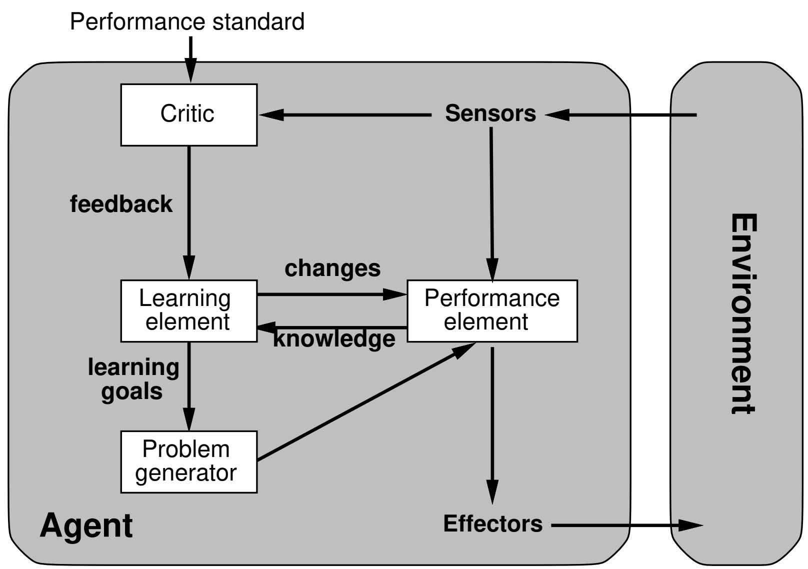
Learning Agents

- Agents that can learn from experience as a result of interacting with the environment (through both perception and action).
- Percepts are used not only to determine the current action (as in agents that do not learn), but also to improve the future performance in making rational decisions. (For example, human chess players learn from experience to improve their skills in playing better chess games in the future.)
- Motivations:
  - This can save the designer's time and effort to pre-program everything into the agent (even if it can be done).
  - This allows the agent to adapt to changing environments – continuous learning rather than one-shot learning.
  - No (natural or artificial) systems can be said to be truly intelligent without the ability to learn.
- Two aspects of learning:
  - Knowledge acquisition (e.g., learning AI from COMP221)
  - Skill acquisition (e.g., learning to ride a bicycle)

A General Model of Learning Agents



- Performance element: selects actions to take based on percepts and learned experiences.
- Learning element: makes improvements.
- Critic: tells the learning element how well the agent is doing.
- Problem generator: suggests actions that will lead to new and informative experiences (exploration).
- Environment

Design of Learning Element

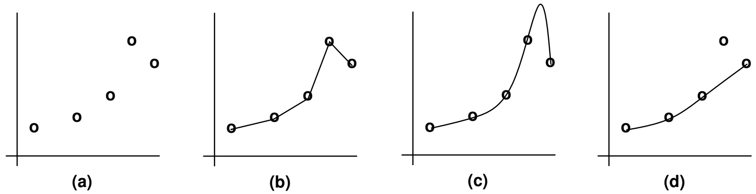
- Elements affecting the design:
  - Performance element:
    - Which *components* to learn (e.g., relationships between states and actions, utilities of states).
    - What *representation* is used for these components (e.g., FOL sentences, belief networks, neural networks).
  - Environment:
    - What *feedback* is available – the richer the information fed back from the environment is, the easier is the learning task (different learning paradigms, to be discussed next, will be resulted).
  - Knowledge base:
    - What *prior knowledge* is available – a learning agent does not have to start from zero.

Design of Learning Element (cont'd)

- Learning paradigms:
  - Inductive vs. deductive learning:
    - Inductive learning:
      From special cases to general rules – generalization.
    - Deductive learning:
      From general rules to special cases – cf. logical reasoning or theorem proving.
  - Supervised vs. reinforcement vs. unsupervised learning:
    - Supervised learning:
      Training examples are input-output pairs with *informative* output (e.g., an examination paper with answers provided by a student is graded by the teacher who also writes down the correct answer for each incorrectly answered question).
    - Reinforcement learning:
      Training examples are input-output pairs with *evaluative* output only (e.g., the teacher only gives an overall grade for the examination paper without providing detailed marks and correct answers for individual questions separately).
    - Unsupervised learning:
      Training examples are input patterns with *no* associated output patterns (e.g., given many Web pages, one learns to group them into subject categories using some similarity measure as reference – the subject categories are not defined in advance but have to be discovered during the process).

Supervised Inductive Learning

- This is the learning paradigm we are interested in here.
- Problem formulation:
  - Given:
    A training set of examples of the form $(x, f(x))$, where x is the input and $f(x)$ is the output (possibly contaminated with noise) of the function f applied to x.
  - Find:
    A hypothesis h that approximates f, in the sense of giving correct (or approximately correct) values of f for the examples in the training set *as well as* other unseen examples.
- Curve-fitting analogy:



- Examples:
  - x is a character image; $f(x)$ is the correct character class of x; $h(x)$ is the hypothesized character class of x returned by the character recognition system. *The learning problem is to learn to recognize characters correctly.*
  - x is a patient record; $f(x)$ is the actual disease the patient has; $h(x)$ is the diagnosis reported by the doctor. *The learning problem is to learn to perform medical diagnosis accurately.*

A More Detailed Example

Attributes				Class
Outlook	Temperature	Humidity	Windy	
sunny	hot	high	false	N
sunny	hot	high	true	N
overcast	hot	high	false	P
rain	mild	high	false	P
rain	cool	normal	false	P
rain	cool	normal	true	N
overcast	cool	normal	true	P
sunny	mild	high	false	N
sunny	cool	normal	false	P
rain	mild	normal	false	P
sunny	mild	normal	true	P
overcast	mild	high	true	P
overcast	hot	normal	false	P
rain	mild	high	true	N

- Each row is a 4-dimensional input pattern and a corresponding 1-dimensional output pattern.
- The P class corresponds to "Play" and the N class corresponds to "Not Play" under the weather condition represented by the input pattern.

Data Mining

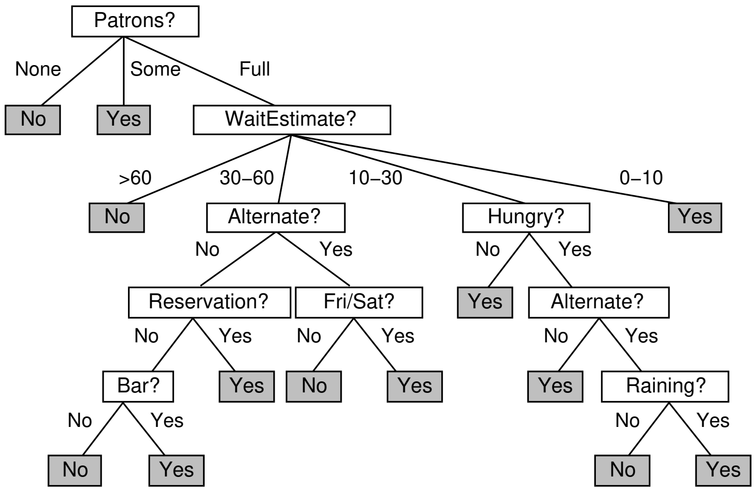
- Application of machine learning techniques to large databases:
  - Extracting significant, previously unknown information from large databases.
  - Presenting the extracted information in user-friendly form for review and examination.
  - Using the extracted information to make decisions.
- Also called *knowledge discovery in databases* (KDD).
- An emerging area (combining techniques in machine learning, pattern recognition, neural networks, statistical data analysis, database management, etc.) with strong implications on practical applications.

Decision Trees as Performance Elements

- Simple decision trees represent Boolean functions. Any Boolean function can be expressed as a decision tree.
- Decision trees are also called *tree classifiers*.
- Terminal nodes:
  - Each terminal node indicates a class.
- Nonterminal nodes:
  - Decision nodes
  - Each nonterminal node specifies a test on the value of an attribute, with one subtree for each possible outcome of the test.
- The path from the root node to each terminal node can be expressed as an implicative sentence (if-then rule), cf. Horn clause.
- In general, decision trees are not as expressive as FOL sentences.

An Example

- Decision to make:
  - Whether to wait for a table at a restaurant.
- Attributes for describing examples:
  - *Alternate, Bar, Fri/Sat, Hungry, Patrons, Price, Raining, Reservation, Type, WaitEstimate*
- A decision tree (smaller, more compact representations exist):



- Example of an implicative sentence corresponding to a path:

$$\forall x \quad \mathit{Patrons}(x, \mathit{Full}) \land \mathit{WaitEstimate}(x, 10{-}30) \land \mathit{Hungry}(x, \mathit{No}) \Rightarrow \mathit{WillWait}(x)$$

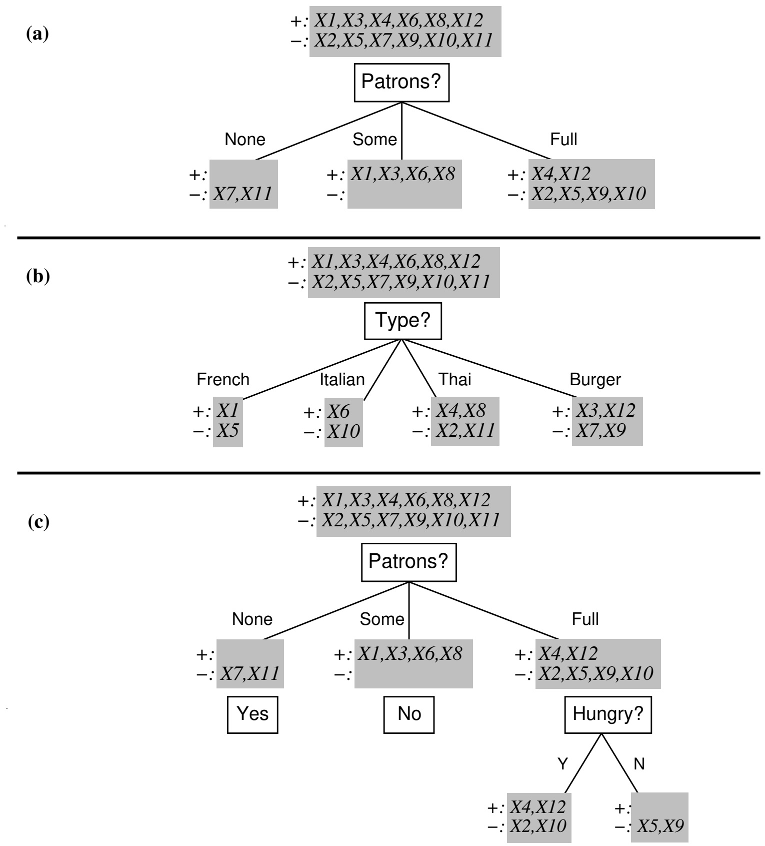
Decision Tree Learning

- The objective of learning is not to just *remember* the training examples. (Otherwise building a database is sufficient.) It is more important to classify correctly cases that were not used for training, i.e., generation to unseen cases.
- For many real-world domains where regularities exist in the examples, a small tree (i.e. concise representation) can usually generalize to unseen cases better than a large tree.
- Training set:

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*

Decision Tree Learning (cont'd)

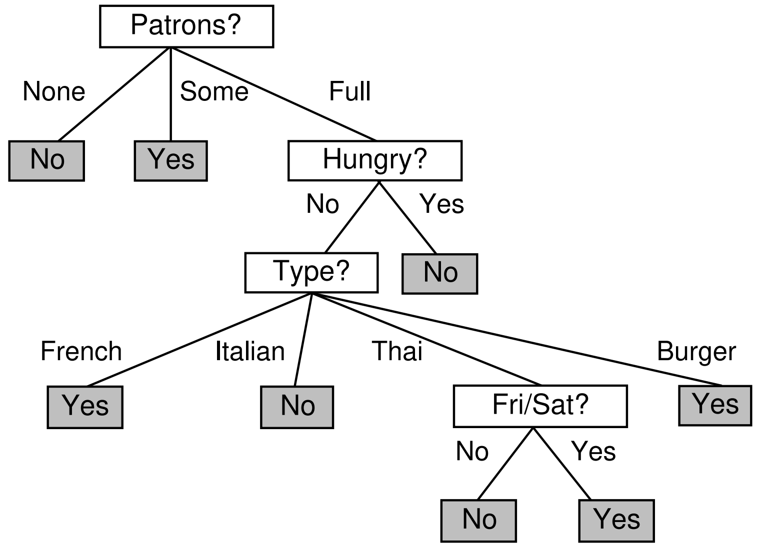
- Tree construction procedure:
  - The basic idea is to test the most "important" attribute first. (An attribute is considered "important" if it makes a large difference to the classification of an example.)



- The issue of quantifying attribute "importance" will be addressed later.

Decision Tree Learning (cont'd)

- Final tree constructed:



Decision Tree Learning Algorithm

```
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
```


Decision Tree Learning Algorithm (cont'd)

- The learning algorithm is recursively defined.
- At each recursive step, there are four possible outcomes:
  - No examples left:
    - Terminal node with default class returned.
  - All remaining examples belong to the same class:
    - Terminal node with class returned.
  - Both positive and negative examples left, but with no remaining attributes:
    - Two possible reasons:
      - Noisy data
      - Inadequate set of attributes
    - Terminal node with majority class returned.
  - Both positive and negative examples left, and with some remaining attributes:
    - Nonterminal node with algorithm applied recursively.