 1, \ldots, n\) do
3: If \(c(x^{(i)}) = 1\), then remove all variables \(x_j\) from \(h\) s.t. \(x_j^{(i)} = 0\).
4: end for
5: return \(h\).

<table>
<thead>
<tr>
<th>(i)</th>
<th>(x^{(i)})</th>
<th>(c(x^{(i)}))</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>(1, 1, 1, 0, 0, 0)</td>
<td>1</td>
</tr>
<tr>
<td>2</td>
<td>(0, 1, 0, 0, 1, 0)</td>
<td>0</td>
</tr>
<tr>
<td>3</td>
<td>(1, 0, 1, 0, 1, 0)</td>
<td>1</td>
</tr>
</tbody>
</table>

\(h = x_1 \land x_2 \land x_3 \land x_4 \land x_5 \land x_6\)
Learn monotone conjunction

**input** Examples \((\mathbf{x}^{(1)}, c(\mathbf{x}^{(1)})), \ldots, (\mathbf{x}^{(n)}, c(\mathbf{x}^{(n)})) \in \{0, 1\}^d \times \{0, 1\}\).

**output** Hypothesis \(h\) (a monotone conjunction).

1: Start with \(h = x_1 \land \cdots \land x_d\).
2: for \(i = 1, \ldots, n\) do
3: If \(c(\mathbf{x}^{(i)}) = 1\), then remove all variables \(x_j\) from \(h\) s.t. \(x_j^{(i)} = 0\).
4: end for
5: return \(h\).

---

<table>
<thead>
<tr>
<th>(i)</th>
<th>(\mathbf{x}^{(i)})</th>
<th>(c(\mathbf{x}^{(i)}))</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>((1, 1, 1, 0, 0, 0))</td>
<td>1</td>
</tr>
<tr>
<td>2</td>
<td>((0, 1, 0, 0, 1, 0))</td>
<td>0</td>
</tr>
<tr>
<td>3</td>
<td>((1, 0, 1, 0, 1, 0))</td>
<td>1</td>
</tr>
</tbody>
</table>

\(h = x_1 \land x_2 \land x_3 \land \overline{x_4} \land \overline{x_5} \land \overline{x_6}\)
Algorithm for monotone conjunctions

**Learn monotone conjunction**

**input** Examples \((\mathbf{x}(1), c(\mathbf{x}(1))), \ldots, (\mathbf{x}(n), c(\mathbf{x}(n))) \in \{0, 1\}^d \times \{0, 1\}\).  

**output** Hypothesis \(h\) (a monotone conjunction).

1: Start with \(h = x_1 \land \cdots \land x_d\).
2: **for** \(i = 1, \ldots, n\) **do**
   3: If \(c(\mathbf{x}(i)) = 1\), then remove all variables \(x_j\) from \(h\) s.t. \(x_j(i) = 0\).
4: **end for**
5: **return** \(h\).

<table>
<thead>
<tr>
<th>(i)</th>
<th>(\mathbf{x}(i))</th>
<th>(c(\mathbf{x}(i)))</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>((1, 1, 1, 0, 0, 0))</td>
<td>1</td>
</tr>
<tr>
<td>2</td>
<td>((0, 1, 0, 0, 1, 0))</td>
<td>0</td>
</tr>
<tr>
<td>3</td>
<td>((1, 0, 1, 0, 1, 0))</td>
<td>1</td>
</tr>
</tbody>
</table>

Final hypothesis (after \(n = 3\) examples): \(h = x_1 \land x_3 \land \bar{x}_4 \land \bar{x}_5 \land \bar{x}_6\)

We'll say variables \(x_1\) and \(x_3\) are in \(h\).
Algorithm for monotone conjunctions

Learn monotone conjunction

**input** Examples \((x^{(1)}, c(x^{(1)})), \ldots, (x^{(n)}, c(x^{(n)})) \in \{0, 1\}^d \times \{0, 1\} \).  
**output** Hypothesis \(h\) (a monotone conjunction).

1. Start with \(h = x_1 \land \cdots \land x_d\).
2. **for** \(i = 1, \ldots, n\) **do**
3. \hspace{1em} If \(c(x^{(i)}) = 1\), then remove all variables \(x_j\) from \(h\) s.t. \(x^{(i)}_j = 0\).
4. **end for**
5. **return** \(h\).

<table>
<thead>
<tr>
<th>(i)</th>
<th>(x^{(i)})</th>
<th>(c(x^{(i)}))</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>(1, 1, 1, 0, 0, 0)</td>
<td>1</td>
</tr>
<tr>
<td>2</td>
<td>(0, 1, 0, 0, 1, 0)</td>
<td>0</td>
</tr>
<tr>
<td>3</td>
<td>(1, 0, 1, 0, 1, 0)</td>
<td>1</td>
</tr>
</tbody>
</table>

\(h = x_1 \land \neg x_2 \land x_3 \land x_4 \land x_5 \land x_6\)
Algorithm for monotone conjunctions

Learn monotone conjunction

**input** Examples \((\mathbf{x}^{(1)}, c(\mathbf{x}^{(1)})), \ldots, (\mathbf{x}^{(n)}, c(\mathbf{x}^{(n)})) \in \{0, 1\}^d \times \{0, 1\}\).

**output** Hypothesis \(h\) (a monotone conjunction).

1: Start with \(h = x_1 \land \cdots \land x_d\).
2: **for** \(i = 1, \ldots, n\) **do**
3: If \(c(\mathbf{x}^{(i)}) = 1\), then remove all variables \(x_j\) from \(h\) s.t. \(x_j^{(i)} = 0\).
4: **end for**
5: **return** \(h\).

<table>
<thead>
<tr>
<th>(i)</th>
<th>(\mathbf{x}^{(i)})</th>
<th>(c(\mathbf{x}^{(i)}))</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>((1, 1, 1, 0, 0, 0))</td>
<td>1</td>
</tr>
<tr>
<td>2</td>
<td>((0, 1, 0, 0, 1, 0))</td>
<td>0</td>
</tr>
<tr>
<td>3</td>
<td>((1, 0, 1, 0, 1, 0))</td>
<td>1</td>
</tr>
</tbody>
</table>

Final hypothesis (after \(n = 3\) examples): \(h = x_1 \land x_3\).

(We’ll say variables \(x_1\) and \(x_3\) are *in* \(h\).)
Properties of the algorithm

Facts:

1. Running time is $O(nd)$.
2. Variables in $h$ is always a superset of the variables in $c$. (We may have some irrelevant variables in $h$—more on this later.)

Theorem: For any $\epsilon, \delta \in (0, 1)$, if $n \geq d^{\epsilon} \ln d \delta$, then with probability at least $1 - \delta$ (over choice of $n$ labeled examples), the hypothesis $h$ satisfies $\Pr_{x \sim P}(h(x) \neq c(x)) \leq \epsilon$. 

13 / 25
Properties of the algorithm

Facts:
1. Running time is $O(nd)$. 
2. Variables in $h$ is always a superset of the variables in $c$. (We may have some irrelevant variables in $h$—more on this later.)

Theorem: For any $\epsilon, \delta \in (0, 1)$, if $n \geq d \epsilon \ln d \delta$, then with probability at least $1 - \delta$ (over choice of $n$ labeled examples), the hypothesis $h$ satisfies $\Pr_{x \sim P}(h(x) \neq c(x)) \leq \epsilon$. 

13 / 25
Properties of the algorithm

**Facts:**

1. Running time is $O(nd)$.

2. Variables in $h$ is always a *superset* of the variables in $c$.
   
   (We may have some irrelevant variables in $h$—more on this later.)
Properties of the algorithm

**Facts:**

1. Running time is $O(nd)$.

2. Variables in $h$ is always a *superset* of the variables in $c$.
   
   (We may have some irrelevant variables in $h$—more on this later.)

**Theorem:** For any $\epsilon, \delta \in (0, 1)$, if

\[
n \geq \frac{d}{\epsilon} \ln \frac{d}{\delta},
\]

then with probability at least $1 - \delta$ (over choice of $n$ labeled examples), the hypothesis $h$ satisfies

\[
\Pr_{x \sim P} (h(x) \neq c(x)) \leq \epsilon.
\]
Analyzing the algorithm

1. Say variable $x_j$ is bad if $x_j$ is in $h$ but not in $c$. 
1. Say variable $x_j$ is bad if $x_j$ is in $h$ but not in $c$. Why can we have bad variables in $h$?
1. Say variable \( x_j \) is bad if \( x_j \) is in \( h \) but not in \( c \).
   Why can we have bad variables in \( h \)?
   Because we may fail to see positive example with \( x_j = 0 \) during learning.
Analyzing the algorithm

1. Say variable $x_j$ is bad if $x_j$ is in $h$ but not in $c$.
   Why can we have bad variables in $h$?
   Because we may fail to see positive example with $x_j = 0$ during learning.

2. Any $x$ with bad $x_j = 0$ has $h(x) = 0$, but could have $c(x) = 1$. 


Analyzing the algorithm

1. Say variable $x_j$ is bad if $x_j$ is in $h$ but not in $c$.
   Why can we have bad variables in $h$?
   Because we may fail to see positive example with $x_j = 0$ during learning.
2. Any $\mathbf{x}$ with bad $x_j = 0$ has $h(\mathbf{x}) = 0$, but could have $c(\mathbf{x}) = 1$.
3. Let $q_j := \Pr_{\mathbf{x} \sim P}(x_j = 0 \land c(\mathbf{x}) = 1)$. 
Analyzing the algorithm

1. Say variable $x_j$ is bad if $x_j$ is in $h$ but not in $c$.
   Why can we have bad variables in $h$?
   Because we may fail to see positive example with $x_j = 0$ during learning.
2. Any $x$ with bad $x_j = 0$ has $h(x) = 0$, but could have $c(x) = 1$.
3. Let $q_j := \Pr_{x \sim P}(x_j = 0 \land c(x) = 1)$.
4. By union bound (i.e., $\Pr(E \cup F) \leq \Pr(E) + \Pr(F)$),
   $$
   \Pr_{x \sim P}(h(x) \neq c(x)) = \Pr_{x \sim P}\left(\bigcup_{j : x_j \text{ is bad}} \{x_j = 0 \land c(x) = 1\}\right) \leq \sum_{j : x_j \text{ is bad}} q_j.
   $$
5. Say $x_j$ is light if $q_j \leq \epsilon/d$.
   If all bad $x_j$ are light, then
   $$
   \sum_{j : x_j \text{ is bad}} q_j \leq \epsilon.
   $$
6. If $x_j$ is heavy (i.e., $q_j > \epsilon/d$), what is the probability that $x_j$ is in $h$?
   $$(1 - q_j)^n \leq \exp(-nq_j) \leq \exp(-n\epsilon/d).$$
7. Probability that there exists a heavy $x_j$ in $h$:
   $$
   \Pr(x_1, \ldots, x_n \sim \text{iid } P) \left(\text{there is a heavy } x_j \text{ in } h\right) \leq \frac{1}{d} \exp\left(-\frac{n\epsilon}{d}\right) \leq \delta
   $$
   by union bound and choice of $n = \left(\frac{d}{\epsilon} \log \frac{d}{\delta}\right)$. 

Analyzing the algorithm

1. Say variable $x_j$ is bad if $x_j$ is in $h$ but not in $c$.
   Why can we have bad variables in $h$?
   Because we may fail to see positive example with $x_j = 0$ during learning.

2. Any $x$ with bad $x_j = 0$ has $h(x) = 0$, but could have $c(x) = 1$.

3. Let $q_j := \Pr_{x \sim P}(x_j = 0 \land c(x) = 1)$.

4. By union bound (i.e., $\Pr(E \cup F) \leq \Pr(E) + \Pr(F)$),
   \[
   \Pr_{x \sim P}(h(x) \neq c(x)) = \Pr_{x \sim P}\left(\bigcup_{j: x_j \text{ is bad}} \{x_j = 0 \land c(x) = 1\}\right) \leq \sum_{j: x_j \text{ is bad}} q_j.
   \]

5. Say $x_j$ is light if $q_j \leq \epsilon/d$. 
Analyzing the algorithm

1. Say variable $x_j$ is **bad** if $x_j$ is in $h$ but not in $c$.
   Why can we have bad variables in $h$?
   Because we may fail to see positive example with $x_j = 0$ during learning.

2. Any $x$ with bad $x_j = 0$ has $h(x) = 0$, but could have $c(x) = 1$.

3. Let $q_j := \Pr_{x \sim P} (x_j = 0 \land c(x) = 1)$.

4. By *union bound* (i.e., $\Pr(E \cup F) \leq \Pr(E) + \Pr(F)$),
   \[
   \Pr_{x \sim P}(h(x) \neq c(x)) = \Pr_{x \sim P}\left( \bigcup_{j : x_j \text{ is bad}} \{x_j = 0 \land c(x) = 1\} \right) \leq \sum_{j : x_j \text{ is bad}} q_j.
   \]

5. Say $x_j$ is **light** if $q_j \leq \epsilon/d$.
   If all bad $x_j$ are light, then $\sum_{j : x_j \text{ is bad}} q_j \leq \epsilon$. 
1. Say variable $x_j$ is bad if $x_j$ is in $h$ but not in $c$.
   Why can we have bad variables in $h$?
   Because we may fail to see positive example with $x_j = 0$ during learning.
2. Any $x$ with bad $x_j = 0$ has $h(x) = 0$, but could have $c(x) = 1$.
3. Let $q_j := \Pr_{x \sim P}(x_j = 0 \land c(x) = 1)$.
4. By union bound (i.e., $\Pr(E \cup F) \leq \Pr(E) + \Pr(F)$),

   \[
   \Pr_{x \sim P}(h(x) \neq c(x)) = \Pr_{x \sim P}\left(\bigcup_{j: x_j \text{ is bad}} \{x_j = 0 \land c(x) = 1\}\right) \leq \sum_{j: x_j \text{ is bad}} q_j.
   \]
5. Say $x_j$ is light if $q_j \leq \epsilon/d$.
   If all bad $x_j$ are light, then $\sum_{j: x_j \text{ is bad}} q_j \leq \epsilon$.
6. If $x_j$ is heavy (i.e., $q_j > \epsilon/d$), what is the probability that $x_j$ is in $h$?
Analyzing the algorithm

1. Say variable $x_j$ is bad if $x_j$ is in $h$ but not in $c$.
   Why can we have bad variables in $h$?
   Because we may fail to see positive example with $x_j = 0$ during learning.

2. Any $x$ with bad $x_j = 0$ has $h(x) = 0$, but could have $c(x) = 1$.

3. Let $q_j := \Pr_{x \sim P}(x_j = 0 \land c(x) = 1)$.

4. By \textit{union bound} (i.e., $\Pr(E \cup F) \leq \Pr(E) + \Pr(F)$),
   \[
   \Pr_{x \sim P}(h(x) \neq c(x)) = \Pr_{x \sim P}\left(\bigcup_{j: x_j \text{ is bad}} \{x_j = 0 \land c(x) = 1\}\right) \leq \sum_{j: x_j \text{ is bad}} q_j.
   \]

5. Say $x_j$ is light if $q_j \leq \epsilon/d$.
   If all bad $x_j$ are light, then $\sum_{j: x_j \text{ is bad}} q_j \leq \epsilon$.

6. If $x_j$ is heavy (i.e., $q_j > \epsilon/d$), what is the probability that $x_j$ is in $h$?
   \[
   (1 - q_j)^n \leq \exp(-nq_j) \leq \exp(-n\epsilon/d).
   \]
Analyzing the algorithm

1. Say variable $x_j$ is bad if $x_j$ is in $h$ but not in $c$.
   Why can we have bad variables in $h$?
   Because we may fail to see positive example with $x_j = 0$ during learning.

2. Any $x$ with bad $x_j = 0$ has $h(x) = 0$, but could have $c(x) = 1$.

3. Let $q_j := \Pr_{x \sim P}(x_j = 0 \land c(x) = 1)$.

4. By union bound (i.e., $\Pr(E \cup F) \leq \Pr(E) + \Pr(F)$),
   \[
  \Pr_{x \sim P}(h(x) \neq c(x)) = \Pr_{x \sim P}\left(\bigcup_{j: x_j \text{ is bad}} \{x_j = 0 \land c(x) = 1\}\right) \leq \sum_{j: x_j \text{ is bad}} q_j.
   \]

5. Say $x_j$ is light if $q_j \leq \epsilon/d$.
   If all bad $x_j$ are light, then $\sum_{j: x_j \text{ is bad}} q_j \leq \epsilon$.

6. If $x_j$ is heavy (i.e., $q_j > \epsilon/d$), what is the probability that $x_j$ is in $h$?
   \[
   (1 - q_j)^n \leq \exp(-nq_j) \leq \exp(-n\epsilon/d).
   \]

7. Probability that there exists a heavy $x_j$ in $h$:
   \[
   \Pr_{x^{(1)}, \ldots, x^{(n)} \sim \text{iid } P}\left(\text{there is a heavy } x_j \text{ in } h\right) \leq d \exp(-n\epsilon/d) \leq \delta
   \]
   by union bound and choice of $n = (d/\epsilon) \ln(d/\delta)$. 

Concept class #2: conjunctions

- $\mathcal{X} = \{0, 1\}^d$, the *Boolean hypercube*.
  Each object is described by $d$ Boolean attributes $\mathbf{x} = (x_1, \ldots, x_d)$.

- $\mathcal{C} =$ *conjunctions*.
  Conjunctions of literals, may appear in negated form.
Concept class #2: conjunctions

- $\mathcal{X} = \{0, 1\}^d$, the Boolean hypercube. Each object is described by $d$ Boolean attributes $\boldsymbol{x} = (x_1, \ldots, x_d)$.
  - E.g., $x_1 = “\text{has stripes}”, x_2 = “\text{eats plants}”, x_3 = “\text{lives in water}”$.

- $\mathcal{C} =$ conjunctions. Conjunctions of literals, may appear in negated form.
  - E.g., $c(\boldsymbol{x}) = x_1 \land x_2 \land \neg x_{13}$. 

What is a learning procedure for conjunctions? How many examples does it need, and what is its time complexity?
Concept class #2: conjunctions

\[ \mathcal{X} = \{0, 1\}^d, \text{ the Boolean hypercube.} \]
Each object is described by \( d \) Boolean attributes \( \mathbf{x} = (x_1, \ldots, x_d) \).

\[ \Rightarrow \text{E.g., } x_1 = \text{“has stripes”}, \ x_2 = \text{“eats plants”}, \ x_3 = \text{“lives in water”}. \]

\[ \mathcal{C} = \text{conjunctions}. \]
Conjunctions of literals, may appear in negated form.

\[ \Rightarrow \text{E.g., } c(\mathbf{x}) = x_1 \land x_2 \land \neg x_3. \]

What is a learning procedure for conjunctions? How many examples does it need, and what is its time complexity?
Reduction

Use feature map: \( \phi(x) := (x_1, \ldots, x_d, \neg x_1, \ldots, \neg x_d) \).

A conjunction \( c \) over \( x_1, \ldots, x_d \) is a monotone conjunction \( c' \) over \( x_1, \ldots, x_d, x'_1, \ldots, x'_d \), where \( x'_i = \neg x_i \).

Use algorithm for monotone conjunctions with feature expansion \( \phi \).

Everything is as before, except now need sample size \( n = 2^d \epsilon \ln 2^d \delta \).
Use feature map: $\phi(x) := (x_1, \ldots, x_d, \neg x_1, \ldots, \neg x_d)$. 
Reduction

- Use feature map: \( \phi(x) := (x_1, \ldots, x_d, \neg x_1, \ldots, \neg x_d) \).

- A conjunction \( c \) over \( x_1, \ldots, x_d \) is a monotone conjunction \( c' \) over \( x_1, \ldots, x_d, x'_1, \ldots, x'_d \), where \( x'_i = \neg x_i \).
Reduction

- Use feature map: \( \phi(x) := (x_1, \ldots, x_d, \neg x_1, \ldots, \neg x_d) \).

- A conjunction \( c \) over \( x_1, \ldots, x_d \) is a monotone conjunction \( c' \) over \( x_1, \ldots, x_d, x'_1, \ldots, x'_d \), where \( x'_i = \neg x_i \).

- Use algorithm for monotone conjunctions with feature expansion \( \phi \).
Use feature map: \( \phi(x) := (x_1, \ldots, x_d, \neg x_1, \ldots, \neg x_d) \).

A conjunction \( c \) over \( x_1, \ldots, x_d \) is a monotone conjunction \( c' \) over \( x_1, \ldots, x_d, x'_1, \ldots, x'_d \), where \( x'_i = \neg x_i \).

Use algorithm for monotone conjunctions with feature expansion \( \phi \).

Everything is as before, except now need sample size

\[
    n = \frac{2d}{\epsilon} \ln \frac{2d}{\delta}.
\]
Concept class #3: disjunctions

- $\mathcal{X} = \{0, 1\}^d$, the Boolean hypercube.
  Each object is described by $d$ Boolean attributes $\mathbf{x} = (x_1, \ldots, x_d)$.

- $C = \text{disjunctions}$. Disjunction of literals, may appear in negated form.
Concept class #3: disjunctions

- $\mathcal{X} = \{0, 1\}^d$, the *Boolean hypercube.*
  Each object is described by $d$ Boolean attributes $\mathbf{x} = (x_1, \ldots, x_d)$.
  - E.g., $x_1 = \text{“has stripes”}$, $x_2 = \text{“eats plants”}$, $x_3 = \text{“lives in water”}$.

- $\mathcal{C} = \textit{disjunctions}.$
  Disjunction of literals, may appear in negated form.
  - E.g., $c(\mathbf{x}) = x_1 \lor x_2 \lor \neg x_{13}$. 

What is a learning procedure for disjunctions? How many examples does it need, and what is its time complexity? 

**Hint:** use a reduction.
Concept class #3: disjunctions

- $X = \{0, 1\}^d$, the Boolean hypercube.
  Each object is described by $d$ Boolean attributes $x = (x_1, \ldots, x_d)$.
    - E.g., $x_1 = \text{“has stripes”}$, $x_2 = \text{“eats plants”}$, $x_3 = \text{“lives in water”}$.

- $C = \text{disjunctions}$.
  Disjunction of literals, may appear in negated form.
    - E.g., $c(x) = x_1 \lor x_2 \lor \neg x_3$.

What is a learning procedure for disjunctions? How many examples does it need, and what is its time complexity?
Concept class #3: disjunctions

- $\mathcal{X} = \{0, 1\}^d$, the Boolean hypercube. Each object is described by $d$ Boolean attributes $\mathbf{x} = (x_1, \ldots, x_d)$.
  - E.g., $x_1 =$ “has stripes”, $x_2 =$ “eats plants”, $x_3 =$ “lives in water”.

- $\mathcal{C} =$ disjunctions. Disjunction of literals, may appear in negated form.
  - E.g., $c(\mathbf{x}) = x_1 \lor x_2 \lor \neg x_{13}$.

What is a learning procedure for disjunctions? How many examples does it need, and what is its time complexity? **Hint:** use a reduction.
Concept class #4: half-spaces

- $\mathcal{X} = \mathbb{R}^d$.
  - Each object is described by $d$ real-valued attributes $\mathbf{x} = (x_1, \ldots, x_d)$.
- $\mathcal{C} = \text{half-spaces}$.
Concept class #4: half-spaces

- $\mathcal{X} = \mathbb{R}^d$.
  Each object is described by $d$ real-valued attributes $\mathbf{x} = (x_1, \ldots, x_d)$.
- $\mathcal{C} = \textit{half-spaces}$.
  - E.g., $c = \{ \mathbf{x} \in \mathbb{R}^d : \mathbf{w}^T \mathbf{x} \leq b \}$. 

Efficiently PAC learnable using linear programming with sample size $n = O(d \epsilon \log \frac{1}{\epsilon} + \frac{1}{\epsilon} \log \frac{1}{\delta})$. 

\[18 / 25\]
Concept class #4: half-spaces

- $\mathcal{X} = \mathbb{R}^d$.
  Each object is described by $d$ real-valued attributes $\mathbf{x} = (x_1, \ldots, x_d)$.
- $\mathcal{C} = \text{half-spaces}$.
  - E.g., $c = \{ \mathbf{x} \in \mathbb{R}^d : \mathbf{w}^T \mathbf{x} \leq b \}$.

Efficiently PAC learnable using linear programming with sample size

$$n = O \left( \frac{d}{\epsilon} \log \frac{1}{\epsilon} + \frac{1}{\epsilon} \log \frac{1}{\delta} \right).$$
Concept class #5: intersections of half-spaces

- \( \mathcal{X} = \mathbb{R}^d \).
  
  Each object is described by \( d \) real-valued attributes \( \mathbf{x} = (x_1, \ldots, x_d) \).

- \( \mathcal{C} = \) intersections of half-spaces.
Concept class #5: intersections of half-spaces

- \( X = \mathbb{R}^d \).
  Each object is described by \( d \) real-valued attributes \( \mathbf{x} = (x_1, \ldots, x_d) \).

- \( C = \text{intersections of half-spaces} \).
  - E.g., \( c = \{ \mathbf{x} \in \mathbb{R}^d : w_1^\top \mathbf{x} \leq b_1 \land w_2^\top \mathbf{x} \leq b_2 \} \).
Concept class #5: intersections of half-spaces

- $\mathcal{X} = \mathbb{R}^d$. Each object is described by $d$ real-valued attributes $\mathbf{x} = (x_1, \ldots, x_d)$.

- $\mathcal{C} =$ intersections of half-spaces.
  - E.g., $c = \{ \mathbf{x} \in \mathbb{R}^d : \mathbf{w}_1^T \mathbf{x} \leq b_1 \land \mathbf{w}_2^T \mathbf{x} \leq b_2 \}$.

Unknown if efficiently PAC learnable or not!
Concept class #5: intersections of half-spaces

- $\mathcal{X} = \mathbb{R}^d$.
  Each object is described by $d$ real-valued attributes $\mathbf{x} = (x_1, \ldots, x_d)$.
- $\mathcal{C} =$ intersections of half-spaces.
  - E.g., $c = \{ \mathbf{x} \in \mathbb{R}^d : \mathbf{w}_1^\top \mathbf{x} \leq b_1 \land \mathbf{w}_2^\top \mathbf{x} \leq b_2 \}$.

Unknown if efficiently PAC learnable or not!

(But is efficiently learnable under additional assumptions on $c$ and $P$.)
Vapnik & Chervonenkis (1971):

For every concept classes $\mathcal{C}$, one of the following is true:

1. There is a number $\mathcal{VC}(\mathcal{C})$ (VC dimension) such that, using sample size $n = O(\mathcal{VC}(\mathcal{C}) \epsilon \log \frac{1}{\epsilon} + 1 \epsilon \log \frac{1}{\delta})$, there is an algorithm that learns every concept in $\mathcal{C}$ for arbitrary distribution $P$. The algorithm is not necessarily efficient.

2. There is no algorithm that learns every concept in $\mathcal{C}$ for arbitrary distribution $P$.

"Intersections of half-spaces" ($\mathcal{C}$) is in the first case. "Learning" is well-understood from sample complexity perspective, but not from computational complexity perspective.
Vapnik & Chervonenkis (1971):

For every concept classes $C$, one of the following is true:

1. There is a number $VC(C)$ (VC dimension) such that, using sample size

   $$n = O \left( \frac{VC(C)}{\epsilon} \log \frac{1}{\epsilon} + \frac{1}{\epsilon} \log \frac{1}{\delta} \right),$$

   there is an algorithm that learns every concept in $C$ for arbitrary distribution $P$. 
Vapnik & Chervonenkis (1971):

For every concept classes \( C \), one of the following is true:

1. There is a number \( VC(C) \) (\textit{VC dimension}) such that, using sample size

   \[
   n = O \left( \frac{VC(C)}{\epsilon} \log \frac{1}{\epsilon} + \frac{1}{\epsilon} \log \frac{1}{\delta} \right),
   \]

   there is an algorithm that learns every concept in \( C \) for arbitrary distribution \( P \).

2. There is no algorithm that learns every concept in \( C \) for arbitrary distribution \( P \).
Vapnik & Chervonenkis (1971):

For every concept classes $C$, one of the following is true:

1. There is a number $\text{VC}(C)$ (VC dimension) such that, using sample size

$$n = O \left( \frac{\text{VC}(C)}{\epsilon} \log \frac{1}{\epsilon} + \frac{1}{\epsilon} \log \frac{1}{\delta} \right),$$

there is an algorithm that learns every concept in $C$ for arbitrary distribution $P$.

The algorithm is not necessarily efficient.

2. There is no algorithm that learns every concept in $C$ for arbitrary distribution $P$. 

Vapnik & Chervonenkis (1971):

For every concept classes $\mathcal{C}$, one of the following is true:

1. There is a number $\text{VC} (\mathcal{C})$ (VC dimension) such that, using sample size

$$ n = O\left(\frac{\text{VC} (\mathcal{C})}{\epsilon} \log \frac{1}{\epsilon} + \frac{1}{\epsilon} \log \frac{1}{\delta}\right), $$

there is an algorithm that learns every concept in $\mathcal{C}$ for arbitrary distribution $P$.

The algorithm is not necessarily efficient.

2. There is no algorithm that learns every concept in $\mathcal{C}$ for arbitrary distribution $P$.

“Intersections of half-spaces” (& many other natural $\mathcal{C}$) is in the first case.
Vapnik & Chervonenkis (1971):

For every concept classes $C$, one of the following is true:

1. There is a number $\text{VC}(C)$ (VC dimension) such that, using sample size $n = O \left( \frac{\text{VC}(C)}{\epsilon} \log \frac{1}{\epsilon} + \frac{1}{\epsilon} \log \frac{1}{\delta} \right)$,

   there is an algorithm that learns every concept in $C$ for arbitrary distribution $P$.

   The algorithm is not necessarily efficient.

2. There is no algorithm that learns every concept in $C$ for arbitrary distribution $P$.

“Intersections of half-spaces” (& many other natural $C$) is in the first case.

“Learning” is well-understood from sample complexity perspective, but not from computational complexity perspective.
The generic strategy

Given $(\mathbf{x}^{(1)}, c^{(1)}), \ldots, (\mathbf{x}^{(n)}, c^{(n)}) \sim_{iid} \mathcal{P}$, what is generic way to learn $c$?
The generic strategy

Given \((x^{(1)}, c(x^{(1)})], \ldots, (x^{(n)}, c(x^{(n)})] \sim_{iid} P\), what is generic way to learn \(c\)?

Find \(h \in C\) that is consistent with these \(n\) examples:

\[
h(x^{(i)}) = c(x^{(i)}), \quad i = 1, \ldots, n.
\]
The generic strategy

Given \((x^{(1)}, c(x^{(1)})), \ldots, (x^{(n)}, c(x^{(n)})) \sim_{iid} P\), what is generic way to learn \(c\)?

Find \(h \in C\) that is \textit{consistent} with these \(n\) examples:

\[ h(x^{(i)}) = c(x^{(i)}), \quad i = 1, \ldots, n. \]

**Theorem:** If \(n \geq O\left(\frac{1}{\epsilon} (VC(C) \log \frac{1}{\epsilon} + \log \frac{1}{\delta})\right)\), then with probability at least \(1 - \delta\) (over choice of \(n\) examples),

\[ \Pr_{x \sim P}(h(x) \neq c(x)) \leq \epsilon \quad \text{for all } h \in C \text{ consistent with all } n \text{ examples}. \]
The generic strategy

Given \((x^{(1)}, c(x^{(1)})), \ldots, (x^{(n)}, c(x^{(n)})) \sim_{iid} P\), what is generic way to learn \(c\)?

Find \(h \in C\) that is **consistent** with these \(n\) examples:

\[ h(x^{(i)}) = c(x^{(i)}), \quad i = 1, \ldots, n. \]

**Theorem:** If \(n \geq O \left( \frac{1}{\epsilon} \left( \text{VC}(C) \log \frac{1}{\epsilon} + \log \frac{1}{\delta} \right) \right) \), then with probability at least \(1 - \delta\) (over choice of \(n\) examples),

\[ \Pr_{x \sim P}(h(x) \neq c(x)) \leq \epsilon \quad \text{for all } h \in C \text{ consistent with all } n \text{ examples.} \]

**A more general theorem** (in some respects): If \((x_1, y_1), \ldots, (x_n, y_n) \sim_{iid} Q\) (\(Q\) is arbitrary distribution on \(X \times \{0, 1\}\)) and \(n \geq O \left( \frac{1}{\epsilon^2} \left( \text{VC}(C) \log \frac{1}{\epsilon} + \log \frac{1}{\delta} \right) \right) \), then with probability at least \(1 - \delta\) (over choice of \(n\) labeled examples),

\[ \forall h \in C. \quad \left| \Pr_{(x,y) \sim Q}(h(x) \neq y) - \frac{1}{n} \sum_{i=1}^{n} 1\{h(x_i) \neq y_i\} \right| \leq \epsilon. \]
The generic strategy

Given \((x^{(1)}, c(x^{(1)})), \ldots, (x^{(n)}, c(x^{(n)})) \sim_{iid} P\), what is generic way to learn \(c\)?

Find \(h \in C\) that is consistent with these \(n\) examples:

\[
h(x^{(i)}) = c(x^{(i)}), \quad i = 1, \ldots, n.
\]

**Theorem:** If \(n \geq O \left( \frac{1}{\epsilon} \left( VC(C) \log \frac{1}{\epsilon} + \log \frac{1}{\delta} \right) \right)\), then with probability at least \(1 - \delta\) (over choice of \(n\) examples),

\[
\Pr_{x \sim P} (h(x) \neq c(x)) \leq \epsilon \quad \text{for all } h \in C \text{ consistent with all } n \text{ examples}.
\]

**A more general theorem** (in some respects): If \((x_1, y_1), \ldots, (x_n, y_n) \sim_{iid} Q\) (\(Q\) is arbitrary distribution on \(X \times \{0, 1\}\)) and \(n \geq O \left( \frac{1}{\epsilon^2} \left( VC(C) \log \frac{1}{\epsilon} + \log \frac{1}{\delta} \right) \right)\), then with probability at least \(1 - \delta\) (over choice of \(n\) labeled examples),

\[
\forall h \in C. \left| \Pr_{(x,y) \sim Q} (h(x) \neq y) - \frac{1}{n} \sum_{i=1}^{n} 1\{h(x_i) \neq y_i\} \right| \leq \epsilon.
\]

I.e., true error rate is close to empirical error rate, for every \(h \in C\).
The generic strategy

Given \((x^{(1)}, c(x^{(1)})), \ldots, (x^{(n)}, c(x^{(n)})) \sim_{iid} P\), what is generic way to learn \(c\)?

Find \(h \in C\) that is consistent with these \(n\) examples:

\[
h(x^{(i)}) = c(x^{(i)}), \quad i = 1, \ldots, n.\]

**Theorem:** If \(n \geq O\left(\frac{1}{\epsilon} \left(\text{VC}(C) \log \frac{1}{\epsilon} + \log \frac{1}{\delta}\right)\right)\), then with probability at least \(1 - \delta\) (over choice of \(n\) examples),

\[
\Pr_{x \sim P} (h(x) \neq c(x)) \leq \epsilon \quad \text{for all } h \in C \text{ consistent with all } n \text{ examples}.
\]

A more general theorem (in some respects): If \((x_1, y_1), \ldots, (x_n, y_n) \sim_{iid} Q\) (\(Q\) is arbitrary distribution on \(X \times \{0, 1\}\)) and \(n \geq O\left(\frac{1}{\epsilon^2} \left(\text{VC}(C) \log \frac{1}{\epsilon} + \log \frac{1}{\delta}\right)\right)\), then with probability at least \(1 - \delta\) (over choice of \(n\) labeled examples),

\[
\forall h \in C. \left| \Pr_{(x,y) \sim Q} (h(x) \neq y) - \frac{1}{n} \sum_{i=1}^{n} 1\{h(x_i) \neq y_i\} \right| \leq \epsilon.
\]

I.e., true error rate is close to empirical error rate, for every \(h \in C\).

Sample size is large enough to allay worries of overfitting (for hypotheses in \(C\)).
There are concept classes $C$ such that both of the following are true:

1. The sample size needed to learn every concept in $C$ is polynomial. (E.g., using the generic strategy.)
2. Under certain cryptographic assumptions (like “factoring is hard”), there is no efficient algorithm that learns every concept in $C$. 

Goldreich, Goldwasser, and Micali (1986); Kearns & Valiant (1994):
Goldreich, Goldwasser, and Micali (1986); Kearns & Valiant (1994):

There are concept classes $C$ such that both of the following are true:
Goldreich, Goldwasser, and Micali (1986); Kearns & Valiant (1994):

There are concept classes $C$ such that both of the following are true:

1. The sample size needed to learn every concept in $C$ is polynomial.
   (E.g., using the generic strategy.)
Goldreich, Goldwasser, and Micali (1986); Kearns & Valiant (1994):

There are concept classes $C$ such that both of the following are true:

1. The sample size needed to learn every concept in $C$ is polynomial.
   (E.g., using the generic strategy.)

2. Under certain cryptographic assumptions (like “factoring is hard”), there is no efficient algorithm that learns every concept in $C$. 
Statistical learning theory

Generalization theory

- How large is the difference between the true error rate and the training error rate?
- How quickly (as a function of $n$) does the error rate of your hypothesis approach the minimum error rate?

(Empirical process theory, high-dimensional probability, . . .)

Lower bounds

- What is the best that any method can do?

(Information theory, minimax analysis, . . .)

Regularization

- How does regularization affect convergence?

(Inverse problems, non-parametric statistics, . . .)
Generalization theory: How large is difference between true error rate and training error rate? How quickly (as a function of $n$) does the error rate of your hypothesis approach the minimum error rate?

(Empirical process theory, high-dimensional probability, . . . )
Generalization theory: How large is difference between true error rate and training error rate? How quickly (as a function of $n$) does the error rate of your hypothesis approach the minimum error rate?

(Empirical process theory, high-dimensional probability, . . . )

Lower bounds: what is the best that any method can do?

(Information theory, minimax analysis, . . . )
Statistical learning theory

- **Generalization theory**: How large is difference between true error rate and training error rate? How quickly (as a function of $n$) does the error rate of your hypothesis approach the minimum error rate?
  
  (Empirical process theory, high-dimensional probability, ...)

- **Lower bounds**: what is the best that any method can do?
  
  (Information theory, minimax analysis, ...)

- **Regularization**: how does regularization affect convergence?
  
  (Inverse problems, non-parametric statistics, ...)

- ...
Many theoretical models for studying learnability beyond PAC learning:

- Noise-tolerance: learn even when labels differ from \( c(x) \) sometimes.
- Real-valued concepts: i.e., regression.
- Convex risk minimization: can be more tractable than zero-one loss.
- Online learning: you are quizzed about your knowledge of concept in sequence of examples over time.
- Learning with queries: you can ask for label of arbitrary \( x \in X \).
- Teaching: what is the best way to teach (i.e., provide sequence of examples) a stubborn learner?
Beyond PAC learning

Many theoretical models for studying learnability beyond PAC learning:

- **Noise-tolerance**: learn even when labels differ from $c(x)$ sometimes.
Many theoretical models for studying learnability beyond PAC learning:

- **Noise-tolerance**: learn even when labels differ from $c(x)$ sometimes.
- **Real-valued concepts**: i.e., regression.
Beyond PAC learning

Many theoretical models for studying learnability beyond PAC learning:

- **Noise-tolerance**: learn even when labels differ from \( c(x) \) sometimes.
- **Real-valued concepts**: i.e., regression.
- **Convex risk minimization**: can be more tractable than zero-one loss.
Many theoretical models for studying learnability beyond PAC learning:

- **Noise-tolerance**: learn even when labels differ from $c(x)$ sometimes.
- **Real-valued concepts**: i.e., regression.
- **Convex risk minimization**: can be more tractable than zero-one loss.
- **Online learning**: you are quizzed about your knowledge of concept in sequence of examples over time.
Beyond PAC learning

Many theoretical models for studying learnability beyond PAC learning:

- **Noise-tolerance**: learn even when labels differ from $c(x)$ sometimes.
- **Real-valued concepts**: i.e., regression.
- **Convex risk minimization**: can be more tractable than zero-one loss.
- **Online learning**: you are quizzed about your knowledge of concept in sequence of examples over time.
- **Learning with queries**: you can ask for label of arbitrary $x \in \mathcal{X}$. 
Beyond PAC learning

Many theoretical models for studying learnability beyond PAC learning:

- **Noise-tolerance**: learn even when labels differ from \( c(x) \) sometimes.
- **Real-valued concepts**: i.e., regression.
- **Convex risk minimization**: can be more tractable than zero-one loss.
- **Online learning**: you are quizzed about your knowledge of concept in sequence of examples over time.
- **Learning with queries**: you can ask for label of arbitrary \( x \in \mathcal{X} \).
- **Teaching**: what is the best way to teach (i.e., provide sequence of examples) a stubborn learner?
- ...
Key takeaways

1. Formal model for concept learning.
2. Learnability of certain simple concept classes.
3. The generic learning strategy (+ why it works at a high-level).
4. Statistical concerns versus computational concerns.