

Last update: May 17, 2010

LEARNING FROM OBSERVATIONS

CMSC 421: CHAPTER 18: SECTIONS 1–3

Outline

- ◇ Learning agents
- ◇ Inductive learning
- ◇ Decision tree learning
- ◇ Measuring learning performance

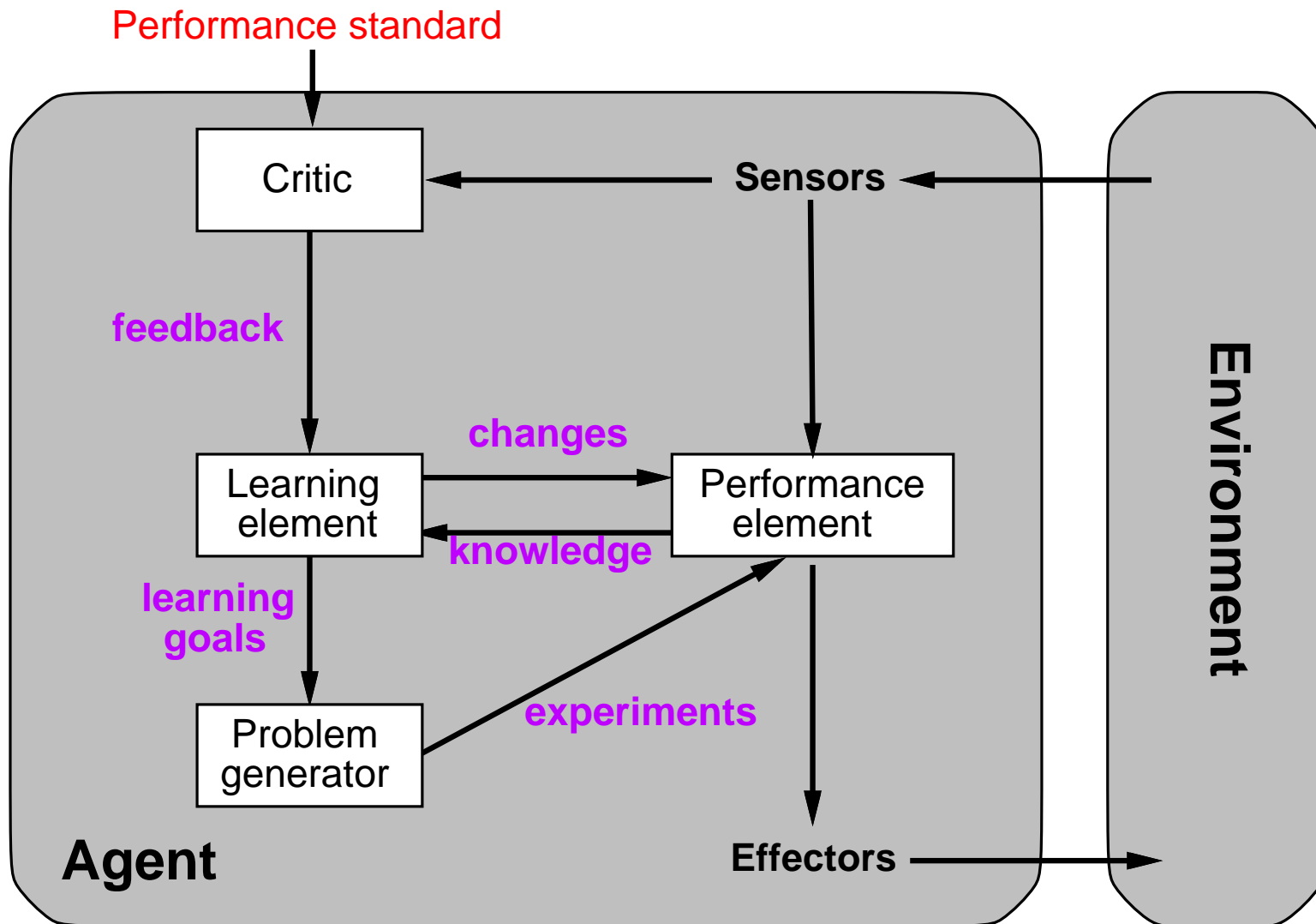
Learning

Learning is essential for unknown environments,
i.e., when designer lacks omniscience

Learning is useful as a system construction method,
i.e., expose the agent to reality rather than trying to write it down

Learning modifies the agent's decision mechanisms to improve performance

Learning agents



Learning element

Design of learning element is dictated by

- ◇ what type of performance element is used
- ◇ which functional component is to be learned
- ◇ how that functional component is represented
- ◇ what kind of feedback is available

Example scenarios:

Performance element	Component	Representation	Feedback
Alpha-beta search	Eval. fn.	Weighted linear function	Win/loss
Logical agent	Transition model	Successor-state axioms	Outcome
Utility-based agent	Transition model	Dynamic Bayes net	Outcome
Simple reflex agent	Percept-action fn	Neural net	Correct action

Supervised learning: correct answers for each instance

Reinforcement learning: occasional rewards

Inductive learning

Simplest form: learn a function from examples

f is the *target function*

An *example* is a pair $x, f(x)$, e.g., $\left(\begin{array}{c|c|c} O & O & X \\ \hline & X & \\ \hline X & & \end{array} , +1 \right)$

Problem: find a *hypothesis* h
such that $h \approx f$
given a *training set* of examples

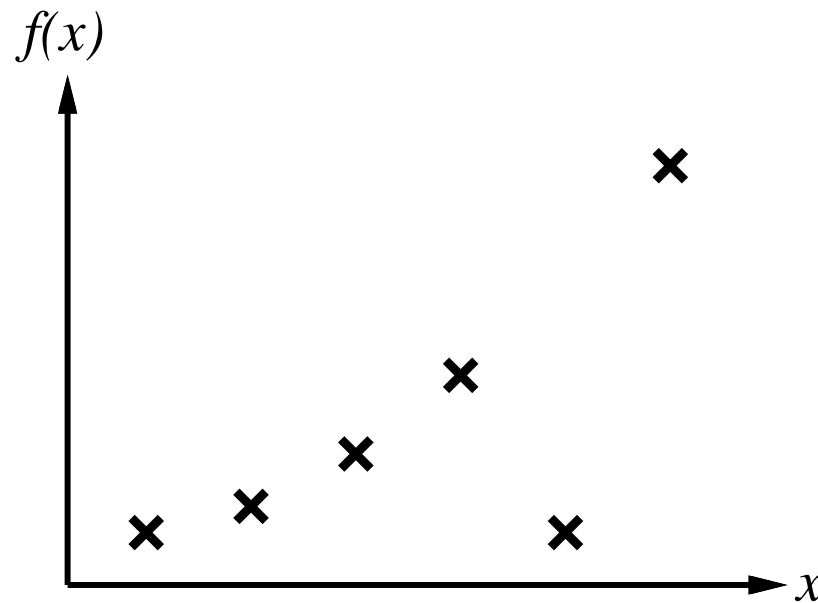
Highly simplified model of real learning:

- Ignores prior knowledge
- Assumes f is deterministic
- Assumes f and its arguments are observable
- Assumes examples are **given**
- Assumes that the agent **wants** to learn f

Inductive learning method

Construct/adjust h to agree with f on training set
(h is *consistent* if it agrees with f on all examples)

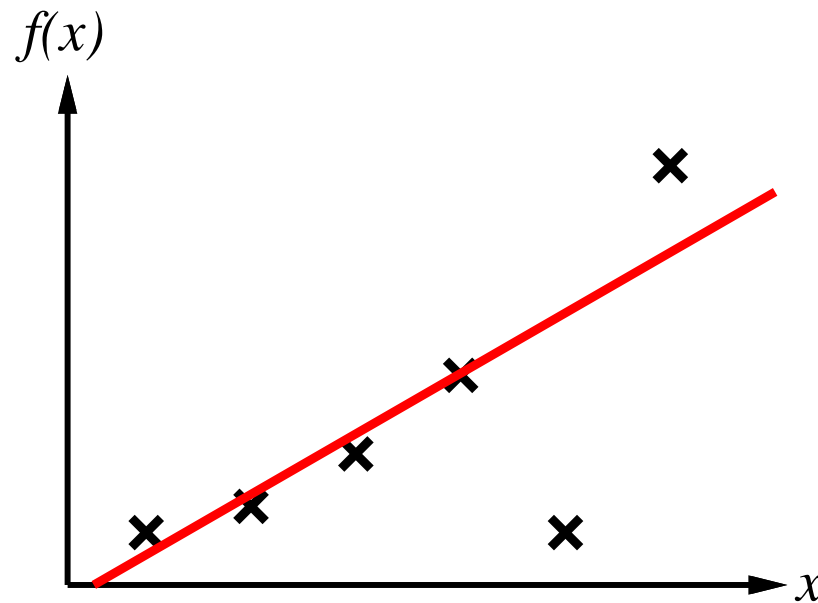
E.g., curve fitting:



Inductive learning method

Construct/adjust h to agree with f on training set
(h is *consistent* if it agrees with f on all examples)

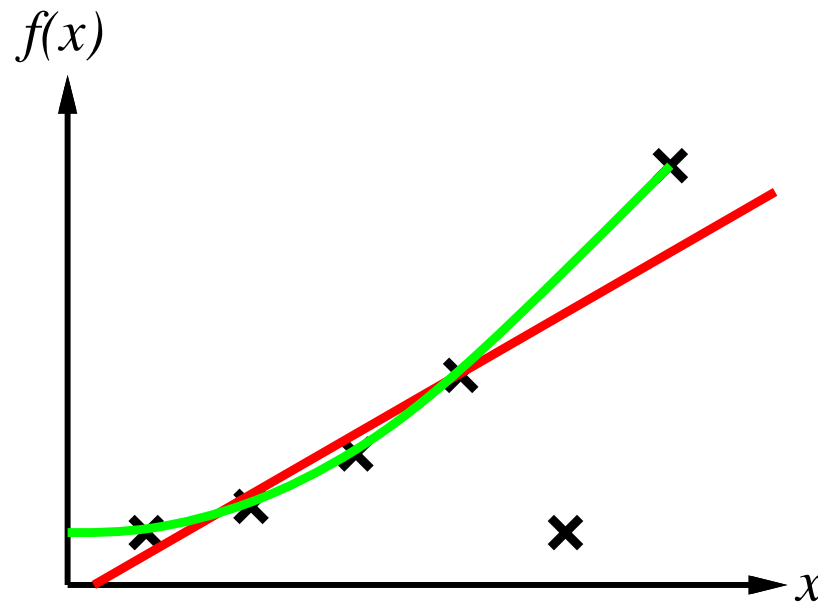
E.g., curve fitting:



Inductive learning method

Construct/adjust h to agree with f on training set
(h is *consistent* if it agrees with f on all examples)

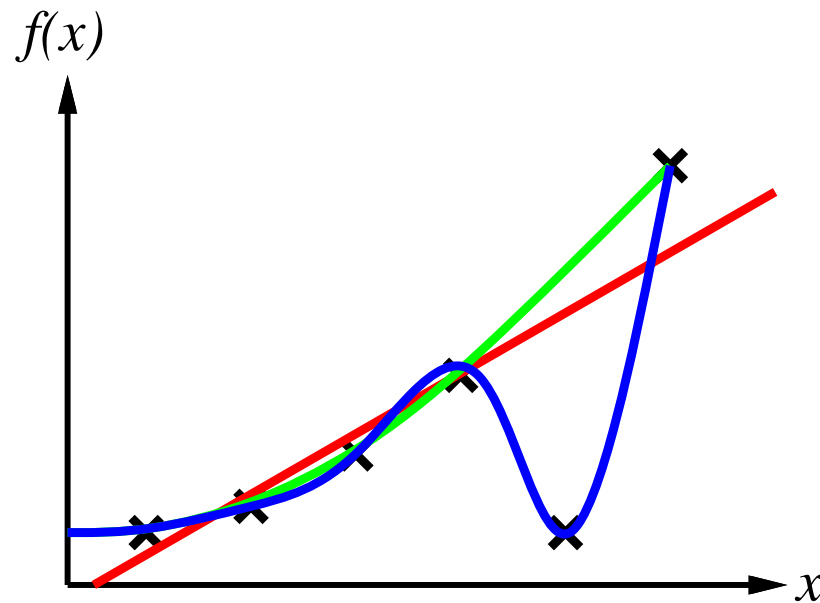
E.g., curve fitting:



Inductive learning method

Construct/adjust h to agree with f on training set
(h is *consistent* if it agrees with f on all examples)

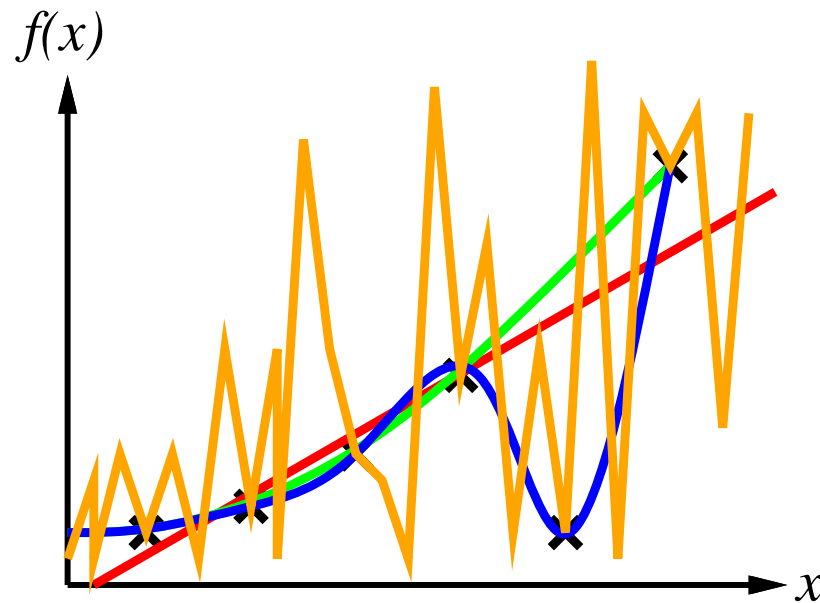
E.g., curve fitting:



Inductive learning method

Construct/adjust h to agree with f on training set
(h is *consistent* if it agrees with f on all examples)

E.g., curve fitting:



Ockham's razor

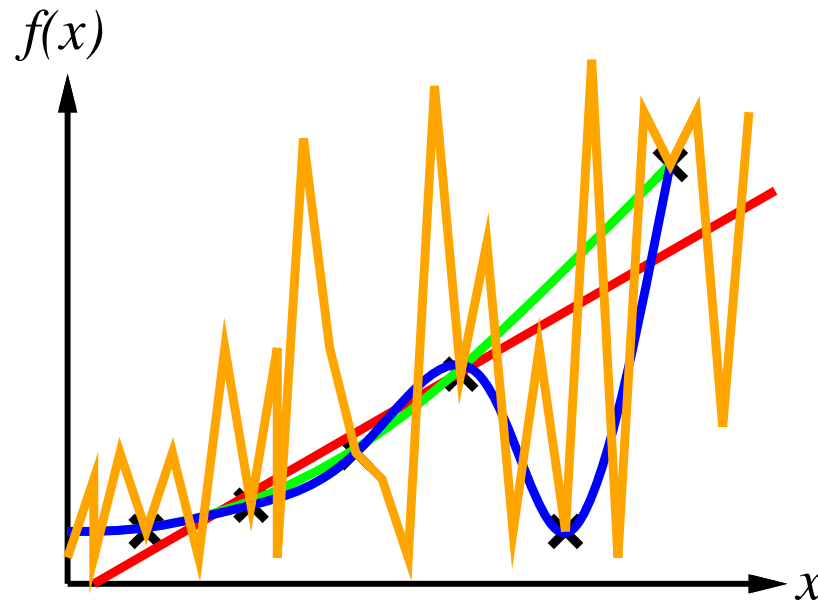
William of Ockham, fourteenth century:

“Pluralitas non est ponenda sine neccesitate”

This translates as

“entities should not be multiplied unnecessarily”

Maximize a combination of consistency and simplicity



Attribute-based representations

Examples described by *attribute values* (Boolean, discrete, continuous, etc.)
 E.g., examples of situations where a friend will/won't wait for a table:

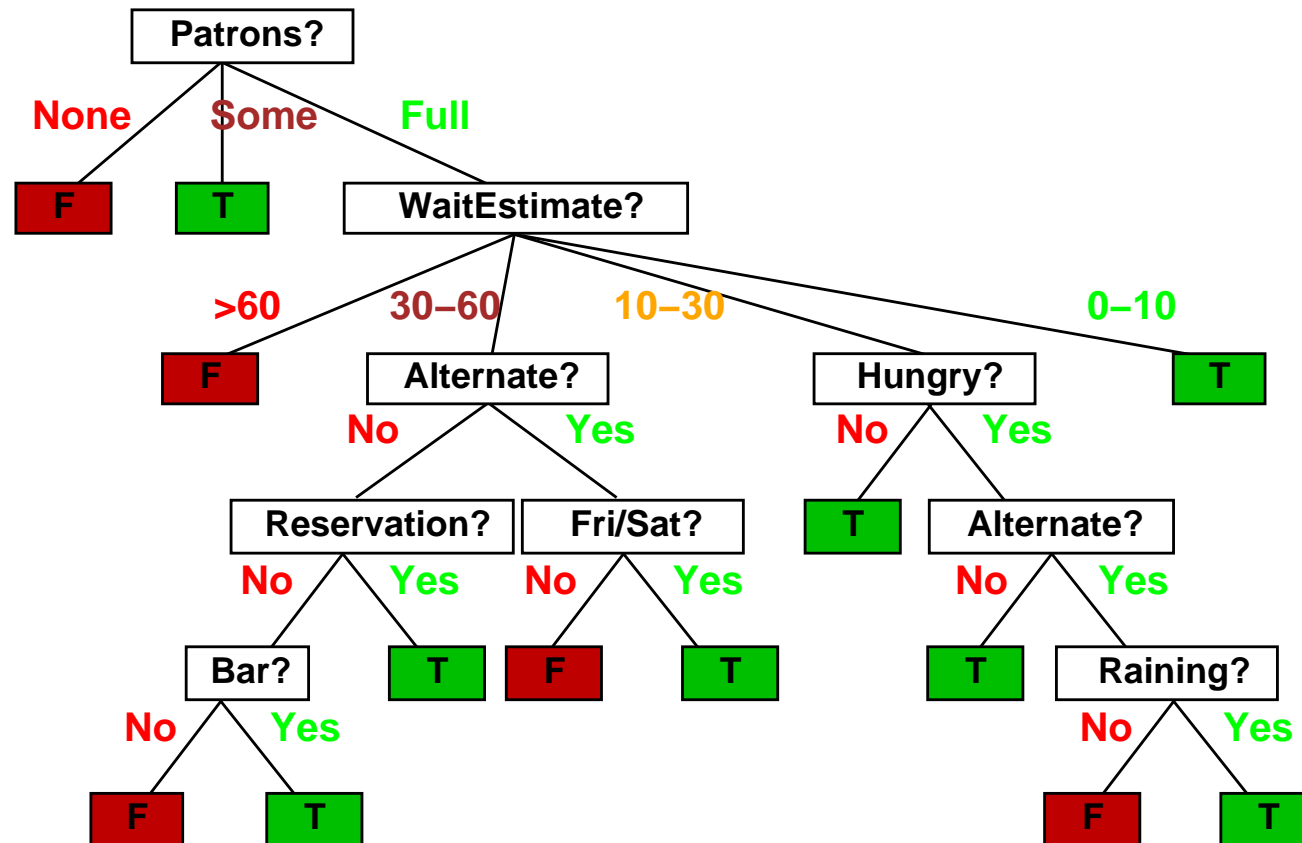
Example	Attributes										Target
	<i>Alt</i>	<i>Bar</i>	<i>Fri</i>	<i>Hun</i>	<i>Pat</i>	<i>Price</i>	<i>Rain</i>	<i>Res</i>	<i>Type</i>	<i>Est</i>	<i>WillWait</i>
X_1	<i>T</i>	<i>F</i>	<i>F</i>	<i>T</i>	<i>Some</i>	<i>\$\$\$</i>	<i>F</i>	<i>T</i>	<i>French</i>	<i>0–10</i>	<i>T</i>
X_2	<i>T</i>	<i>F</i>	<i>F</i>	<i>T</i>	<i>Full</i>	<i>\$</i>	<i>F</i>	<i>F</i>	<i>Thai</i>	<i>30–60</i>	<i>F</i>
X_3	<i>F</i>	<i>T</i>	<i>F</i>	<i>F</i>	<i>Some</i>	<i>\$</i>	<i>F</i>	<i>F</i>	<i>Burger</i>	<i>0–10</i>	<i>T</i>
X_4	<i>T</i>	<i>F</i>	<i>T</i>	<i>T</i>	<i>Full</i>	<i>\$</i>	<i>F</i>	<i>F</i>	<i>Thai</i>	<i>10–30</i>	<i>T</i>
X_5	<i>T</i>	<i>F</i>	<i>T</i>	<i>F</i>	<i>Full</i>	<i>\$\$\$</i>	<i>F</i>	<i>T</i>	<i>French</i>	<i>>60</i>	<i>F</i>
X_6	<i>F</i>	<i>T</i>	<i>F</i>	<i>T</i>	<i>Some</i>	<i>\$\$</i>	<i>T</i>	<i>T</i>	<i>Italian</i>	<i>0–10</i>	<i>T</i>
X_7	<i>F</i>	<i>T</i>	<i>F</i>	<i>F</i>	<i>None</i>	<i>\$</i>	<i>T</i>	<i>F</i>	<i>Burger</i>	<i>0–10</i>	<i>F</i>
X_8	<i>F</i>	<i>F</i>	<i>F</i>	<i>T</i>	<i>Some</i>	<i>\$\$</i>	<i>T</i>	<i>T</i>	<i>Thai</i>	<i>0–10</i>	<i>T</i>
X_9	<i>F</i>	<i>T</i>	<i>T</i>	<i>F</i>	<i>Full</i>	<i>\$</i>	<i>T</i>	<i>F</i>	<i>Burger</i>	<i>>60</i>	<i>F</i>
X_{10}	<i>T</i>	<i>T</i>	<i>T</i>	<i>T</i>	<i>Full</i>	<i>\$\$\$</i>	<i>F</i>	<i>T</i>	<i>Italian</i>	<i>10–30</i>	<i>F</i>
X_{11}	<i>F</i>	<i>F</i>	<i>F</i>	<i>F</i>	<i>None</i>	<i>\$</i>	<i>F</i>	<i>F</i>	<i>Thai</i>	<i>0–10</i>	<i>F</i>
X_{12}	<i>T</i>	<i>T</i>	<i>T</i>	<i>T</i>	<i>Full</i>	<i>\$</i>	<i>F</i>	<i>F</i>	<i>Burger</i>	<i>30–60</i>	<i>T</i>

Classification of examples is *positive* (T) or *negative* (F)

Decision trees

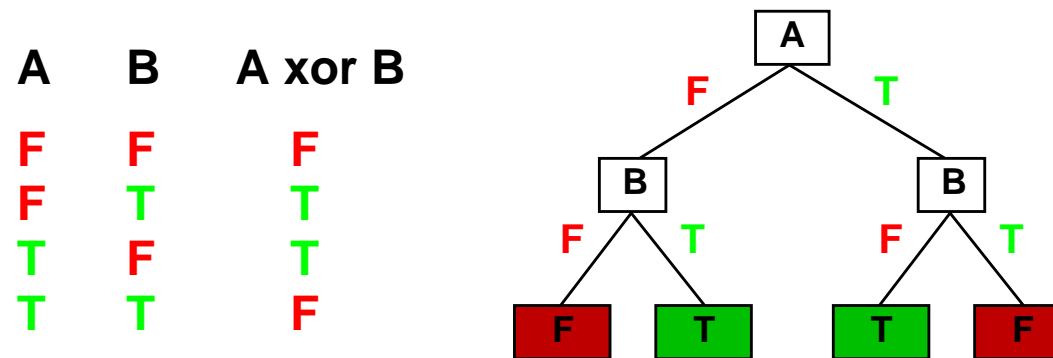
One possible representation for hypotheses

E.g., here is your friend's "true" tree for deciding whether to wait:



Expressiveness

Decision trees can express any function of the input attributes.
E.g., for Boolean functions, truth table row \rightarrow path to leaf:



Trivially, there's a consistent decision tree for any training set that has a different path to a leaf for each example (unless f nondeterministic in x) but it probably won't generalize to new examples

Ockham's razor \Rightarrow

Prefer to find more **compact** decision trees

Hypothesis spaces

How many distinct decision trees with n Boolean attributes?

Hypothesis spaces

How many distinct decision trees with n Boolean attributes?

= number of Boolean functions

Hypothesis spaces

How many distinct decision trees with n Boolean attributes?

= number of Boolean functions

= number of distinct truth tables with 2^n rows

Hypothesis spaces

How many distinct decision trees with n Boolean attributes?

= number of Boolean functions

= number of distinct truth tables with 2^n rows = 2^{2^n}

Hypothesis spaces

How many distinct decision trees with n Boolean attributes?

= number of Boolean functions

= number of distinct truth tables with 2^n rows = 2^{2^n}

E.g., with 6 Boolean attributes, there are 18,446,744,073,709,551,616 trees

Hypothesis spaces

How many distinct decision trees with n Boolean attributes?

= number of Boolean functions

= number of distinct truth tables with 2^n rows = 2^{2^n}

E.g., with 6 Boolean attributes, there are 18,446,744,073,709,551,616 trees

How many purely conjunctive hypotheses (e.g., $Hungry \wedge \neg Rain$)?

Hypothesis spaces

How many distinct decision trees with n Boolean attributes?

= number of Boolean functions

= number of distinct truth tables with 2^n rows = 2^{2^n}


E.g., with 6 Boolean attributes, there are 18,446,744,073,709,551,616 trees

How many purely conjunctive hypotheses (e.g., $Hungry \wedge \neg Rain$)?

Each attribute can be in (positive), in (negative), or out

$\Rightarrow 3^n$ distinct conjunctive hypotheses

More expressive hypothesis space

- increases chance that target function can be expressed 
- increases number of hypotheses consistent w/ training set

\Rightarrow may get worse predictions 

Decision tree learning

Aim: find a small tree consistent with the training examples

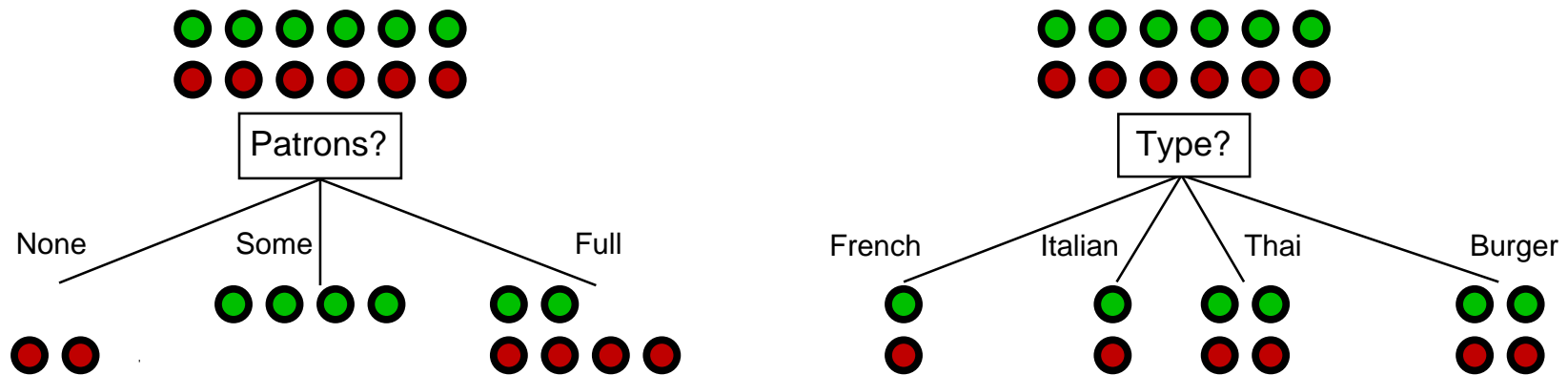
Idea: construct the tree by going downward recursively from the top

At each node, choose “most significant” attribute as root of (sub)tree

```
function DTL(examples, attributes, default) returns a decision tree
  if examples is empty then return default
  else if all examples have the same classification then return the classification
  else if attributes is empty then return MODE(examples)
  else
    best ← CHOOSE-ATTRIBUTE(attributes, examples)
    tree ← a new decision tree with root test best
    for each value  $v_i$  of best do
       $examples_i \leftarrow \{\text{elements of } examples \text{ with } best = v_i\}$ 
      subtree ← DTL( $examples_i$ , attributes – best, MODE(examples))
      add a branch to tree with label  $v_i$  and subtree subtree
  return tree
```

Choosing an attribute

Idea: a good attribute splits the examples into subsets that are (ideally) “all positive” or “all negative”



Patrons? is a better choice—gives **information** about the classification

Information

Information answers questions

The more clueless I am about the answer initially, the more information is contained in the answer

Scale: 1 bit = answer to Boolean question with prior $\langle 0.5, 0.5 \rangle$
(this is the maximum possible amount of info in a Boolean answer)

Information in an answer when prior distribution is $\langle P_1, \dots, P_n \rangle$ is

$$H(\langle P_1, \dots, P_n \rangle) = \sum_{i=1}^n -P_i \log_2 P_i$$

where $0 \log_2 0$ is taken to be 0

Also called the *entropy* of the prior distribution

$$H(\langle 0.5, 0.5 \rangle) = -0.5 \log_2 0.5 - 0.5 \log_2 0.5 = 0.5 + 0.5 = 1$$

$$H(\langle 1, 0 \rangle) = -1 \log_2 1 - 0 \log_2 0 = 0 + 0 = 0$$

Choosing an attribute

We're building the decision tree node by node, going top down

At node N , suppose there's a set E of p positive and n negative examples

Entropy at N is $H(\langle p/(p+n), n/(p+n) \rangle)$

\Rightarrow need $H(\langle p/(p+n), n/(p+n) \rangle)$ bits to classify an example

E.g., for the 12 restaurant examples, $p = n = 6$ so we need 1 bit

What attribute A to use to classify these examples?

If A has k possible values, this splits E into subsets $E_1, \dots, E_i, \dots, E_k$

Each of them will be a new node of the decision tree

Each (we hope) will need less information to complete the classification

Basic idea:

- ◇ Compute the **expected** number of additional bits we'll need if we use A
- ◇ Choose the attribute A that minimizes the above value

Choosing an attribute (continued)

Expected number of additional bits we'll need after using A :

$$\begin{aligned} \text{Remainder}(A) &= \text{weighted average over all the subsets created by } A \\ &= \sum_i P(\text{example is in } E_i)(\text{entropy of } E_i) \end{aligned}$$

If E_i has p_i positive and n_i negative examples, then

$$\begin{aligned} P(\text{example is in } E_i) &= (p_i + n_i)/(p + n) \\ \text{entropy of } E_i &= H(\langle p_i/(p_i + n_i), n_i/(p_i + n_i) \rangle) \end{aligned}$$

Expected number of additional bits we'll need if we use A

$$= \sum_i \frac{p_i + n_i}{p + n} H(\langle p_i/(p_i + n_i), n_i/(p_i + n_i) \rangle)$$

Want to choose an attribute A that minimizes the above value

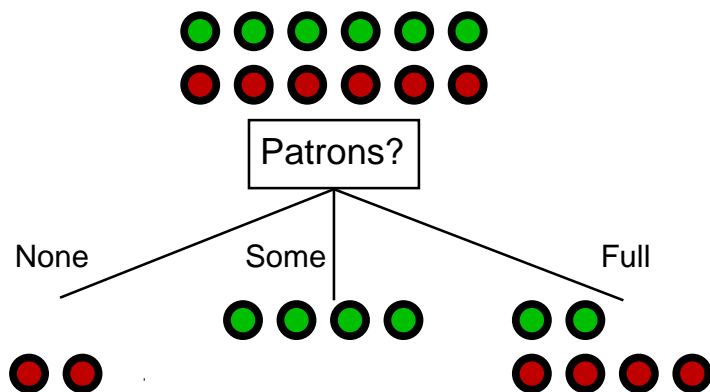
Information, continued

Choose attribute that minimizes the following quantity:

$$\sum_i \frac{p_i + n_i}{p + n} H(\langle p_i / (p_i + n_i), n_i / (p_i + n_i) \rangle)$$

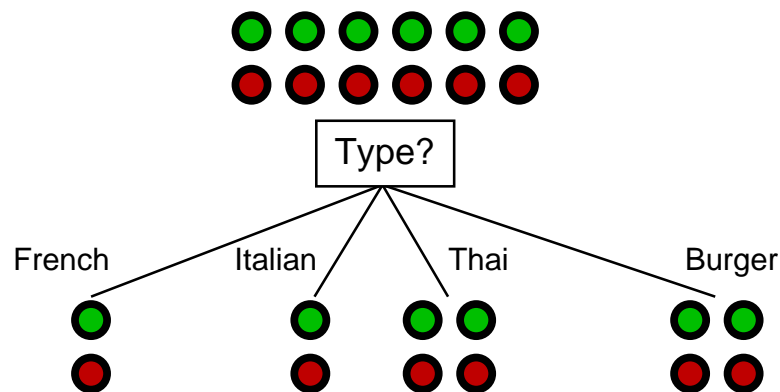
For *Patrons?*, this is

$$\begin{aligned} & \left(\frac{2}{12} + \frac{4}{12}\right) H(\langle 0, 1 \rangle) + \frac{6}{12} H(\langle 2/6, 4/6 \rangle) \\ &= \frac{1}{2} 0 + \frac{1}{2} 0.918 \\ &= 0.459 \text{ bits} \end{aligned}$$



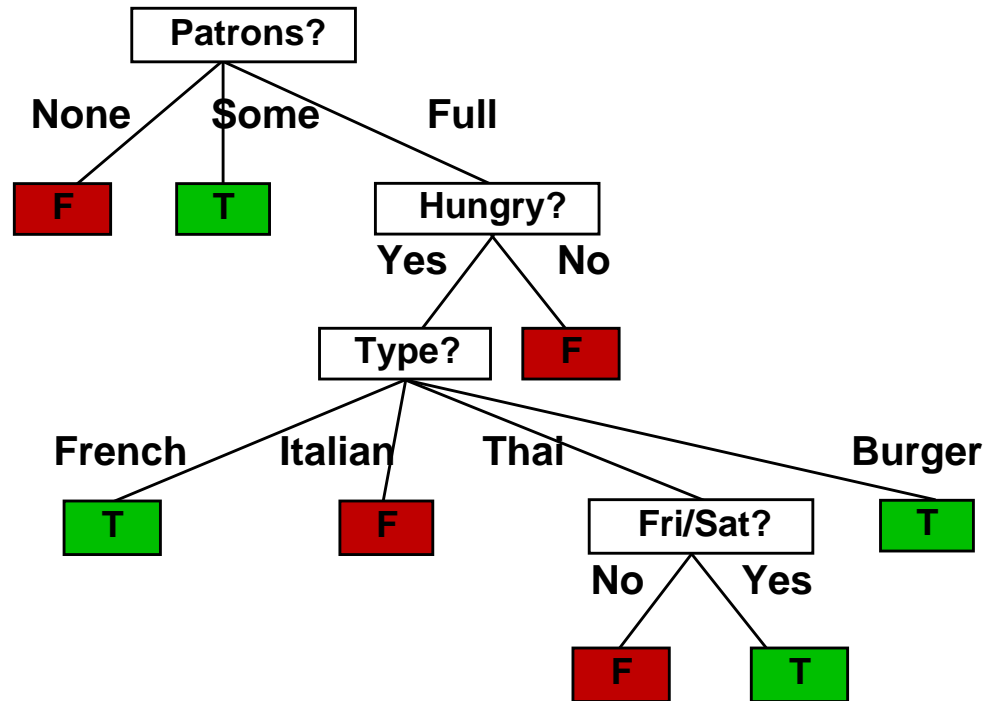
For *Type?*, it is

$$\begin{aligned} & \left(\frac{2}{12} + \frac{2}{12} + \frac{4}{12} + \frac{4}{12}\right) H(\langle 0.5, 0.5 \rangle) \\ &= \frac{12}{12} 1 \\ &= 1 \text{ bit} \end{aligned}$$



Example, continued

Decision tree learned from the 12 examples:



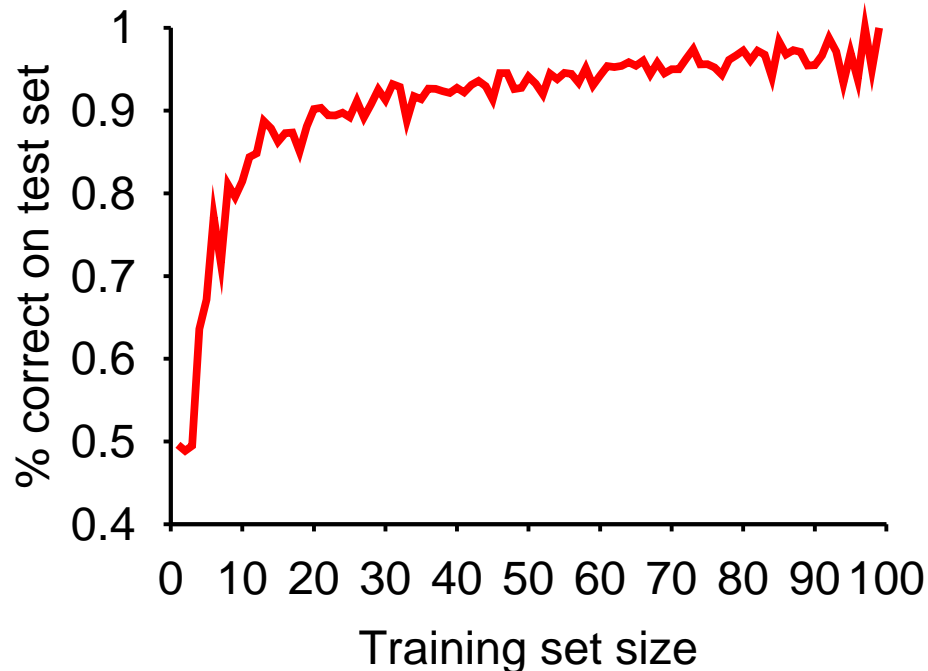
Substantially simpler than the “true” tree shown earlier.
The small amount of data doesn’t justify a more complex hypothesis

Performance measurement

How do we know that $h \approx f$? (Hume's **Problem of Induction**)

- 1) Use theorems of computational/statistical learning theory
- 2) Try h on a *test set*: a different set of examples from the same probability distribution

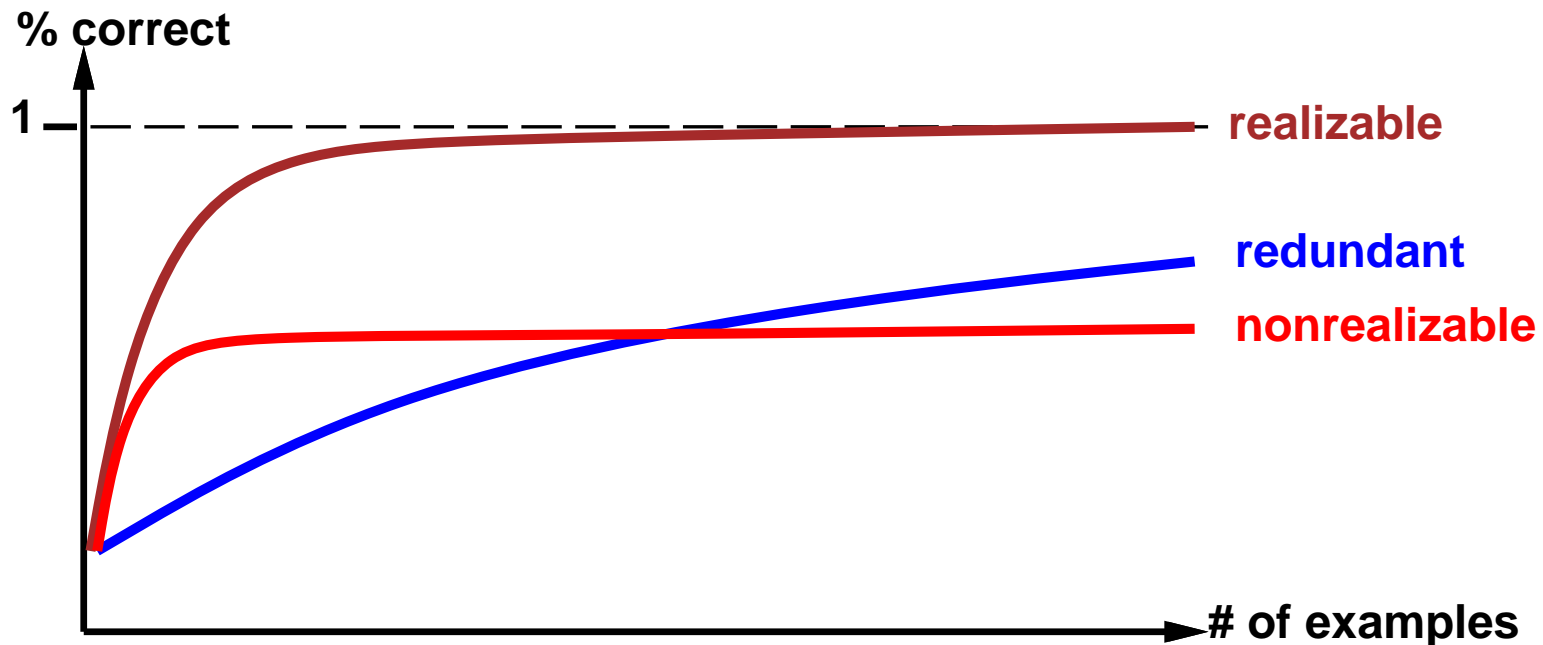
Learning curve = % correct on test set as a function of training set size



Performance measurement, continued

Learning curve depends on

- *realizable* (can express target function) vs. *non-realizable*
non-realizability can be due to missing attributes
or restricted hypothesis class (e.g., thresholded linear function)
- redundant expressiveness (e.g., loads of irrelevant attributes)



Summary

Learning needed for unknown environments, lazy designers

Learning agent = performance element + learning element

Learning method depends on type of performance element, available feedback, type of component to be improved, and its representation

For supervised learning, the aim is to find a simple hypothesis that is approximately consistent with training examples

Decision tree learning using information gain

Learning performance = prediction accuracy measured on test set