Learning – Outline of What we will Cover

- Background
- Learning agents
- Inductive learning
- Decision tree learning
- Brief Overview of Neural Networks

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

Machine Learning General Overview

- Supervised learning: given sample input-output pairs, learn a function to predict outputs given valid inputs
- Unsupervised learning: learn patterns in the input with no feedback of any kind
- Reinforcement learning: learn from occasional rewards or punishments
- Evolutionary learning: adaptive learning using assessment

Fields related to machine learning

- Pattern classification
- Probabilistic learning
- Neural networks
- Machine learning
- Computational learning
- …

Learning

- Rote learning
  - Learning foreign words (and their translations)
  - Learning telephone numbers, bank accounts, ..
- Inductive learning
  - Learning from examples
  - Learning by observing and discovering

Early work on automatic learning

- Rosenblatt (1957): Perceptrons
- Samuel (1963): Rote learning
- Winston (1970): Concept learning
- Mitchell (1977): Version spaces

(See Russel and Norvig for more details)

Rosenblatt's perceptrons

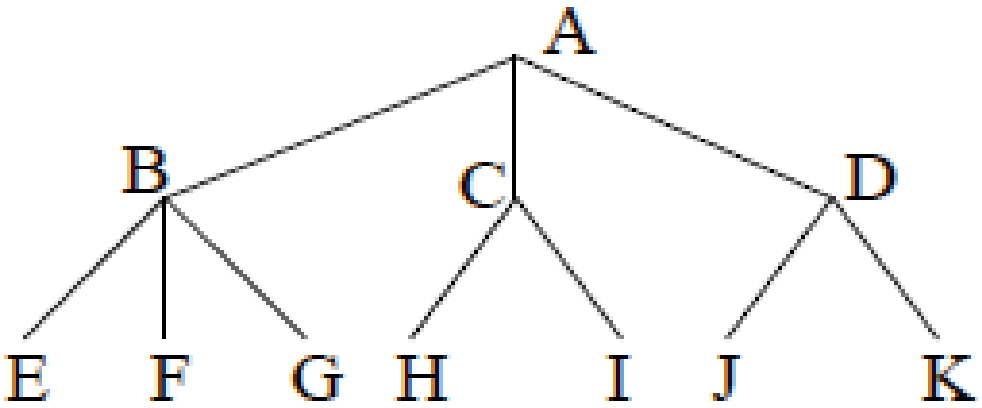
- Early work on neural networks
- In contrast to the other early work, Rosenblatt did not used explicit descriptions of concepts

- Ultimately inspired much later work, e.g.,
  - Multilayer perceptrons
  - Radial Basis Function networks
  - Hopfield networks
  - ….

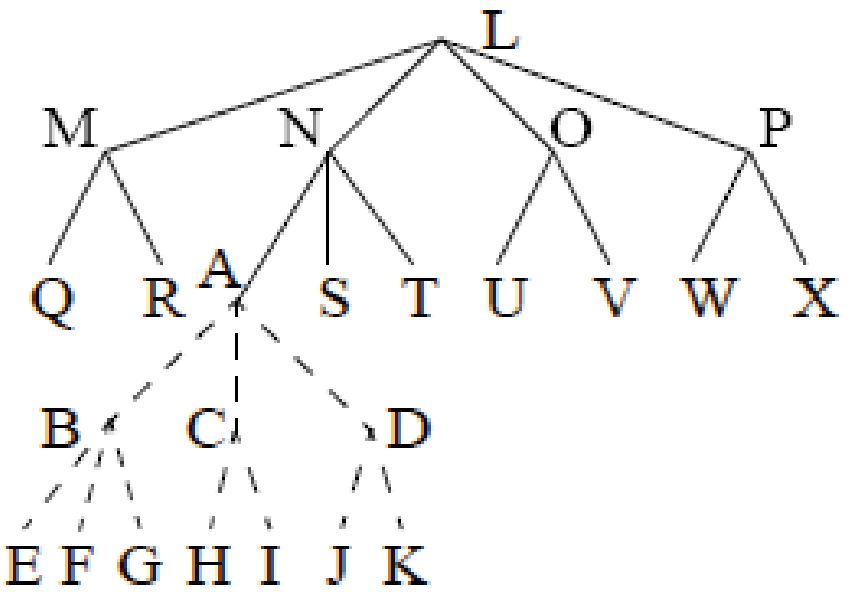
Samuel's rote learning

- The game of checkers
- Each board position is stored along with its evaluation
- Using search algorithms to determine evaluation values of board positions

Example



Minimax(A) = 10
→ Store (A,10)



Upon encountering A during search: Retrieve evaluation of A (identical to searching the subtree with root A)

Winston's concept learning

- Domain: blocks world
- Supply the program with positive and negative instances (near misses) to learn a description of a class
- Positive instances: belong to class C
- Negative instances: do not belong to class C
- Apply inductive-learning heuristics such as "require link" and "forbid link"

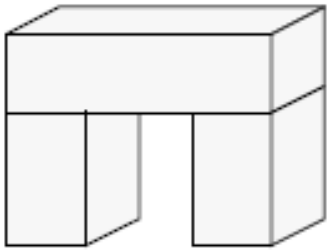
Winston's learning algorithm

1. Create a structural description of an instance of a class
2. Collect descriptions of other instances and try to generalize the description by retaining the relevant parts
3. Collect descriptions of "near misses" and use them to restrict the generalized description

Example

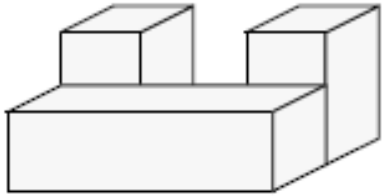
- The class ARCH
- The program is presented with a few instances

1



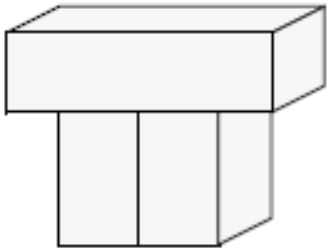
Example

2



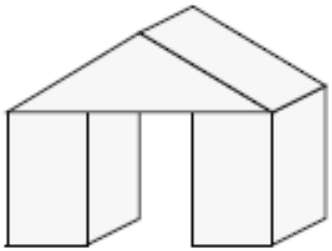
Near-Miss

3



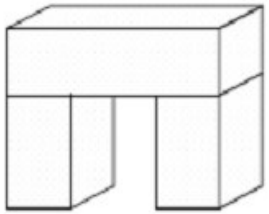
Near-Miss

4

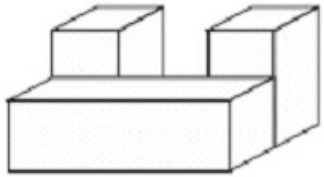


Example

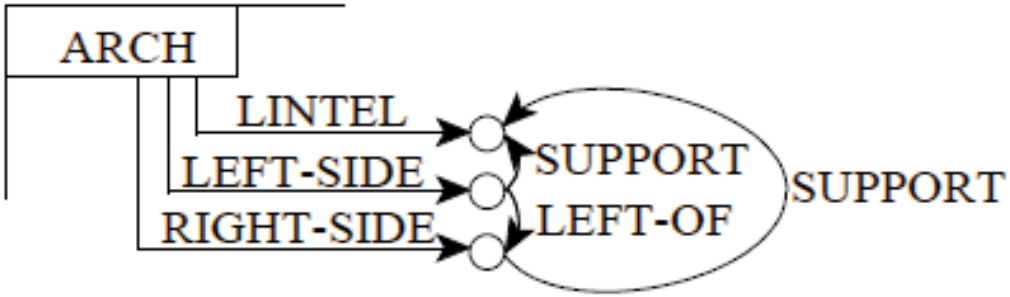
“Require-link” heuristic

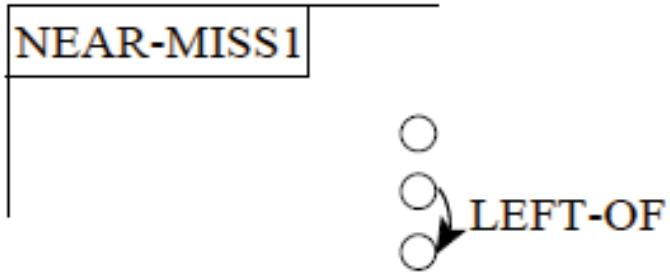


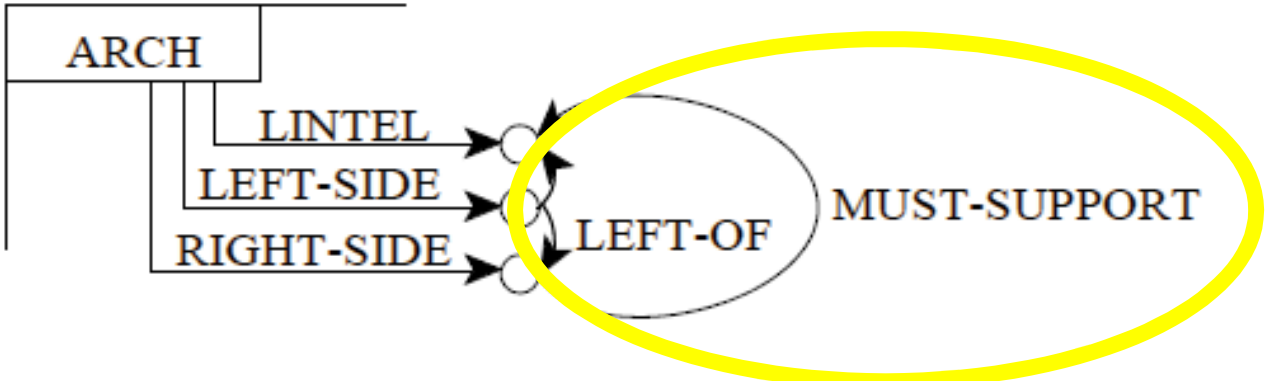
Example



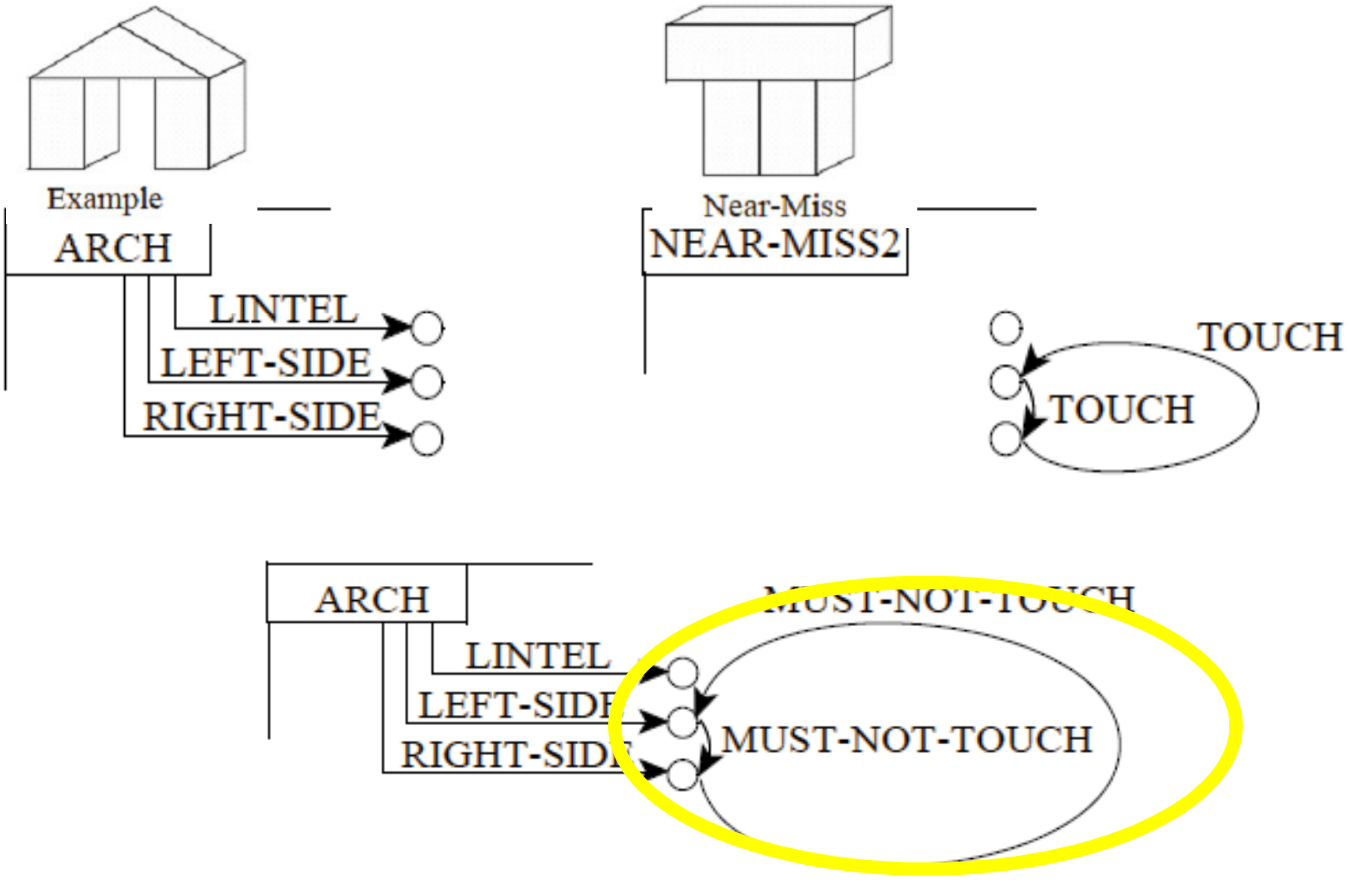
Near-Miss







“Forbid-link” heuristic



Mitchell's Version Spaces

- Another way to learn descriptions of concepts

- Goal: to generate a description that covers all positive instances and that rejects all negative instances

(See also Mitchell's *Machine Learning*, chapter 2)

Example

- Concept: *car*
- Descriptions in terms of frames

Car023	
origin:	*Japan*
manufacturer:	*Honda*
color:	*Blue*
decade:	*1970*
type:	*Economy*

Representation language (bias)

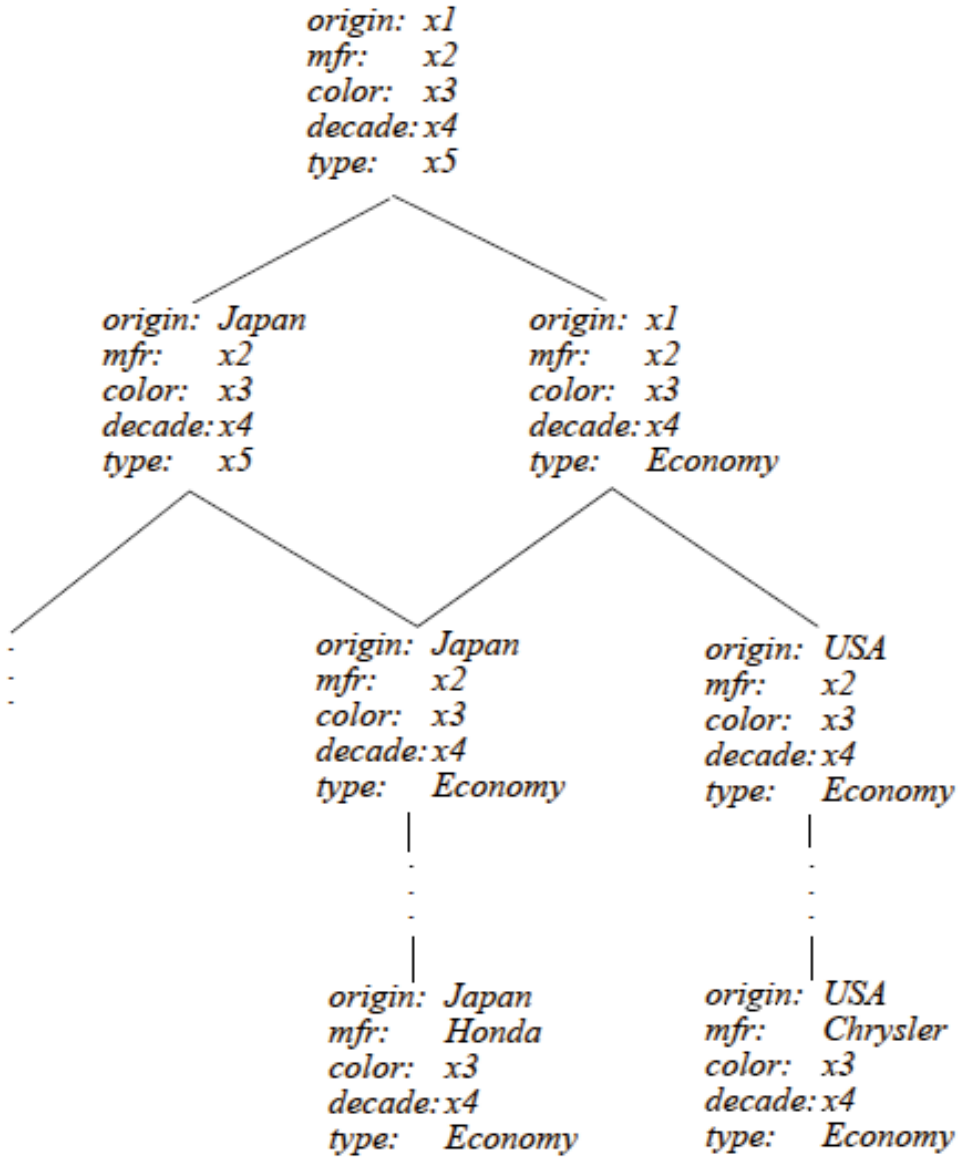
$$
\begin{array}{lll}
origin & \in & \{Japan, USA, Britain, Germany, Italy\} \\
manufacturer & \in & \{Honda, Toyota, Ford, Chrysler, Jaguar, BMW, Fiat\} \\
color & \in & \{Blue, Green, Red, White\} \\
decade & \in & \{1950, 1960, 1970, 1980, 1990, 2000\} \\
type & \in & \{Economy, Luxury, Sports\}
\end{array}
$$

Japanese economy car

$$
\begin{array}{ll}
origin: & Japan \\
manufacturer: & x1 \\
color: & x2 \\
decade: & x3 \\
type: & Economy
\end{array}
$$

Partially ordered concept space

- The representation language defines a partially ordered representation of concepts
- The illustration shows part of this space



Two subsets: G and S

- G is the subset of the most general description that covers all instances encountered

- S is the subset of the most specific descriptions of all instances encountered

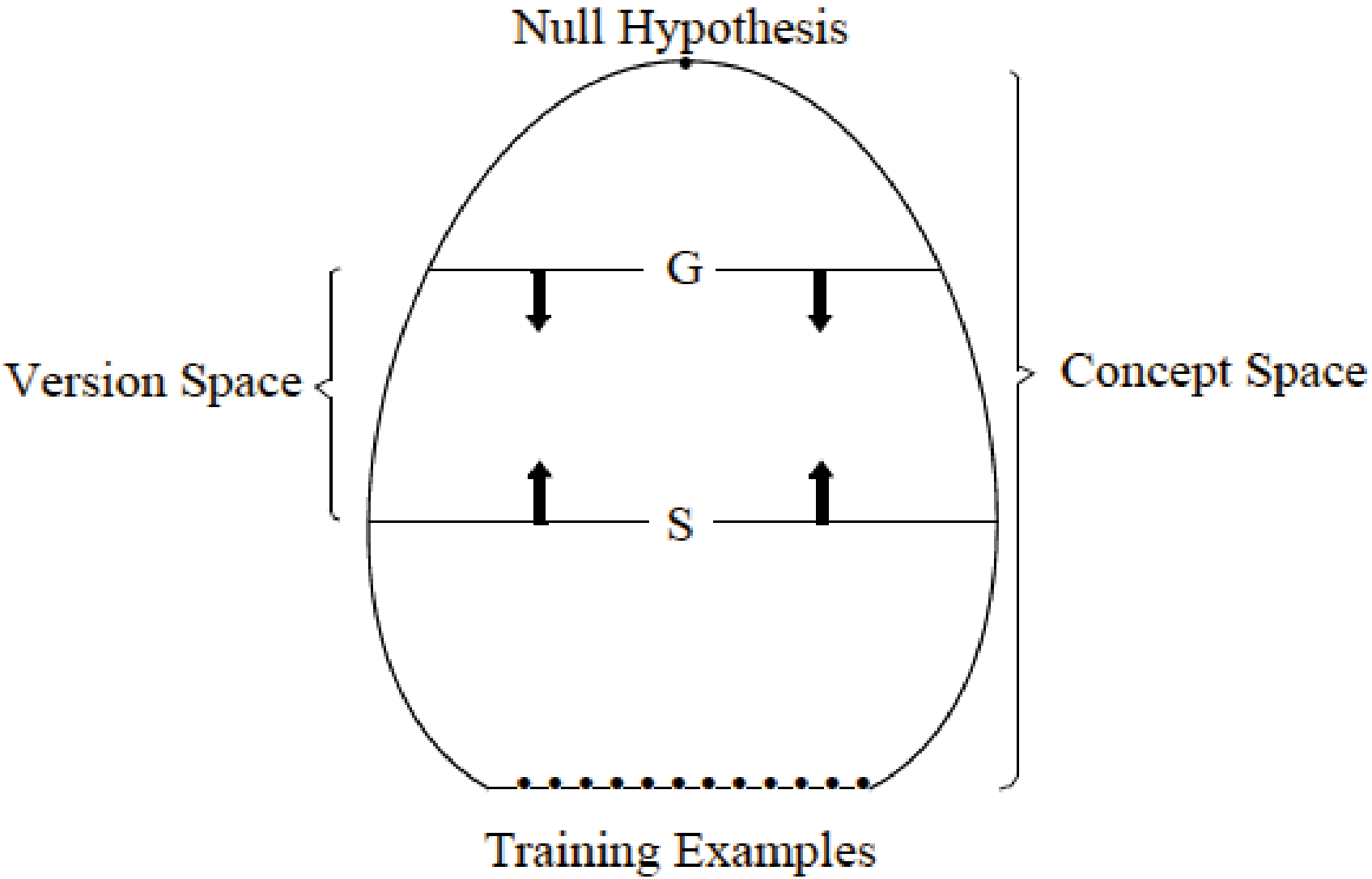
Learning

- Each time a negative instance is encountered that is an element of G, the subset G is made more specific to exclude the negative instance

- Each time a positive instance is encountered that is not yet an element of S, the subset S is made more general to include the positive instance

(These steps are executed by the *candidate elimination algorithm*)

The version space



"Modern" learning algorithms

- Concept learning algorithms
  - Version spaces (combined with Support Vector Machines),
- Statistical algorithms
  - Gaussian mixtures, Naïve Bayes, ...
- Neural networks
  - multilayer perceptron, Kohonen maps, ...
- Advanced algorithms
  - Support Vector Machines, kernel approaches
- ...

Overview of learning algorithms

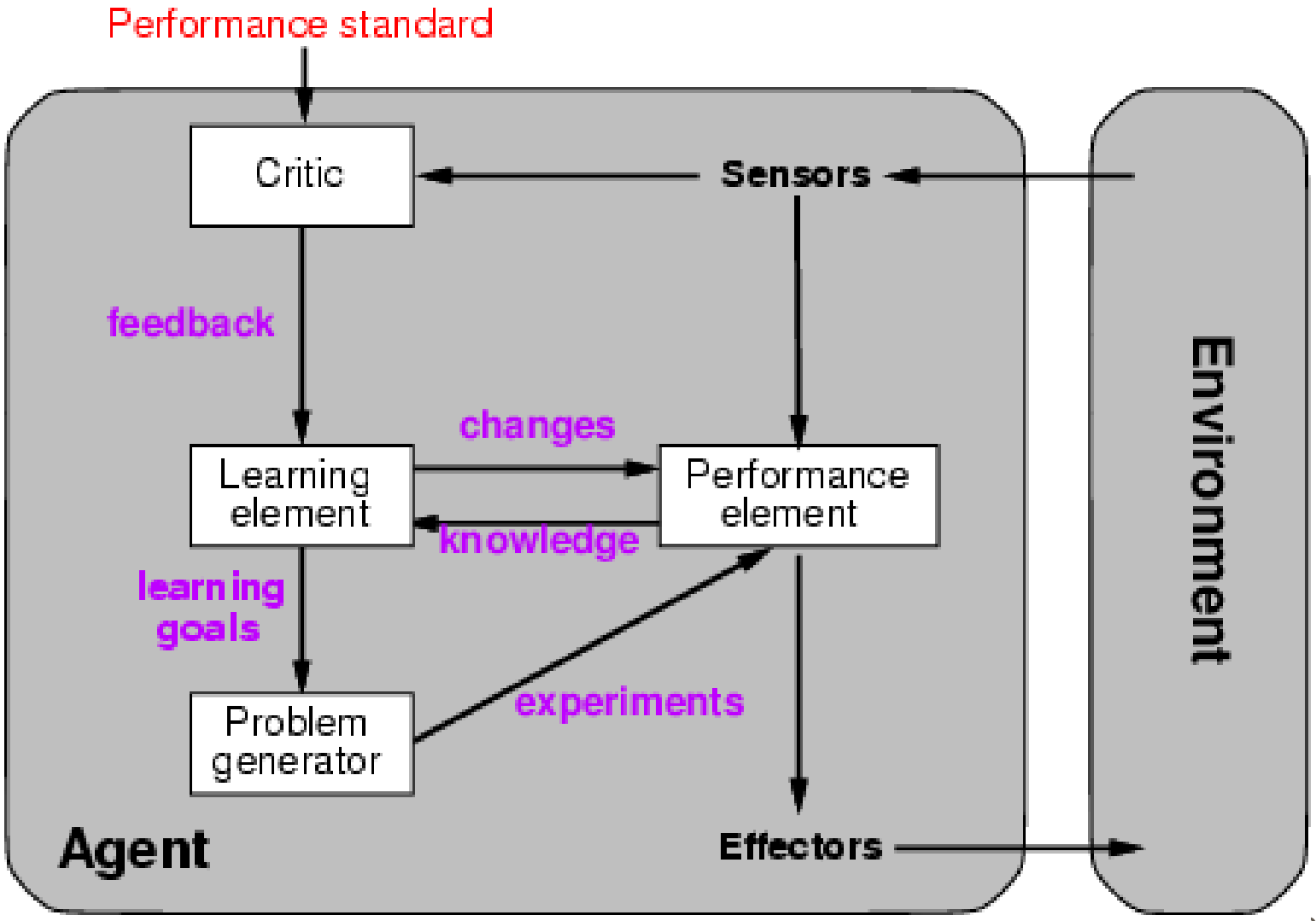
- We will examine several learning algorithms
- The basics of each algorithm will be explained
- Then, the algorithm is demonstrated on a standard data set
  - In particular, the effects of parameters will be assessed

Learning element

- Design of a learning element is affected by
  - Which components of the performance element are to be learned
  - What feedback is available to learn these components
  - What representation is used for the components

- Type of feedback:
  - **Supervised learning**: correct answers for each example
  - **Unsupervised learning**: correct answers not given
  - **Reinforcement learning**: occasional reward for right answer
  - **Evolutionary learning**: improvement through assessment

Learning agents



Inductive learning

- Simplest form: learn a function from examples

f is the **target function**

An **example** is a pair (*x*, *f(x)*)

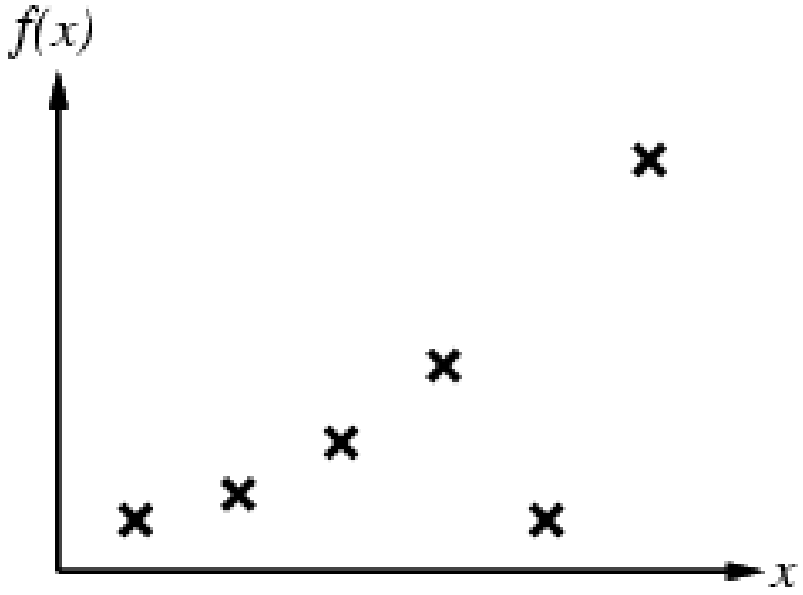
Problem: find a **hypothesis** *h*
such that $h \approx f$
given a **training set** of examples

This is a highly simplified model of real learning:

- **Ignores prior knowledge**
- **Assumes examples are given**

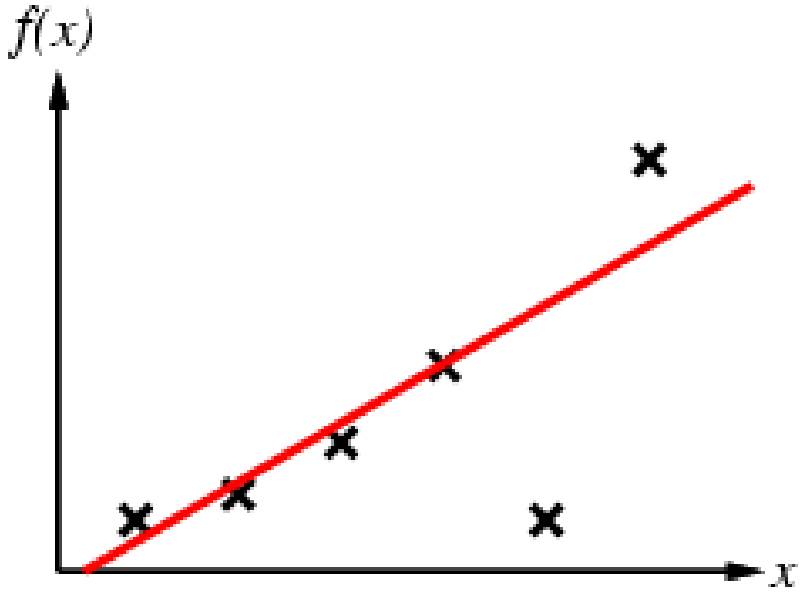
Inductive learning method

- 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:



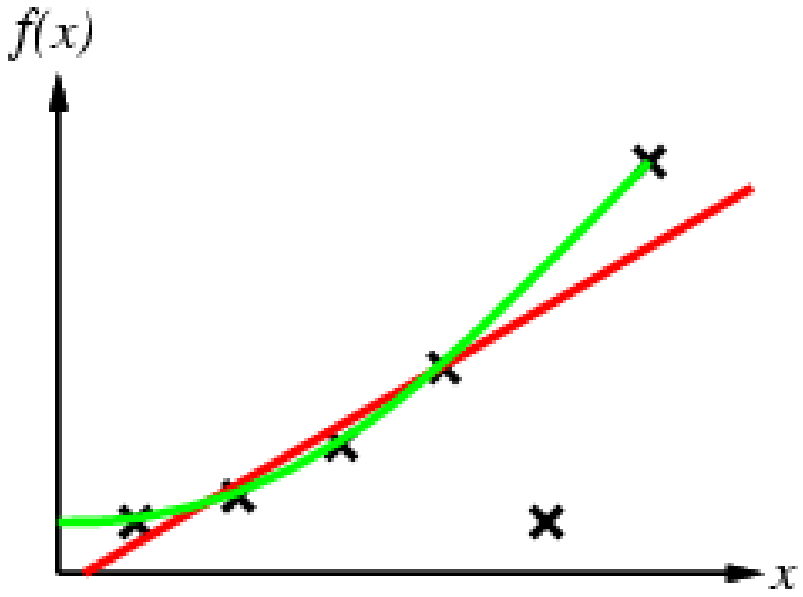
Inductive learning method

- 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:



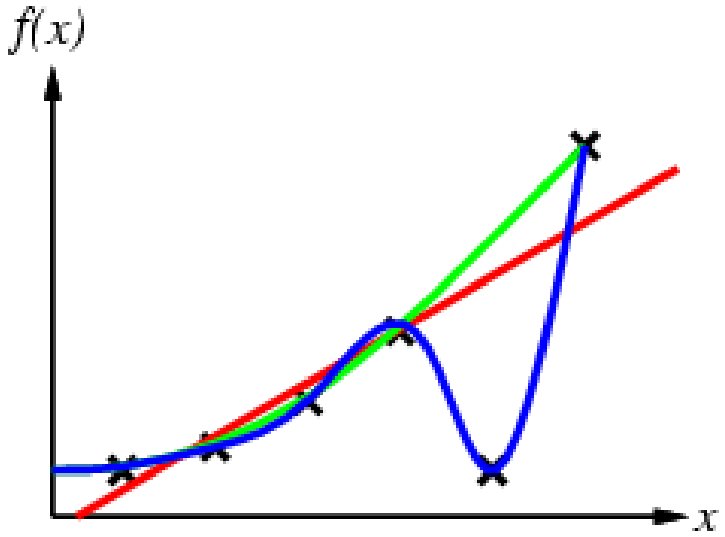
Inductive learning method

- 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:



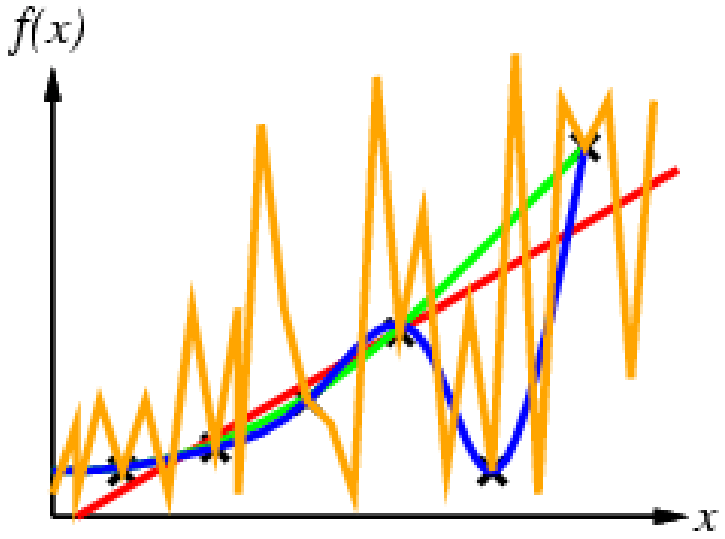
Inductive learning method

- 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:



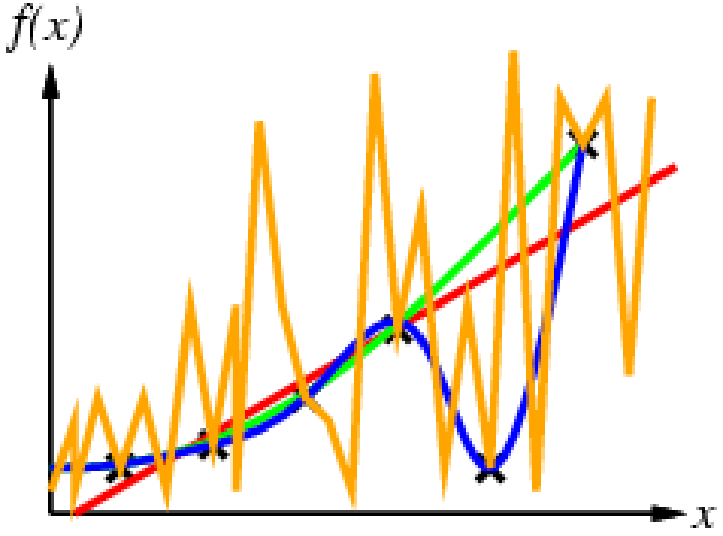
Inductive learning method

- 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:



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: prefer the simplest hypothesis that is **consistent** with data

Learning decision trees

Problem: decide whether to wait for a table at a restaurant, based on the following attributes:

1. Alternative: is there an alternative restaurant nearby?
2. Bar: is there a comfortable bar area to wait in?
3. Fri/Sat: is today Friday or Saturday?
4. Hungry: are we hungry?
5. Patrons: number of people in the restaurant (None, Some, Full)
6. Price: price range ($, $$, $$$)
7. Raining: is it raining outside?
8. Reservation: have we made a reservation?
9. Type: kind of restaurant (French, Italian, Thai, Burger)
10. WaitEstimate: estimated waiting time (0-10, 10-30, 30-60, >60)

Attribute-based representations

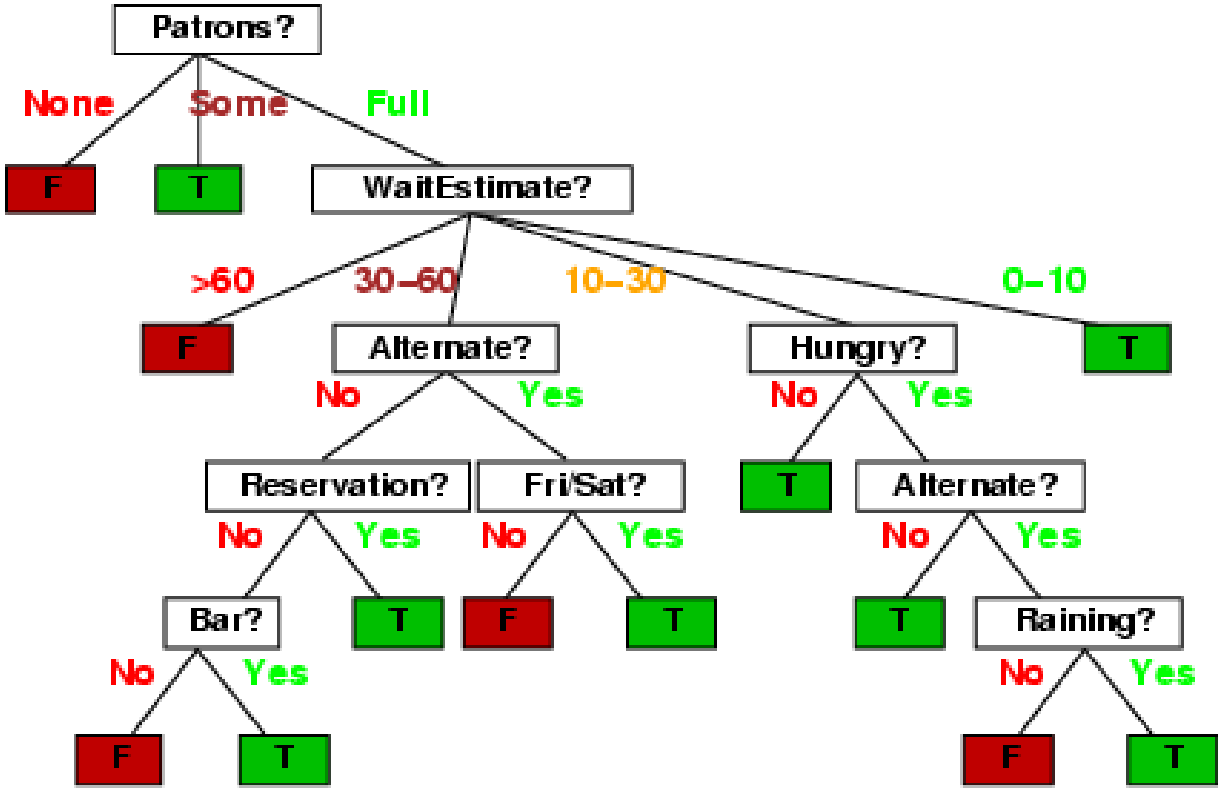
- Examples described by **attribute values** (Boolean, discrete, continuous)
- E.g., situations where I will/won't wait for a table:

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

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

Decision trees

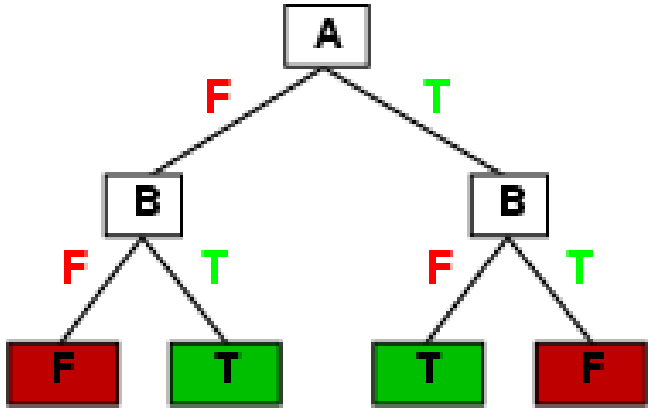
- One possible representation for hypotheses
- E.g., here is the "true" tree for deciding whether to wait:



Expressiveness

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

A	B	A xor B
F	F	F
F	T	T
T	F	T
T	T	F



- Trivially, there is a consistent decision tree for any training set with one path to leaf for each example (unless *f* is non-deterministic in *x*) but it probably will not generalize to new examples

- Prefer to find more **compact** decision 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

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 with training set
    $\Rightarrow$ may get worse predictions

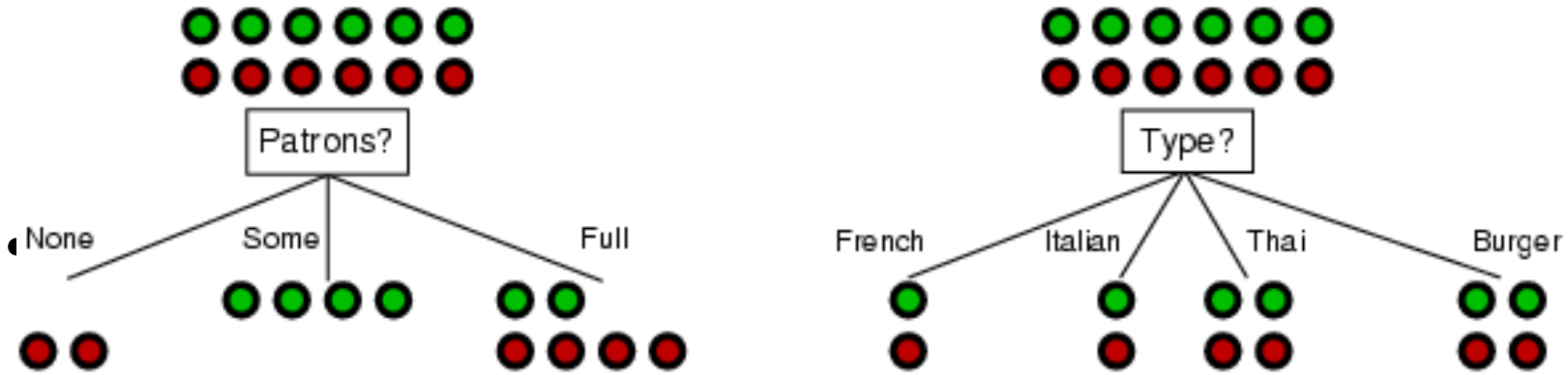
Decision tree learning

- Aim: find a small tree consistent with the training examples
- Idea: (recursively) 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 ← {elements of examples 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"



Using information theory

- To implement `Choose-Attribute` in the DTL algorithm
- Information Content (Entropy):

$$I(P(v_1), \ldots, P(v_n)) = \Sigma_{i=1} -P(v_i) \log_2 P(v_i)$$

- For a training set containing *p* positive examples and *n* negative examples:

$$I(\frac{p}{p+n}, \frac{n}{p+n}) = -\frac{p}{p+n}\log_2\frac{p}{p+n} - \frac{n}{p+n}\log_2\frac{n}{p+n}$$

Information gain

- A chosen attribute A divides the training set E into subsets E_1, … , E_v according to their values for A, where A has v distinct values.

$$remainder(A) = \sum_{i=1}^{v} \frac{p_i + n_i}{p + n} I\left(\frac{p_i}{p_i + n_i}, \frac{n_i}{p_i + n_i}\right)$$

- Information Gain (IG) or reduction in entropy from the attribute test:

$$IG(A) = I\left(\frac{p}{p+n}, \frac{n}{p+n}\right) - remainder(A)$$

- Choose the attribute with the largest IG

Information gain

For the training set, $p = n = 6$, $I(6/12, 6/12) = 1$ bit

Consider the attributes *Patrons* and *Type* (and others too):

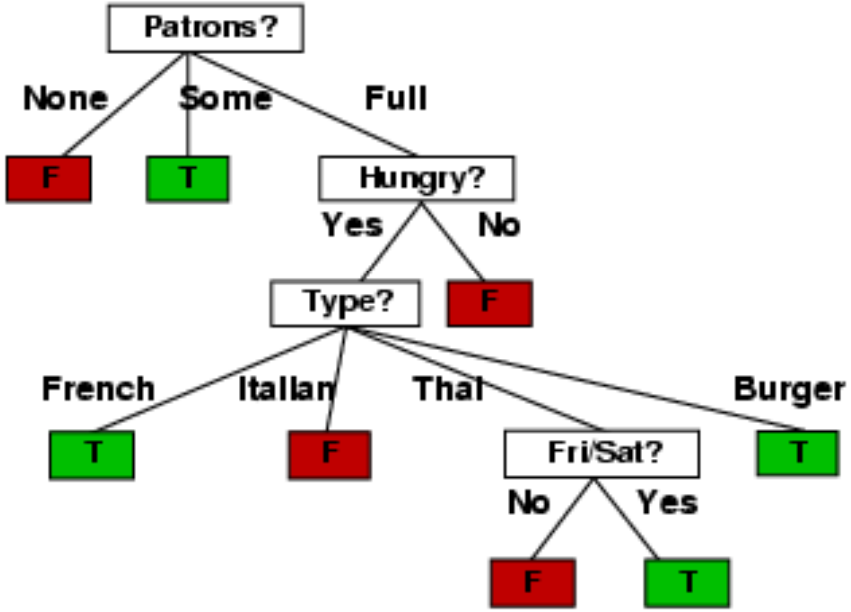
$$IG(Patrons) = 1 - [\frac{2}{12} I(0,1) + \frac{4}{12} I(1,0) + \frac{6}{12} I(\frac{2}{6}, \frac{4}{6})] = .0541 \text{ bits}$$

$$IG(Type) = 1 - [\frac{2}{12} I(\frac{1}{2}, \frac{1}{2}) + \frac{2}{12} I(\frac{1}{2}, \frac{1}{2}) + \frac{4}{12} I(\frac{2}{4}, \frac{2}{4}) + \frac{4}{12} I(\frac{2}{4}, \frac{2}{4})] = 0 \text{ bits}$$

Patrons has the highest IG of all attributes and so is chosen by the DTL algorithm as the root

Example contd.

- Decision tree learned from the 12 examples:

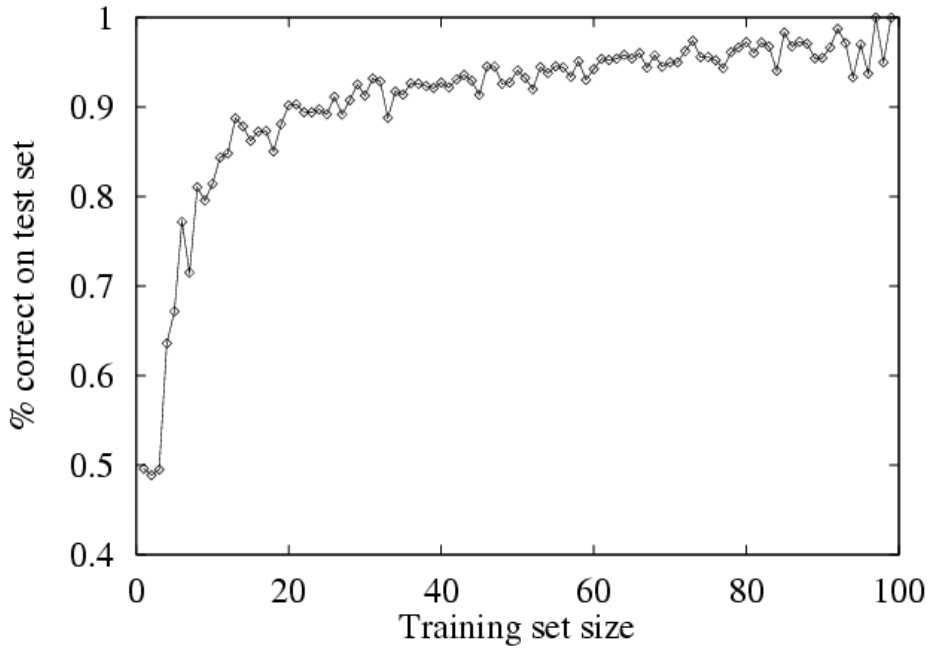


- Substantially simpler than "true" tree---a more complex hypothesis isn't justified by small amount of data

Performance measurement

- How do we know that $h \approx f$?
  1. Use theorems of computational/statistical learning theory
  2. Try h on a new **test set** of examples
     (use **same** distribution over example space as training set)

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



Summary

- Learning needed for unknown environments, lazy designers
- Learning agent = performance element + learning element
- For supervised learning, the aim is to find a simple hypothesis approximately consistent with training examples
- Decision tree learning using information gain
- Learning performance = prediction accuracy measured on test set
- Vigorous area of work and research in AI