Learning

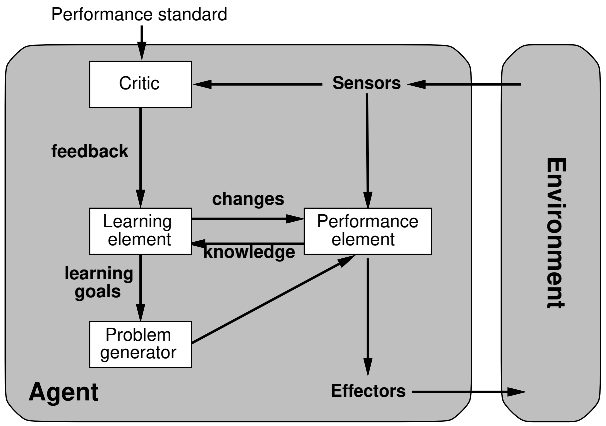
Outline

- General Model of Learning Agents
- Inductive Learning
- Learning Decision Trees
- Learning General Logical Descriptions

General Model of Learning Agents

Four conceptual components of a **learning agent**:

- Performance element (a conventional agent)
- Learning element
- Critic
- Problem generator



Design of the Learning Element

The design of the learning element is affected by

- Components of the performance element
- Representation of the components
- Available feedback

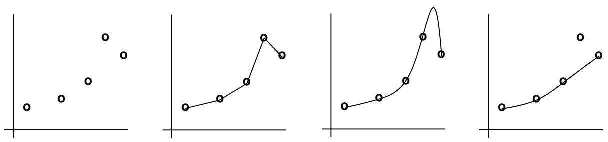
  Supervised/unsupervised learning
  Reinforcement learning
- Prior knowledge

Performance element = a function from a state to a state.
Goal is a boolean function on states.

Inductive Learning

- An example is a pair $\langle x, f(x)\rangle$
- Pure inductive inference (induction):
  Given a collection of examples of f,
  return a function h that approximates f.
- Such a function h is called a hypothesis.
- Incremental learning:
  Hypothesis is updated whenever a new example arrives.

- Example: $\langle x, y\rangle$ are points such that $y = f(x)$.
- The task is to find $h(x)$ that fits the points well.



(a) (b) (c) (d)

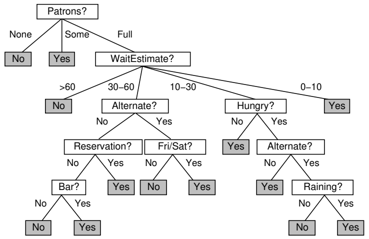
- Any preference for one hypothesis over another (beyond mere consistency with examples) is called a **bias**.

A simple reflex agent taught by a teacher

```
global examples ← {}

function REFLEX-PERFORMANCE-ELEMENT(percept) returns an action
  if (percept, a) in examples then return a
  else
     h ← INDUCE(examples)
     return h(percept)

procedure REFLEX-LEARNING-ELEMENT(percept, action)
  inputs: percept, feedback percept
          action, feedback action

  examples ← examples ∪ {(percept,action)}
```


Learning Decision Trees

- A decision tree represents a function from an n-dimensional set of attributes to the set $\{\text{Yes}, \text{No}\}$ (a Boolean function).



Expressiveness of Decision Trees

- $\forall r(\text{Patrons}(r, \text{Full}) \wedge \text{WaitEstimate}(r, 0-10) \wedge \text{Hungry}(r, \text{N}) \rightarrow \text{WillWait}(r))$

  Decision trees are effectively propositional.
- Any boolean function can be written as a decision tree.

  Such a representation may require an exponential space.
- For instance, it is difficult to represent parity and majority functions as decision trees.
- There are 2^{2^n} different Boolean functions (with n Boolean attributes). E.g. 2^{2^6} is about 2×10^{19}.

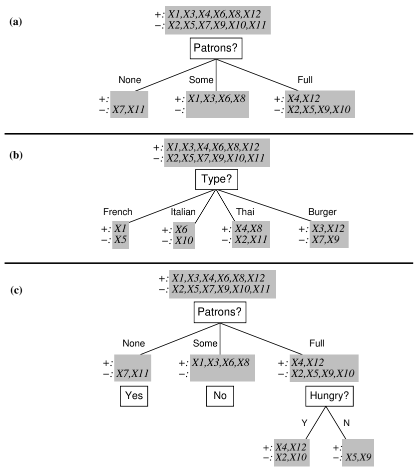
Inducing decision trees from examples

Example	Attributes										Goal
	Alt	*Bar*	*Fri*	*Hun*	*Pat*	*Price*	*Rain*	*Res*	*Type*	*Est*	*WillWait*
X_1	Yes	No	No	Yes	Some	$$$	No	Yes	French	0–10	Yes
X_2	Yes	No	No	Yes	Full	$	No	No	Thai	30–60	No
X_3	No	Yes	No	No	Some	$	No	No	Burger	0–10	Yes
X_4	Yes	No	Yes	Yes	Full	$	No	No	Thai	10–30	Yes
X_5	Yes	No	Yes	No	Full	$$$	No	Yes	French	>60	No
X_6	No	Yes	No	Yes	Some	$$	Yes	Yes	Italian	0–10	Yes
X_7	No	Yes	No	No	None	$	Yes	No	Burger	0–10	No
X_8	No	No	No	Yes	Some	$$	Yes	Yes	Thai	0–10	Yes
X_9	No	Yes	Yes	No	Full	$	Yes	No	Burger	>60	No
X_{10}	Yes	Yes	Yes	Yes	Full	$$$	No	Yes	Italian	10–30	No
X_{11}	No	No	No	No	None	$	No	No	Thai	0–10	No
X_{12}	Yes	Yes	Yes	Yes	Full	$	No	No	Burger	30–60	Yes

- Both **positive** and **negative** examples wrt. **WillWait**
- **Ockham's razor**: most likely hypothesis is the simplest one that is consistent with all observations.

An algorithm for learning decision trees

```
function DECISION-TREE-LEARNING(examples, attributes, default) returns a decision tree
  inputs: examples, set of examples
          attributes, set of attributes
          default, default value for the goal predicate

  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 MAJORITY-VALUE(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 ← {elements of examples with best = v_i}
        subtree ← DECISION-TREE-LEARNING(examples_i, attributes − best,
                                         MAJORITY-VALUE(examples))
        add a branch to tree with label v_i and subtree subtree
     end
     return tree
```

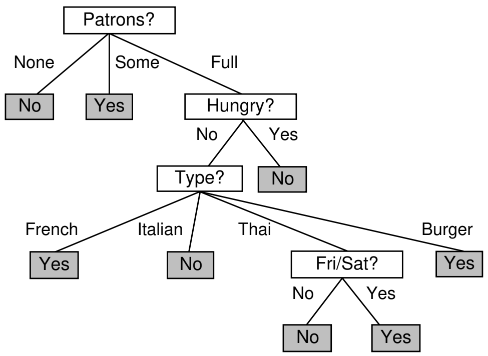



Selecting Attributes

The goal is to find an attribute that classifies given examples as much as possible.

E.g. **Patrons** gives a better classification than **Type**.

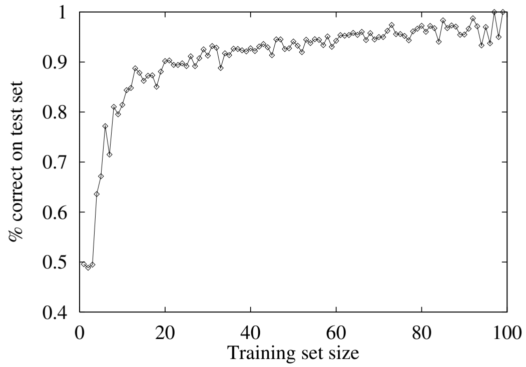
The decision tree induced from the training set



Assessing the performance of the learning element

1. Collect a large set of examples
2. Partitioning: **training set** and **test set**
3. Use the learning algorithm to generate a hypothesis H
4. Measure the percentage of examples in the test set that are correctly classified by H
5. Repeat steps 1-4 for randomly selected training sets of increasing size

☞ **Learning curve** for a specific algorithm



Using Information Theory

➤ Information content

$$I(P(v_1),\ldots,P(v_n)) = \sum_{i=1}^{n} -P(v_i)\log_2 P(v_i) \text{ (bits)}$$

where $P(v_1),\ldots,P(v_n)$ give the probabilities of the possible values $v_1,\ldots,v_n$ of the given variable.

➤ E.g. $I(\frac{1}{2},\frac{1}{2}) = -\frac{1}{2}\log_2\frac{1}{2} - \frac{1}{2}\log_2\frac{1}{2} = 1$ bits.

But $I(\frac{1}{100},\frac{99}{100}) \approx 0.08$ bits.

- For decision trees:
  $$\text{Gain}(A) = I(\frac{p}{p+n}, \frac{n}{p+n}) - \text{Remainder}(A)$$
  where
  $$\text{Remainder}(A) = \sum_{i=1}^{v} \frac{p_i + n_i}{p+n} \times I(\frac{p_i}{p_i+n_i}, \frac{n_i}{p_i+n_i}).$$
- $\text{Gain(Patrons)} \approx 0.541$ bits and $\text{Gain(Type)} = 0$ bits.

☞ Knowing the value of **Patrons** gives us more information than knowing the value of **Type**.

Learning General Logical Descriptions

For the decision tree that was learned earlier (H_r):

$$\begin{aligned}
&\forall r(\text{WillWait}(r) \leftrightarrow \\
&\quad \text{P}(r, \text{Some}) \lor \\
&\quad (\text{P}(r, \text{Full}) \land \neg\text{Hungry}(r) \land \text{T}(r, \text{French})) \lor \\
&\quad (\text{P}(r, \text{Full}) \land \neg\text{Hungry}(r) \land \text{T}(r, \text{Thai}) \land \text{Fri/Sat}(r)) \lor \\
&\quad (\text{P}(r, \text{Full}) \land \neg\text{Hungry}(r) \land \text{T}(r, \text{Burger}))\ )
\end{aligned}$$

Hypothesis space $\mathbf{H} = \{H_1, \ldots, H_n\}$ of a learning algorithm: $H_1 \lor \ldots \lor H_n$ is believed.

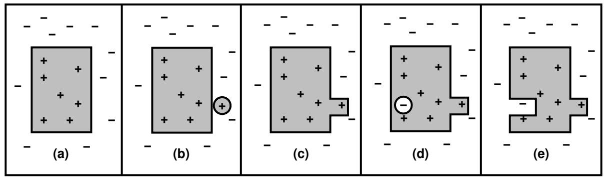
Representation of examples

- Our first example $\text{Alternate}(X_1) \land \neg\text{Bar}(X_1) \land \neg\text{Fri/Sat}(X_1) \land \text{Hungry}(X_1) \land \ldots$ gets the classification $\text{WillWait}(X_1)$.
- **False negative/positive** example for a hypothesis

  ☞ the example and the hypothesis in question are logically inconsistent with each other.
- **Generalization/specialization** of the hypothesis.

  ☞ Removing/adding conditions from/to the hypothesis.

Current-best-hypothesis search




```
function CURRENT-BEST-LEARNING(examples) returns a hypothesis

  H ← any hypothesis consistent with the first example in examples
  for each remaining example in examples do
      if e is false positive for H then
          H ← choose a specialization of H consistent with examples
      else if e is false negative for H then
          H ← choose a generalization of H consistent with examples
      if no consistent specialization/generalization can be found then fail
  end
  return H
```

The algorithm works as follows ...

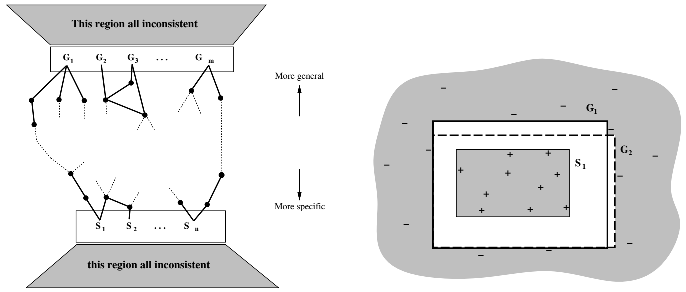
- H_1: $\forall x(\text{WillWait}(x) \leftrightarrow \text{Alternate}(x))$
- H_2: $\forall x(\text{WillWait}(x) \leftrightarrow \text{Alternate}(x) \wedge \text{P}(x, \text{Some}))$
- H_3: $\forall x(\text{WillWait}(x) \leftrightarrow \text{P}(x, \text{Some}))$
- H_4: $\forall x(\text{WillWait}(x) \leftrightarrow$
  $\text{P}(x, \text{Some}) \vee (\text{P}(x, \text{Full}) \wedge \text{Fri/Sat}(x)))$

Also H_4': $\forall x(\text{WillWait}(x) \leftrightarrow \neg\text{WaitEstimate}(x, 30-60))$ and H_4'': $\forall x(\text{WillWait}(x) \leftrightarrow$ $\text{P}(x, \text{Some}) \vee (\text{P}(x, \text{Full}) \wedge \text{WaitEstimate}(x, 10-30)))$ are consistent with the examples.

Least-commitment search

```
function VERSION-SPACE-LEARNING(examples) returns a version space
  local variables: V, the version space: the set of all hypotheses

  V ← the set of all hypotheses
  for each example e in examples do
     if V is not empty then V ← VERSION-SPACE-UPDATE(V, e)
  end
  return V
------------------------------------------------------------
function VERSION-SPACE-UPDATE(V, e) returns an updated version space
  V ← {h ∈ V : h is consistent with e}
```

- The algorithm finds a subset of V that is consistent with examples.
- Problem: the hypothesis space is enormous.



- **Boundary sets** $S_1, \ldots, S_n$ and $G_1, \ldots, G_m$ represent the hypothesis space.
- Problem: the version space may collapse.

Belief Network Learning Problems

	Structure	
	Known	Unknown
Observable variables	Only CPTs are learned	Heuristic search of the structure
Hidden variables	Neural networks	No good/general algorithms