Machine Learning:
Issues, Answers, and Quandaries

Thomas G. Dietterich
Department of Computer Science
Oregon State University
Corvallis, OR 97331
tgd@cs.orst.edu

Thanks to the following people:

Hussein Almuallim, Jonathan Amsterdam, William Cohen, David Haussler, Lenny Pitt, Carol Rivin, and Dietrich Wettschereck

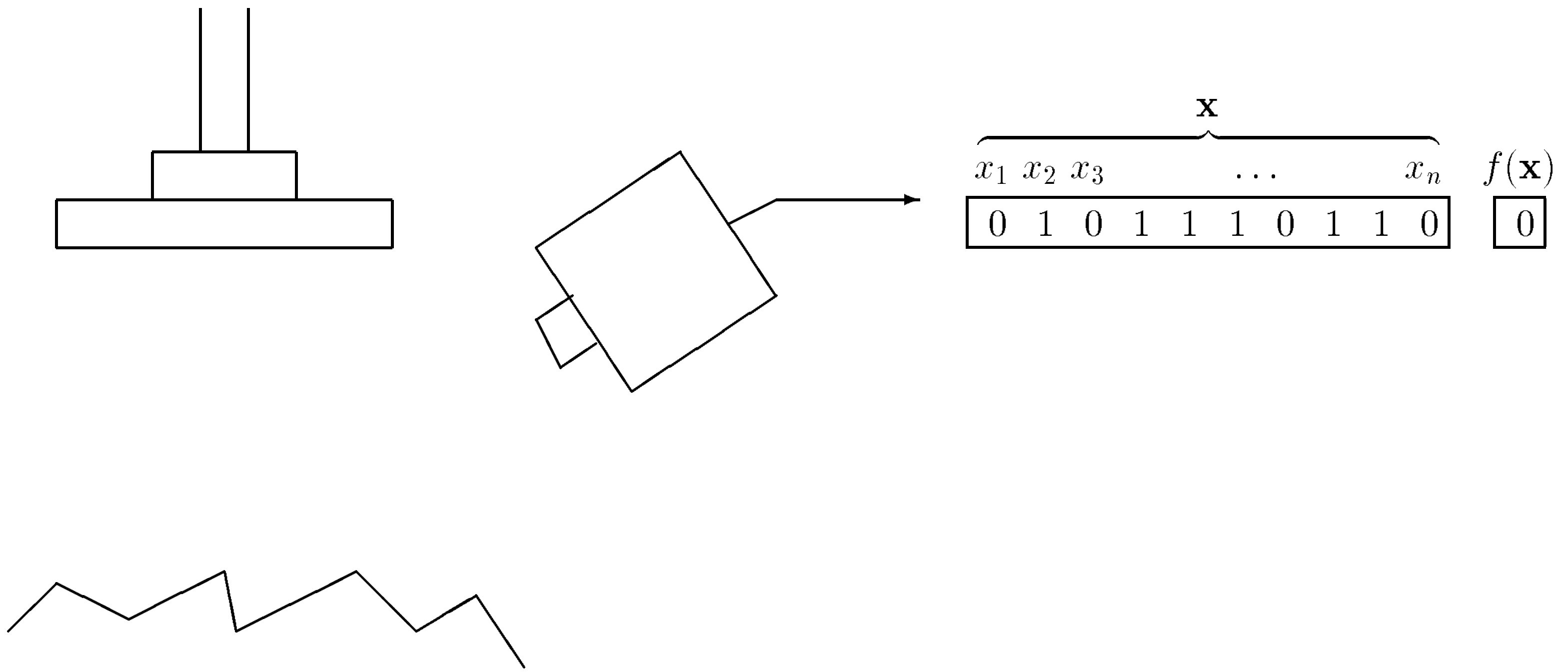
A Learning Robot on Mars?

Given: Examples of locations on the ground and whether each was safe.

Task: Learn where it is safe-to-step.

Can we provide an efficient algorithm that is guaranteed to succeed?

Formalizing the Task



Formalizing the Task

Given: A set S of training examples.

Each example has the form $\langle \mathrm{x}, f(\mathrm{x}) \rangle$.

$f : \{0,1\}^n \mapsto \{0,1\}$

Find: f.

Perfect Learning is Impossible

Truth Table for f

x_1	x_2	$\cdot$	$\cdot$	$\cdot$	$\cdot$	$\cdot$	x_n	$f(\mathrm{x})$
0	0	0	0	0	0	0	0	1
0	0	0	0	0	0	0	1	1
				.				?
				.				?
				.				?
1	0	0	1	0	0	1	0	0
1	0	0	1	0	0	1	1	1
				.				?
				.				?
				.				?
1	1	1	1	1	1	1	1	0

We have no basis for guessing the "?" rows

When $n = 10$ and $m = 512$ the probability of learning a randomly chosen function is less than 7.46×10^{-155}.

Approximately Correct Learning

Allow learning algorithm A to output $A(S) = f'$, an *approximation* to f.

Allow $\epsilon \cdot 2^n$ rows to be incorrect.

x_1	x_2	$\cdot$	$\cdot$	$\cdot$	$\cdot$	$\cdot$	x_n	$f'(\mathrm{x})$
0	0	0	0	0	0	0	0	1
0	0	0	0	0	0	0	1	1
			$\cdot$					0
			$\cdot$					1
			$\cdot$					0
1	0	0	1	0	0	1	0	0
1	0	0	1	0	0	1	1	1
			$\cdot$					1
			$\cdot$					1
			$\cdot$					0
1	1	1	1	1	1	1	1	0

Any
$\epsilon 2^n$
Wrong

A Theorem

Theorem. Given m examples, no algorithm can learn more than

$$2^m \sum_{i=0}^{\epsilon 2^n} \binom{2^n - m}{i}$$

functions approximately correctly.

For a small problem ($n = 10$)
and a large sample ($m = 512$)
and error rate ($\epsilon = .1$)
the probability of learning a
randomly chosen function is $< 4.65 \times 10^{-45}$.

Frequently Approximately Correct Learning

Relax this even more:

Allow the algorithm to be totally wrong with frequency δ.

- With frequency δ, $A(S) = f'$ can be very wrong.
- With frequency $1 - \delta$, $A(S) = f'$ can have $\epsilon 2^n$ incorrect rows.

Limits on Learning From Examples

Theorem. (Dietterich, 1989) Given a sample of size m, no learning algorithm can learn more than

$$\frac{1}{1-\delta} \cdot 2^m \sum_{i=0}^{\epsilon 2^n} \binom{2^n - m}{i}$$

functions frequently approximately correctly.

For a small problem ($n = 10$)
and a large sample ($m = 512$)
and error rate ($\epsilon = .1$)
and confidence level ($\delta = .1$)
the probability of learning a
randomly chosen function is $< 5.17 \times 10^{-45}$.

Conclusion

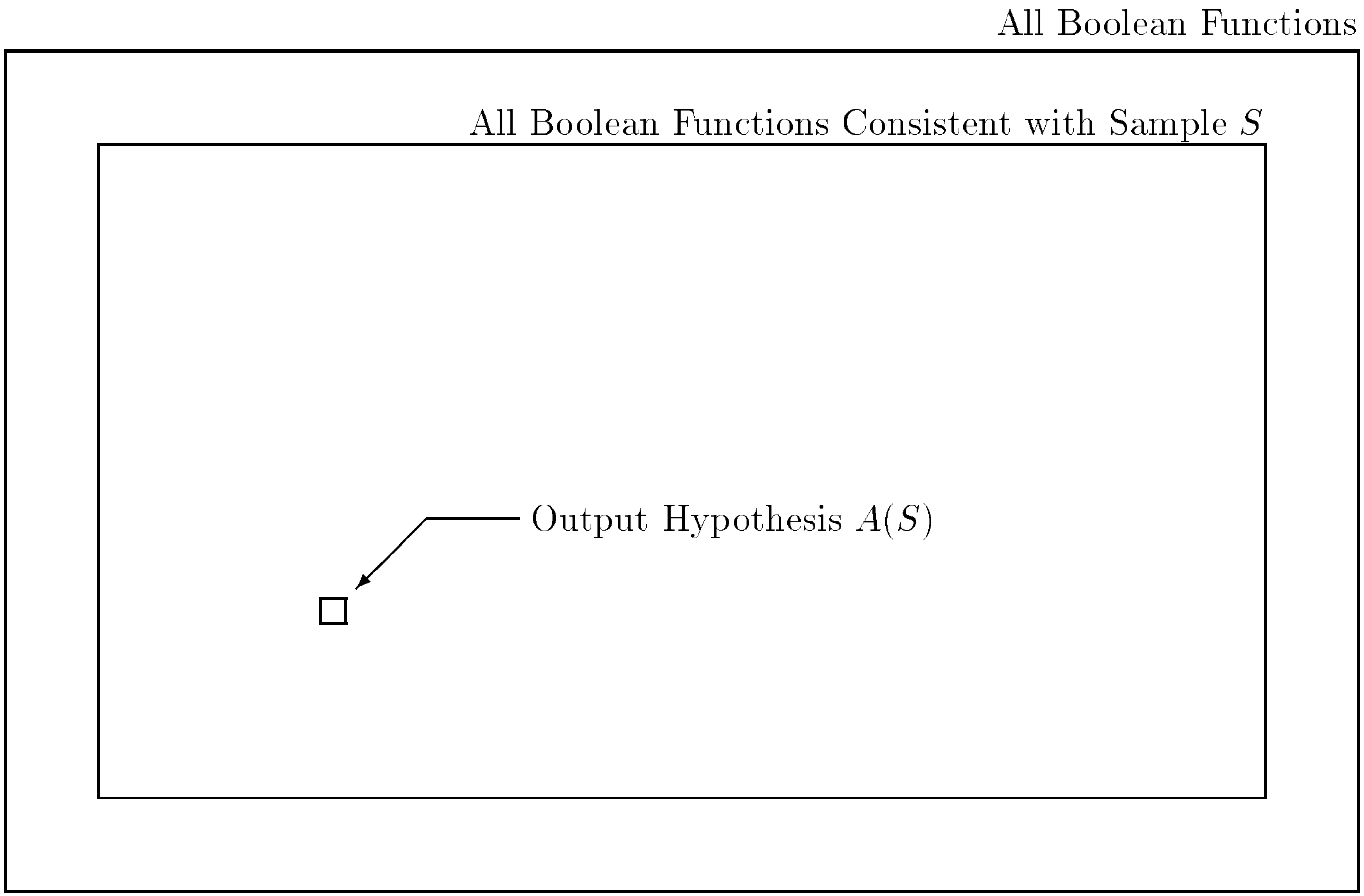
There are no general purpose learning algorithms.

Quandary 1

Human learning appears to be very general.

How can we reconcile the generality of human learning with these results?

Bias



Bias = Criteria—in addition to S—used to select an hypothesis.

Bias = Assumptions about the unknown function f.

Outline

1. **How can we define "bias"?**
2. **How can we implement biases?**
3. **How can we tell if the output of an algorithm is correct?**
4. **How can we tell if our bias is correct?**
5. **How can we recover from incorrect biases?**
   [How can we deal with noise?]
6. **How can we compare biases?**

How can we define "bias"?

- **Absolute Bias**

  Rule out parts of the hypothesis space.

  Draw f' from the remainder of the hypothesis space.

- **Preference Bias**

  If both f'_1 and f'_2 are consistent with S, then always prefer f'_1.

- **Random Bias**

  Draw f' at random.

How can we specify biases?

- **Absolute Bias**
  - Restricted representation for hypotheses.
  - Restricted derivation of hypotheses.
  - Assign prior probability $= 0$.
- **Preference Bias**
  - Partially order hypotheses by syntactic simplicity.
  - Partially order hypotheses by derivation length.
  - Assign prior probabilities > 0.

Examples of Restricted Representation (Absolute Bias)

- Simple conjunction

$$c_1 \wedge c_2 \wedge \ldots \wedge c_n \equiv f' = 1$$

- k-DNF (set of rules with at most k conditions)

$$\begin{array}{rcl} c_{1,1} \wedge c_{1,2} \wedge \ldots \wedge c_{1,k} & \supset & f' = 1 \vee \\ c_{2,1} \wedge c_{2,2} \wedge \ldots \wedge c_{2,k} & \supset & f' = 1 \vee \\ & \ldots & \\ c_{l,1} \wedge c_{l,2} \wedge \ldots \wedge c_{l,k} & \supset & f' = 1 \end{array}$$

- linear threshold units (perceptrons)

$$f'(\mathrm{x}) = 1 \equiv \sum_{i=1}^{n} w_i x_i > \theta$$

- k-3NN (3-layer neural net with k hidden units)

$$f'(\mathrm{x}) = \sigma\left(\sum_{j=1}^{k} w_j \sigma_j(\mathrm{w}^{(\mathrm{j})} \cdot \mathrm{x})\right)$$

Restricted Derivation
(Absolute Bias)

Given: theory Θ

designated predicate symbol $C(\mathrm{x})$

Assume: $\forall_{\mathrm{x}} \; f(\mathrm{x}) \supset C(\mathrm{x})$

Propose: all implicants of C:

all $f'(\mathrm{x})$ such that

$\Theta \vdash \forall_{\mathrm{x}} \; f'(\mathrm{x}) \supset C(\mathrm{x})$

(Flann and Dietterich, 1989)

Example (Cohen, 1990)

Given: theory Θ of bridge rules

designated symbol $legal\_bridge\_bid(\mathrm{x})$

Assume: $\forall_{\mathrm{x}}\ good\_bridge\_bid(\mathrm{x}) \supset legal\_bridge\_bid(\mathrm{x})$

Propose: all $f'(\mathrm{x})$ such that

$\Theta \vdash \forall_{\mathrm{x}}\ f'(\mathrm{x}) \supset legal\_bridge\_bid(\mathrm{x})$

Proofs employ a restricted proof theory (SLD resolution)

Syntactic Simplicity
(Preference Bias)

- Smallest Disjunctive Normal Form (DNF)
- Smallest Decision Tree
- Smallest Finite State Automaton
- Smallest 3-layer Neural Network

Derivation Simplicity (Preference Bias)

Prefer f' such that

$$\Theta \vdash \ \forall_{\mathrm{x}}\, f'(\mathrm{x}) \ \supset \ C(\mathrm{x})$$

with the shortest derivation.

This corresponds to specializing C as little as possible to make it consistent with the training sample S.

Outline

1. How can we define “bias”?
2. $\boxed{\textbf{2. How can we implement biases?}}$
3. How can we tell if the output of an algorithm is correct?
4. How can we tell if our bias is correct?
5. How can we recover from incorrect biases?
   [How can we deal with noise?]
6. How can we compare biases?

Implementing These Biases

Given: Training set S
Bias

Search for: f' that minimizes error on S
and maximizes preference

- Restricted representation bias defines the search space.
- Preference bias controls the search.

Choosing a Search Algorithm

Many search methods can be employed to implement biases.

Greedy algorithms: ID3, set covering methods, Boolean conjunctions, cascade-correlation.

Gradient descent: Backpropagation, perceptron learning rule.

Conjugate gradient: `opt`, Autoclass

Genetic algorithms: Boolean functions, neural networks.

Simulated annealing: Boltzmann Machine

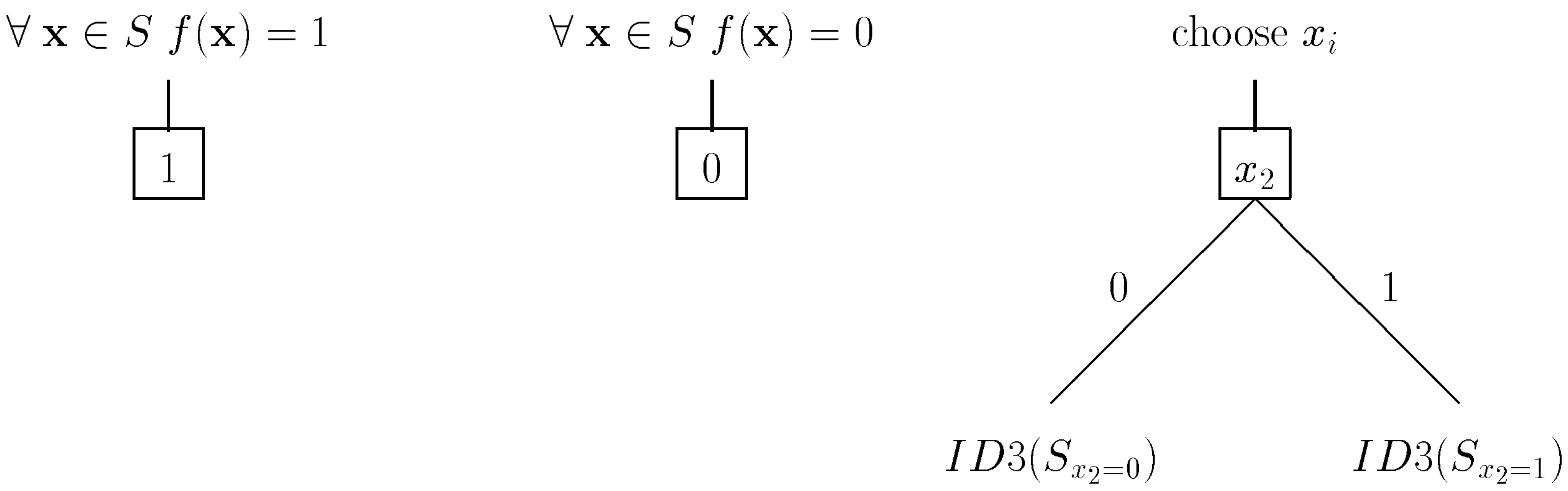
Exhaustive search: PVM

Example: ID3, CART

Bias: No absolute bias.
Preference bias: prefer small decision trees.

Algorithm: Greedy. Grow a decision tree recursively starting at the root.

$ID3(S):$



Example: Explanation-based Learning

Given: A **domain theory** Θ

A predicate symbol C from Θ

Bias: Preference bias: specific hypotheses

Absolute bias: f' must be an implicant of C

$(\Theta \vdash f' \supset C)$

Algorithm:

Generate all implicants for the positive examples

Generate all implicants for the negative examples

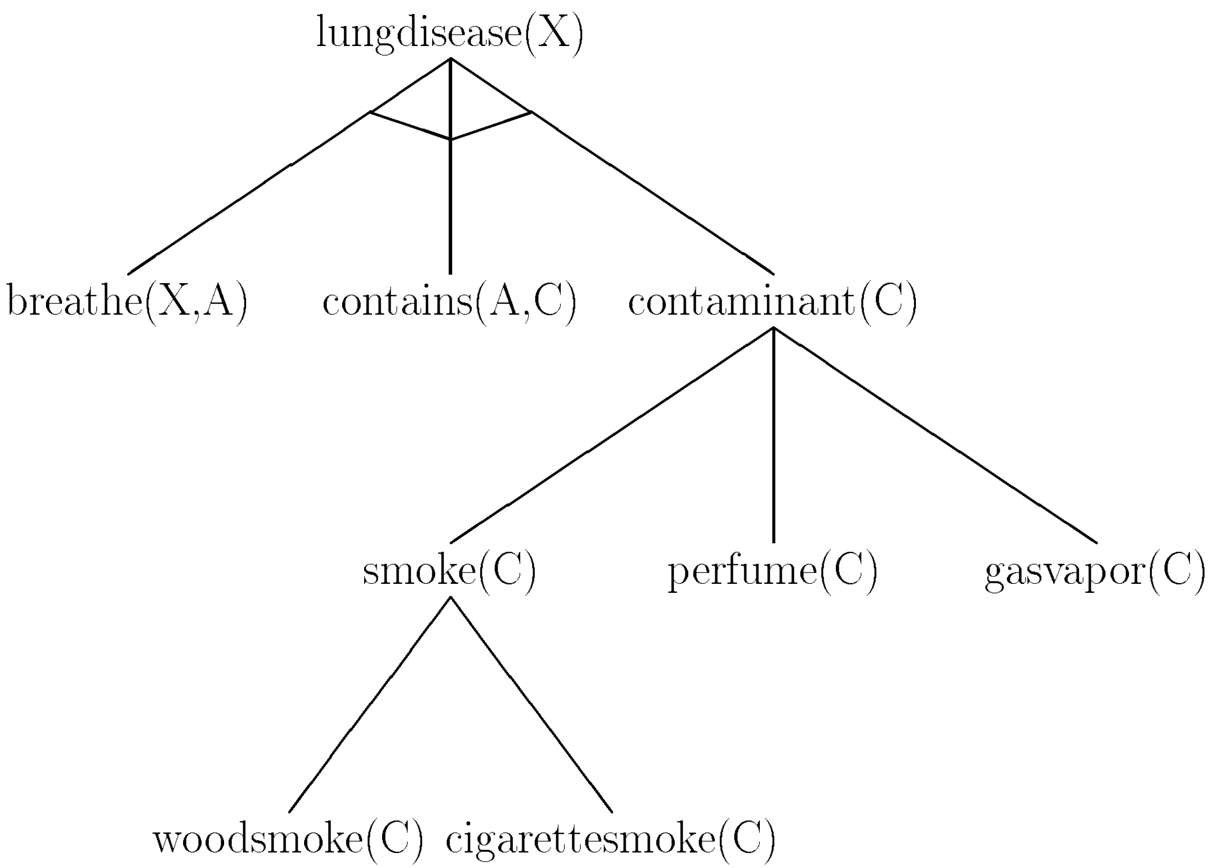
Choose an implicant that covers all positive examples and no negative examples.

The implicants are also called “explanations”.

Implicants of *lungdisease(X)*

$$
\begin{array}{llllllll}
 & & & & lungdisease(X) & \supset & lungdisease(X) \\
breathe(X,A) & \wedge & contains(A,C) & \wedge & contaminant(C) & \supset & lungdisease(X) \\
breathe(X,A) & \wedge & contains(A,C) & \wedge & smoke(C) & \supset & lungdisease(X) \\
breathe(X,A) & \wedge & contains(A,C) & \wedge & woodsmoke(C) & \supset & lungdisease(X) \\
breathe(X,A) & \wedge & contains(A,C) & \wedge & cigarettesmoke(C) & \supset & lungdisease(X) \\
breathe(X,A) & \wedge & contains(A,C) & \wedge & perfume(C) & \supset & lungdisease(X) \\
breathe(X,A) & \wedge & contains(A,C) & \wedge & gasvapor(C) & \supset & lungdisease(X)
\end{array}
$$

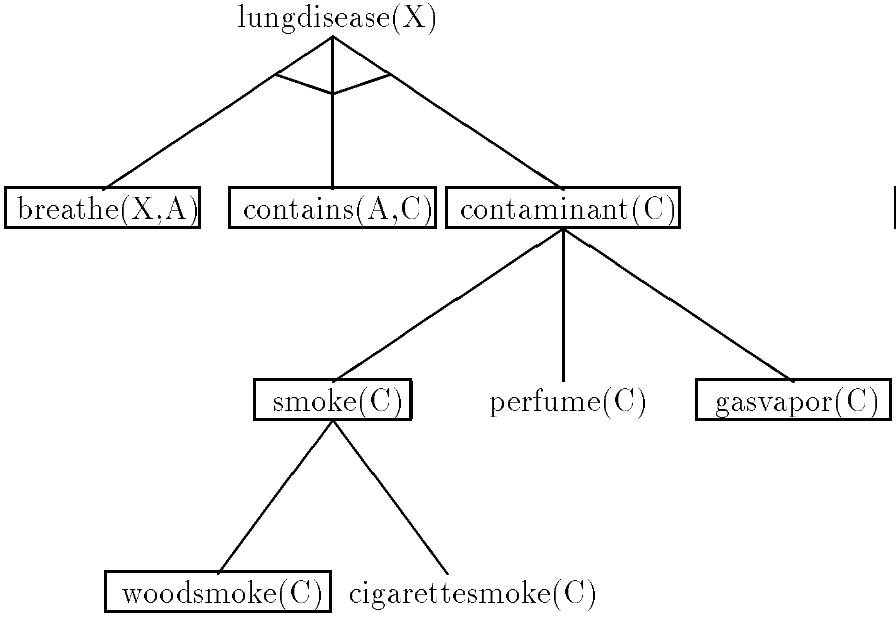
Explanation-based Learning



Assumption: $lungcancer(x) \supset lungdisease(x)$.

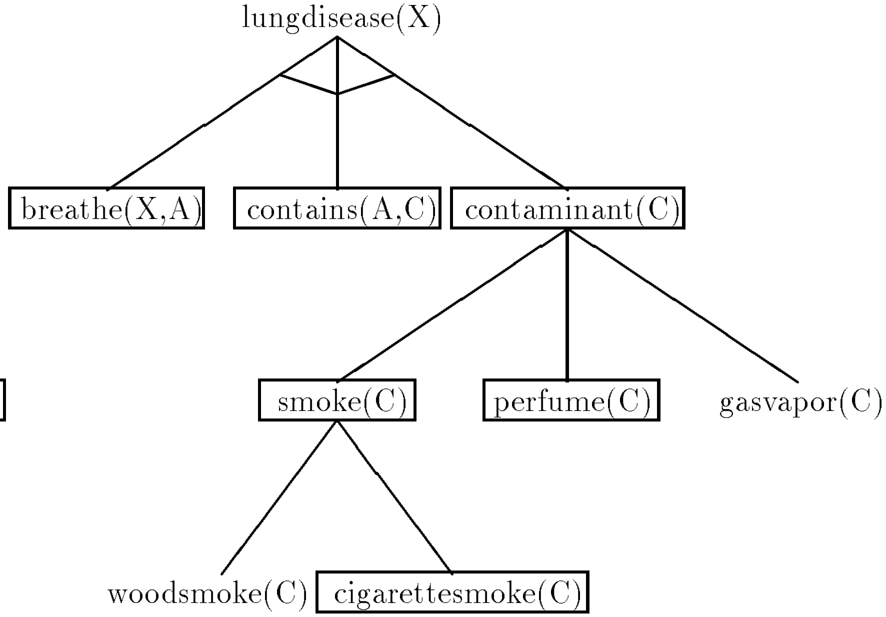
Generating Implicants (Explanations)

Example 1



$lungcancer(e1)\ \wedge\ breathe(e1,a1) \wedge$
$contains(a1,c1)\ \wedge\ woodsmoke(c1) \wedge$
$contains(a1,c2)\ \wedge\ gasvapor(c2)$

Example 2



$lungcancer(e2)\ \wedge\ breathe(e2,a2) \wedge$
$contains(a2,c3)\ \wedge\ cigarettesmoke(c3) \wedge$
$contains(a2,c4)\ \wedge\ perfume(c4)$

Result: $lungcancer(X)\ \equiv\ breath(X,A) \wedge contains(A,C) \wedge smoke(C)$

Computational Complexity of Biases

Bias	Time Complexity
Boolean conjunction	Polynomial
k-term-DNF	NP-hard
k-DNF	Polynomial
k-CNF	Polynomial
k-DL	Polynomial
LTU	Polynomial
k-3NN	NP-hard
k-tip-prefixes	Polynomial
smallest DT	NP-hard
smallest Boolean Formula	NP-hard*
smallest FSM	NP-hard*

*Finding even a *small* hypothesis is as hard as several cryptographic schemes.

Outline

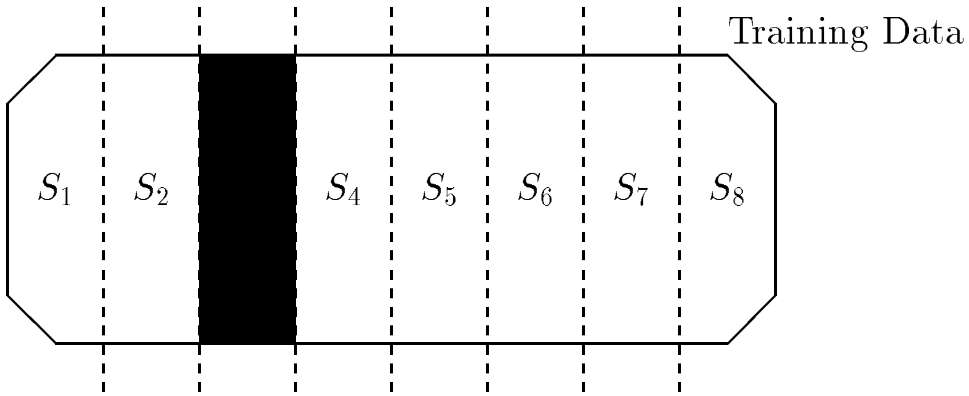
1. How can we define "bias"?
2. How can we implement biases?
3. **How can we tell if the output of an algorithm is correct?**
4. How can we tell if our bias is correct?
5. How can we recover from incorrect biases?
   [How can we deal with noise?]
6. How can we compare biases?

Can we tell if f' is correct?

- If our hypothesis space H is small enough,
- and our sample $|S|$ is large enough,
- and we can find a consistent $f' \in H$
- then f' is probably approximately correct.

$$|S| \geq \frac{1}{\epsilon}\left(\ln \frac{1}{\delta} + \ln |H|\right)$$

Cross Validation



$$\left.\begin{array}{lcl}
A(\cup_{i\neq 1} S_i) = f_1' & \text{test } f_1'(S_1) & \longrightarrow \text{error } e_1 \\
A(\cup_{i\neq 2} S_i) = f_2' & \text{test } f_2'(S_2) & \longrightarrow \text{error } e_2 \\
A(\cup_{i\neq 3} S_i) = f_3' & \text{test } f_3'(S_3) & \longrightarrow \text{error } e_3 \\
 & \cdot & \\
 & \cdot & \\
 & \cdot & \\
A(\cup_{i\neq k} S_i) = f_k' & \text{test } f_k'(S_k) & \longrightarrow \text{error } e_k
\end{array}\right\} \begin{array}{c} \text{Average} \\ \text{to} \\ \text{estimate} \\ A(S) = f' \\ \text{error} \end{array}$$

Outline

1. How can we define “bias”?
2. How can we implement biases?
3. How can we tell if the output of an algorithm is correct?
4. **How can we tell if our bias is correct?**
5. How can we recover from incorrect biases?
   [How can we deal with noise?]
6. How can we compare biases?

Correctness of Bias

- **Given:** An ensemble Π of learning problems
- A bias is **correct** if it allows all of these problems to be solved efficiently.

> **Quandary 2**
>
> Do the learning problems that naturally confront an agent in our world satisfy a sufficiently strong bias?

- If so, what is this bias?
- How can we find it?

Outline

1. How can we define “bias”?
2. How can we implement biases?
3. How can we tell if the output of an algorithm is correct?
4. How can we tell if our bias is correct?
5. **How can we recover from incorrect biases?**

   [How can we deal with noise?]
6. How can we compare biases?

Recovering from Incorrect Bias

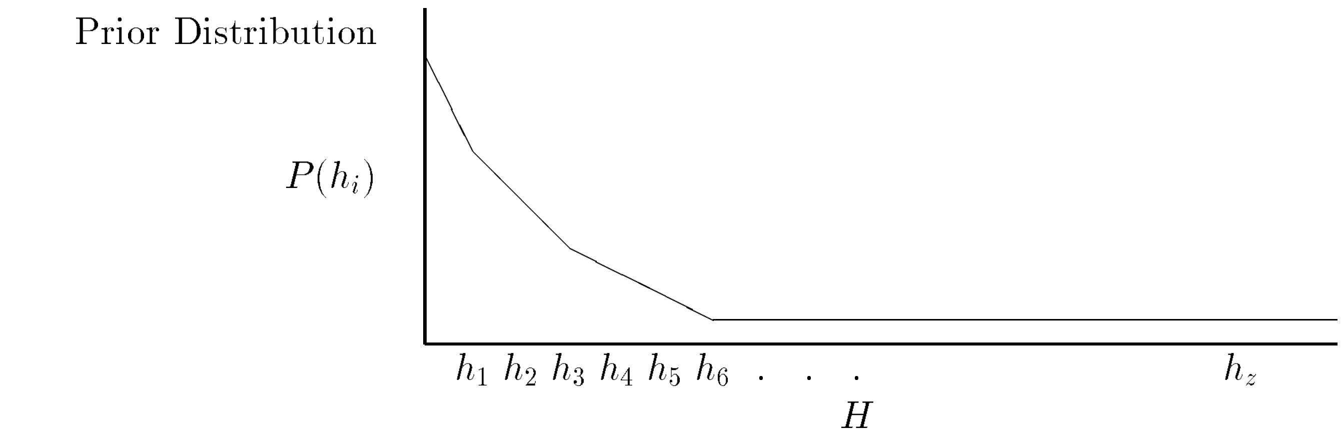
Suppose our bias prevents us from finding a good f'.

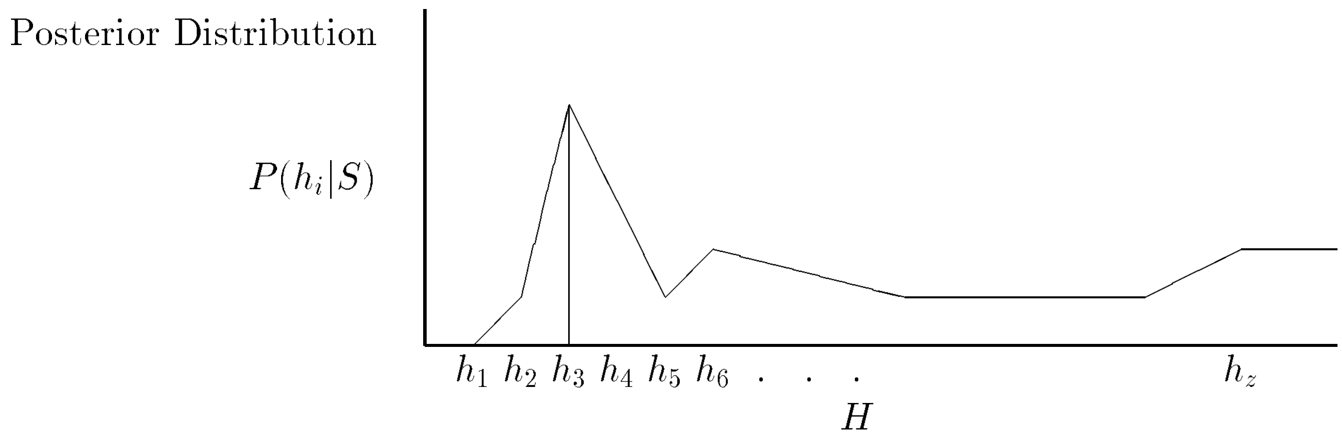
- An absolute bias excludes all good approximations.
- A relative bias orders them badly.
- The data are noisy (i.e., f has a stochastic component).

We can recover from this by

- Averaging multiple hypotheses.
- Error-correcting output codes.
- Controlling over-fitting.

Averaging Multiple Hypotheses





$$P[f(\mathrm{x}) = 1|S] = \sum_i P(h_i|S) \text{ when } h_i(\mathrm{x}) = 1$$

Averaging Multiple Hypotheses

Oppen & Haussler (1991) have shown that

$$\text{error-rate(Bayes)} \approx 0.71 \cdot \text{error-rate(Boltz)}$$

where the Boltz algorithm selects a single hypothesis according to the posterior distribution $P(h_i|S)$.

Approximation: Generate k "good" hypotheses and vote.

Error-Correcting Output Codes

Multiclass learning problems: $f : \{0,1\}^n \mapsto \{c_1, \ldots, c_k\}$.

Learn one binary function f_i for each of the k classes.

Class	Code Word									
	f_0	f_1	f_2	f_3	f_4	f_5	f_6	f_7	f_8	f_9
0	1	0	0	0	0	0	0	0	0	0
1	0	1	0	0	0	0	0	0	0	0
2	0	0	1	0	0	0	0	0	0	0
3	0	0	0	1	0	0	0	0	0	0
4	0	0	0	0	1	0	0	0	0	0
5	0	0	0	0	0	1	0	0	0	0
6	0	0	0	0	0	0	1	0	0	0
7	0	0	0	0	0	0	0	1	0	0
8	0	0	0	0	0	0	0	0	1	0
9	0	0	0	0	0	0	0	0	0	1

An error in any one f_i can cause an error in f'.

Error-Correcting Codes (Dietterich and Bakiri, 1991)

Assign an l-bit **codeword** to each class c_i.

Class	Code Word w_i														
	f_0	f_1	f_2	f_3	f_4	f_5	f_6	f_7	f_8	f_9	f_{10}	f_{11}	f_{12}	f_{13}	f_{14}
0	1	1	0	0	0	0	1	0	1	0	0	1	1	0	1
1	0	0	1	1	1	1	0	1	0	1	1	0	0	1	0
2	1	0	0	1	0	0	0	1	1	1	1	0	1	0	1
3	0	0	1	1	0	1	1	1	0	0	0	0	1	0	1
4	1	1	1	0	1	0	1	1	0	0	1	0	0	0	1
5	0	1	0	0	1	1	0	1	1	1	0	0	0	0	1
6	1	0	1	1	1	0	0	0	0	1	0	1	0	0	1
7	0	0	0	1	1	1	1	0	1	0	1	1	0	0	1
8	1	1	0	1	0	1	1	0	0	1	0	0	0	1	1
9	0	1	1	1	0	0	0	0	1	0	1	0	0	1	1

The codewords are chosen to have large Hamming distances.

Apply A to learn each of the functions f_j.

Applying Error-Correcting Codes

$$
\begin{array}{rcc}
 & & \text{Nearest} \\
 & & \text{codeword} \\
f_1(\mathrm{x}) = & \boxed{1} \longrightarrow & \boxed{1} \\
f_2(\mathrm{x}) = & \boxed{0} \longrightarrow & \boxed{1} \\
f_3(\mathrm{x}) = & \boxed{0} \longrightarrow & \boxed{0} \\
f_4(\mathrm{x}) = & \boxed{1} \longrightarrow & \boxed{0} \\
\cdot & \cdot \longrightarrow & \cdot \\
\cdot & \cdot \longrightarrow & \cdot \\
\cdot & \cdot \longrightarrow & \cdot \\
f_l(\mathrm{x}) = & \boxed{1} \longrightarrow & \boxed{1}
\end{array}
\left.\vphantom{\begin{array}{c}1\\1\\1\\1\\1\\1\\1\\1\end{array}}\right\} = w_3 \ \text{so} \ f'(\mathrm{x}) = c_3
$$

This method has yielded reductions of 10–23% in error rates on real problems.

Outline

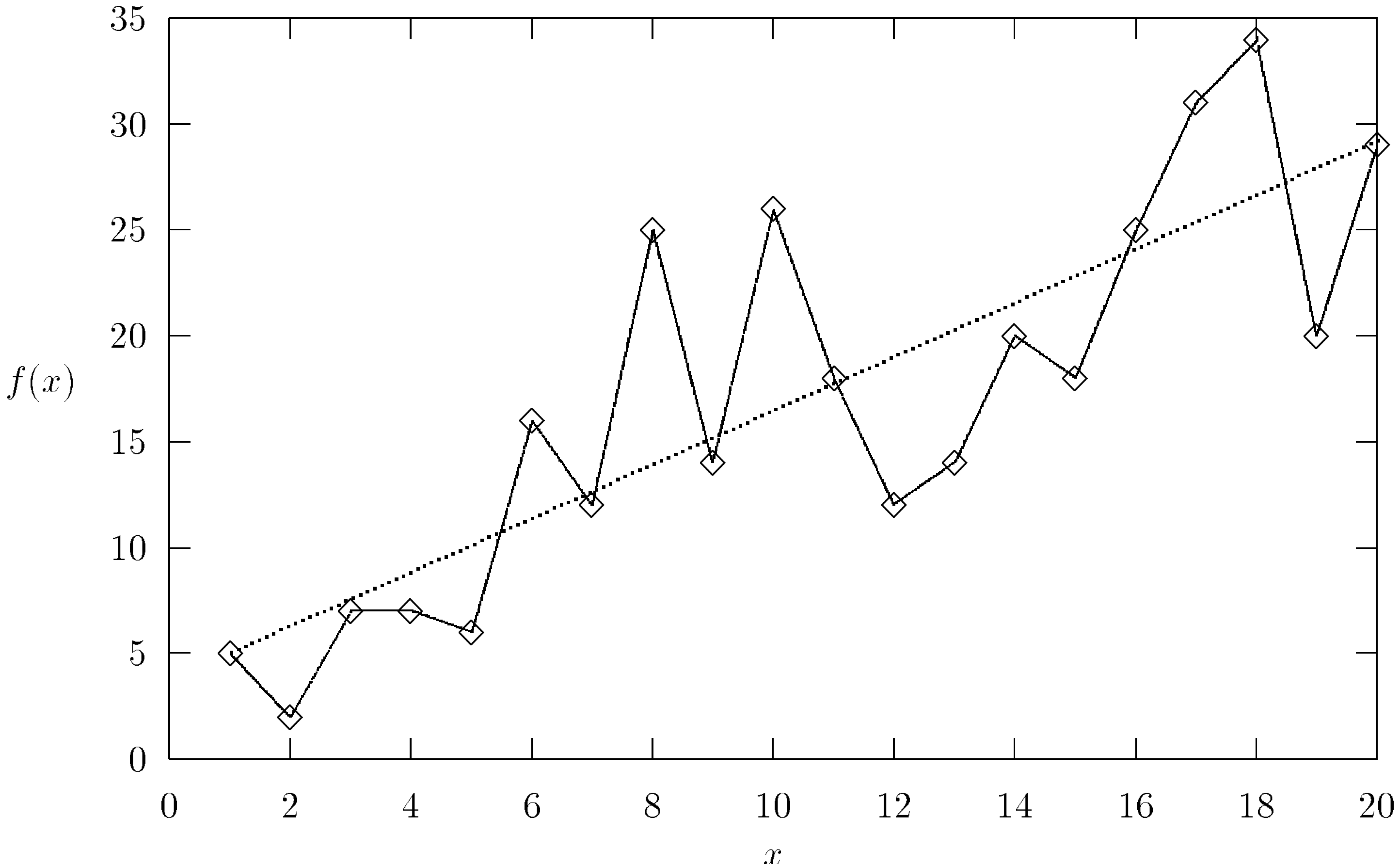
1. How can we define "bias"?
2. How can we implement biases?
3. How can we tell if the output of an algorithm is correct?
4. How can we tell if our bias is correct?
5. How can we recover from incorrect biases?

   [How can we deal with noise?]

6. How can we compare biases?

Noise and Overfitting

If algorithm A implements a preference bias, and
the training data S is noisy, then
A just keeps "walking along the preference order."
We can always find a good hypothesis $A(S) = f'$ that fits the data.



Controlling Overfitting

We could control overfitting if we could decide when to stop.

$$h_1 \prec h_2 \prec h_3 \prec \ldots \prec h_i \prec h_{i+1} \prec \ldots$$

Define a quantity $c(h)$ to be the "complexity" of hypothesis h.

Two approaches:

- **Minimium Description Length Principle**
  Stop when h_{i+1} is just encoding h_i plus noise.
- **Cross Validation**
  Stop when $c(h) > \alpha$.

Minimum Description Length Principle

$$P(h_i|S) \propto P(h_i) \cdot P(S|h_i) \quad \text{(Bayes Theorem)}$$

Let our complexity measure be $c(h_i) = -\lg P(h_i)$
This is the minimum number of bits needed to represent h_i.

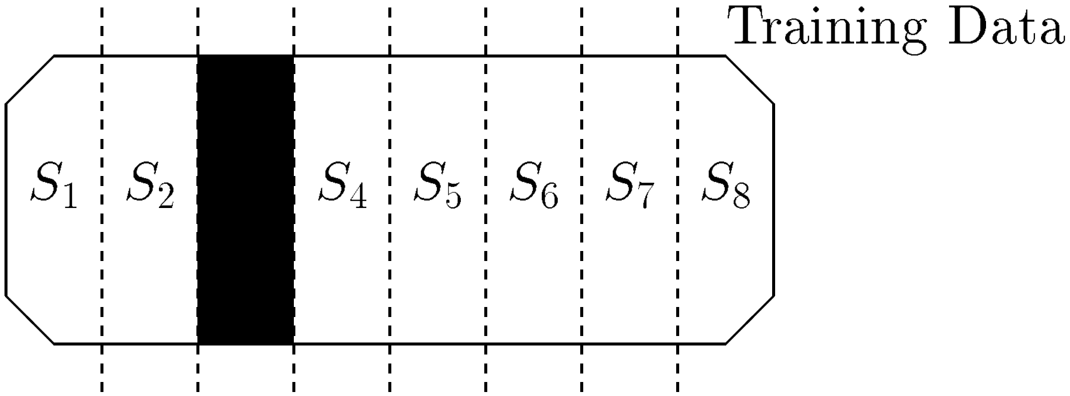
$$c(h_i|S) \propto c(h_i) + c(S|h_i)$$

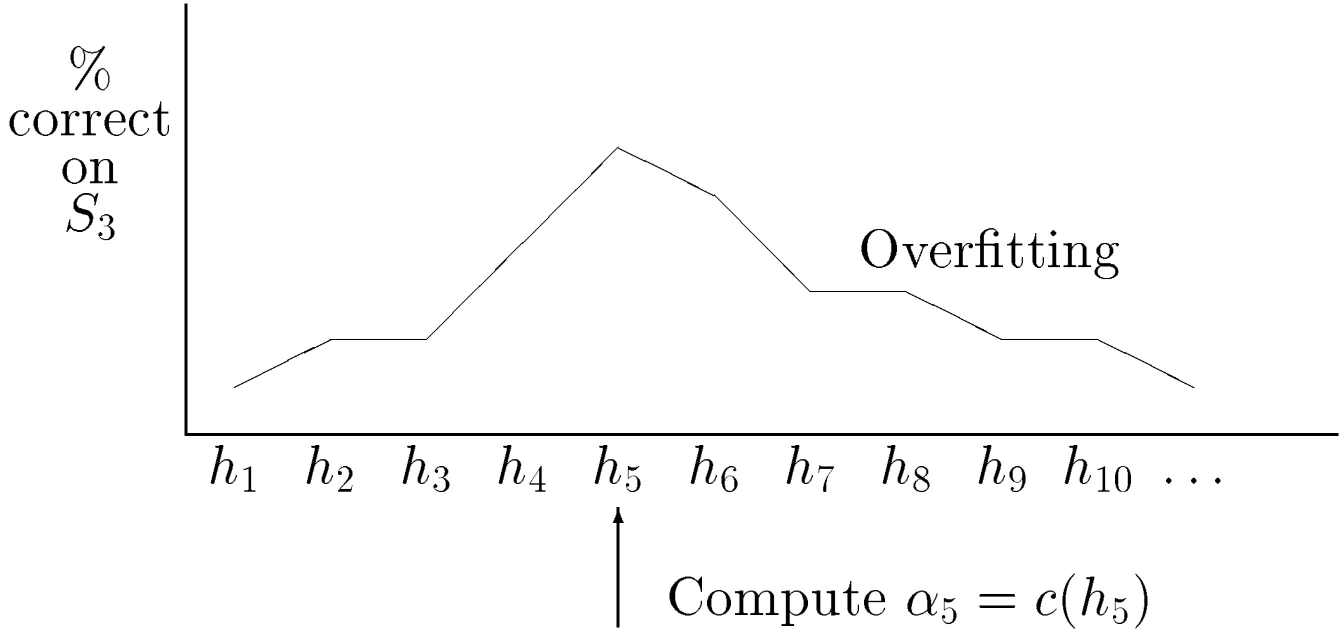
Choose the hypothesis h_i that minimizes the total description length:

$$\text{length}(h_i) + \text{length}(\text{training data}|h_i).$$

Intuition: random noise is not compressible.

Determining α by Cross Validation





α = Average of α_i.

Outline

1. How can we define "bias"?
2. How can we implement biases?
3. How can we tell if the output of an algorithm is correct?
4. How can we tell if our bias is correct?
5. How can we recover from incorrect biases?
   [How can we deal with noise?]
6. **How can we compare biases?**

Comparing Biases

Experimental Comparisons:

Choose ensemble of problems Π.

Carefully run algorithms on Π.

Many studies have been conducted.
These have not provided much insight.

For many tasks, many algorithms perform equally well.

Quandary 3

Do these (apparently different) algorithms actually implement very similar biases?

Open Problem: Compare biases analytically.

Framing the Problem

Quandary 4.
Are we solving the right problem?

- Prior Knowledge?
- Conventions Between Teacher and Student?
- Experimentation?
- Reinforcement Learning?

Prior Knowledge

We know a lot about the Martian surface.

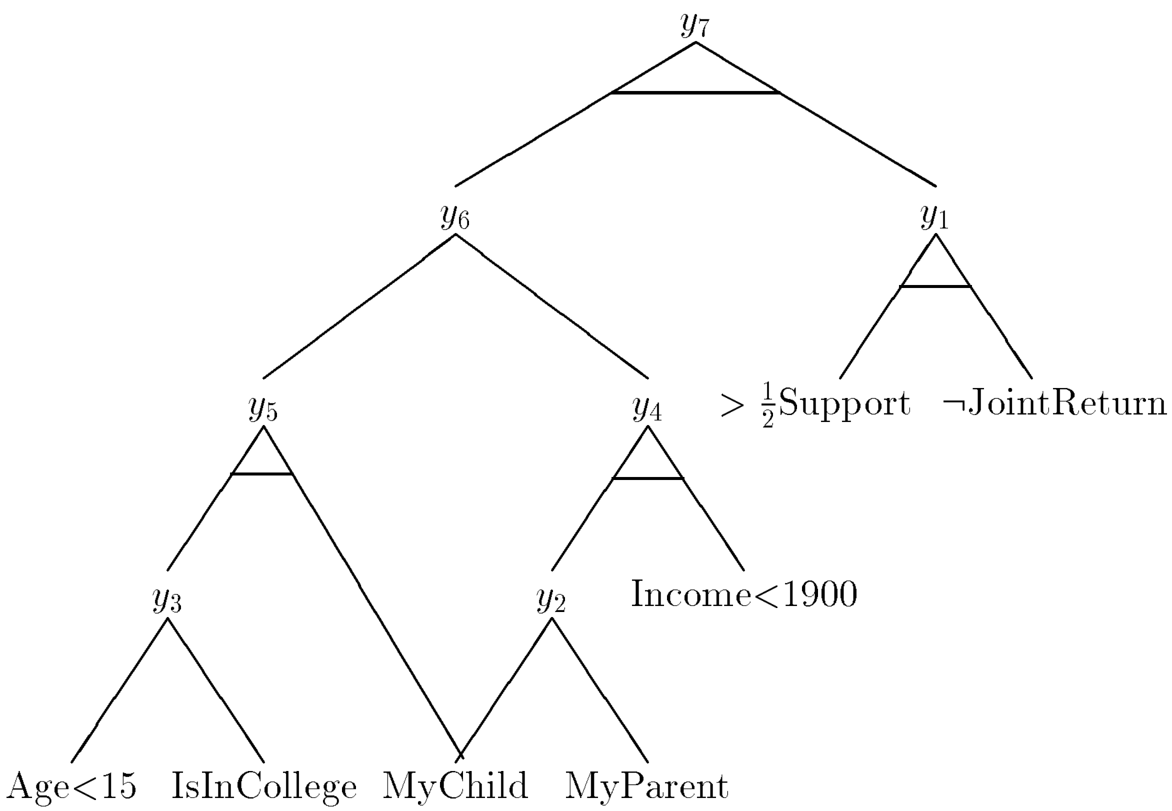
- Surface properties: temperature, reflectance
- Likely chemical/mineral composition
- Likely surface processes (wind storms, water flow, vulcanism)

We need to represent this for the robot.
We need learning algorithms that can exploit this knowledge.

Conventions Between Teacher and Student

Tutoring: Teacher gives series of "lessons"

Each lesson introduces one "subconcept"



All Boolean Formulas can be learned this way.

(VanLehn, 1983, 1987; Rivest & Sloan, 1988)

Experimentation

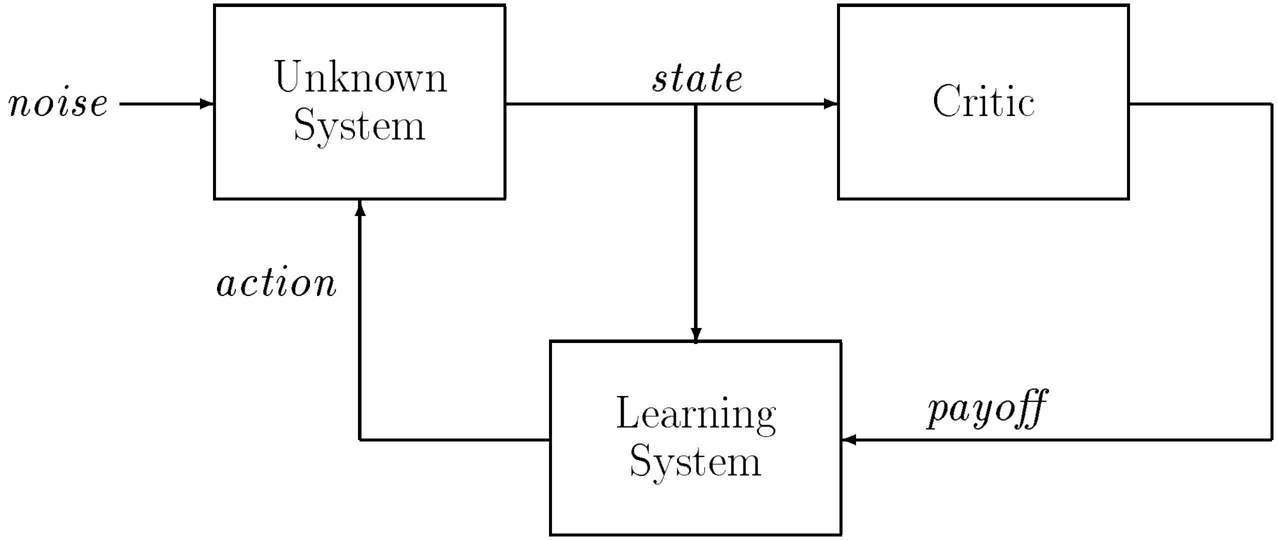
Human learners are active.

Does experimentation help?

Types of experiments:

- $f(\mathrm{x}_{new}) = ?$
- Specify region E.
  Nature randomly chooses $\mathrm{x} \in E$ and returns $f(\mathrm{x})$.
- Disassembly: f is a system.
  Disassemble the system to reveal subcomponents: $f_1, f_2, \ldots$.
- Control: $f(\mathrm{x}, g(\mathrm{y}))$
  g is stochastic but slowly varying.
  Hold y constant.

Reinforcement Learning



Goal: Learn $f(\text{state}) = \text{action}$
to maximize long-term payoff.

- Payoffs do not tell you what action to take.
- Payoffs may be delayed (temporal credit assignment).
- Tradeoff between exploration and exploitation.

Summary

- **There is no general learning algorithm.**
- **Machine learning = study of biases.**
  - Absolute vs. preference bias.
  - Implement via heuristic optimization algorithms.
  - Detect success by cross-validation.
  - Handle noise by controlling overfitting.
  - Recover from incorrect bias via redundancy.
- **Re-defining the problem:**
  - Incorporating prior knowledge.
  - Exploiting the learning situation.
  - Performing experiments.
  - Reinforcement learning.