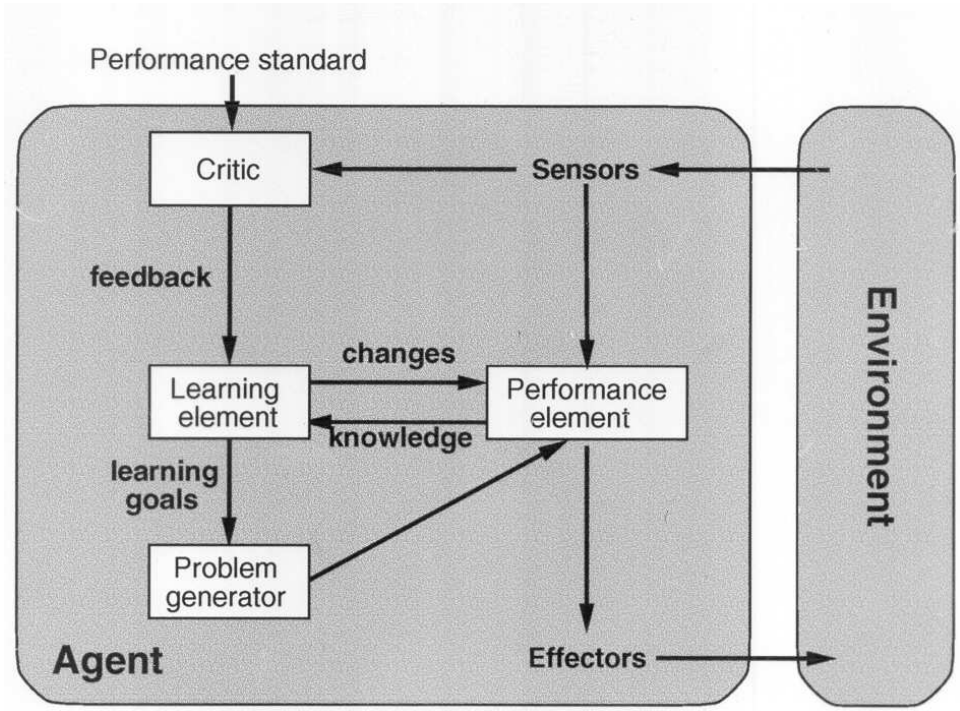
Machine Learning

Tuomas Sandholm
Carnegie Mellon University
Computer Science Department

Machine Learning

Knowledge acquisition bottleneck



Knowledge acquisition vs. speedup learning

Recall: Components of the performance element

1. Direct mapping from conditions on the current state to actions
2. Means to infer relevant properties of the world from the percept sequence
3. Info about the way the world evolves
4. Info about the results of possible actions
5. Utility info indicating the desirability of world states
6. Action-value info indicating the desirability of particular actions in particular states
7. Goals that describe classes of states whose achievement maximizes the agent's utility

Representation of components

Available feedback in machine learning

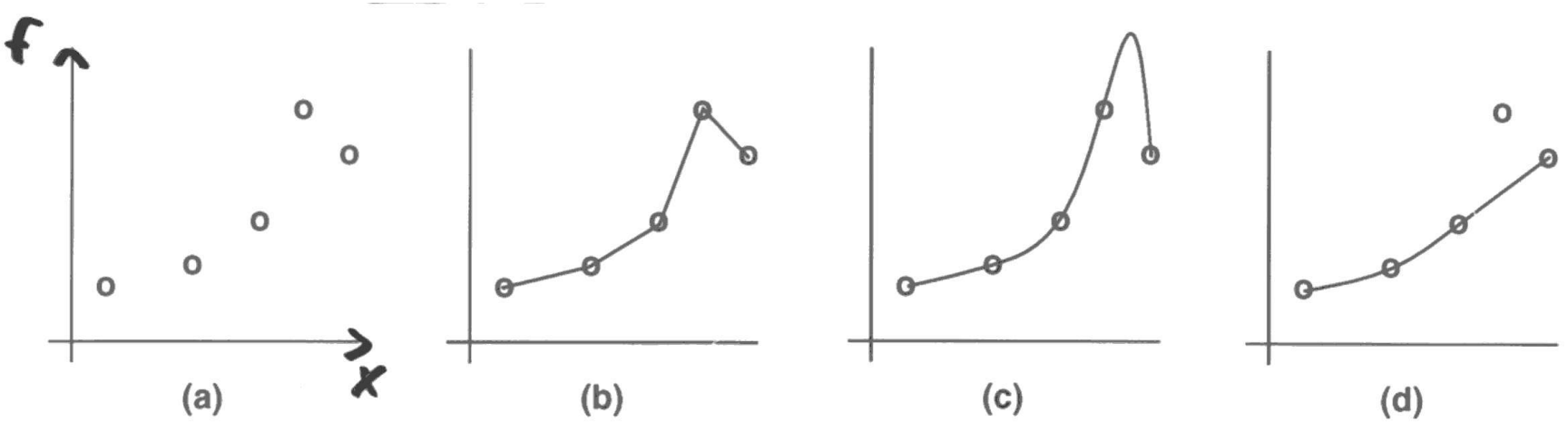
1. Supervised learning
   - Instance: <feature vector, classification>
   - Example: x f(x)
2. Reinforcement learning
   - Instance: <feature vector>
   - Example: x rewards based on performance
3. Unsupervised learning
   - Instance:<feature vector>
   - Example: x

All learning can be seen as learning a function, f(x).

Prior knowledge.

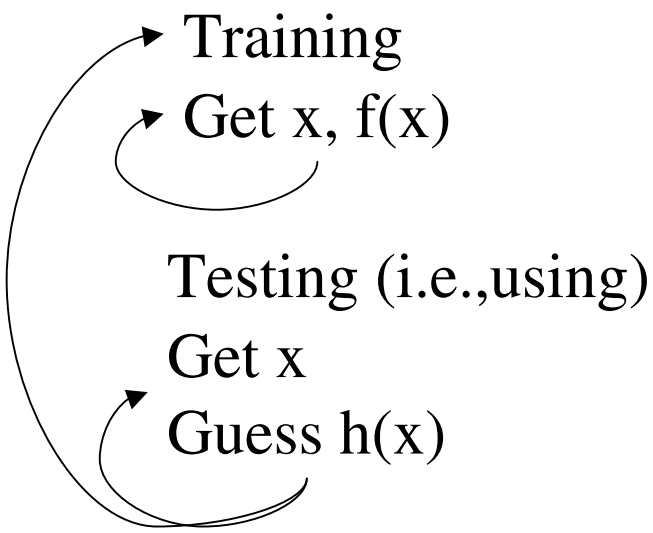
Induction

Given a collection of pairs <x , f(x)>
Return a hypothesis h(x) that approximates f(x).



Bias = preference for one hypothesis over another.
Incremental vs. batch learning.

The cycle in supervised learning



x may or may not have been seen in the training examples

global *examples* ← {}

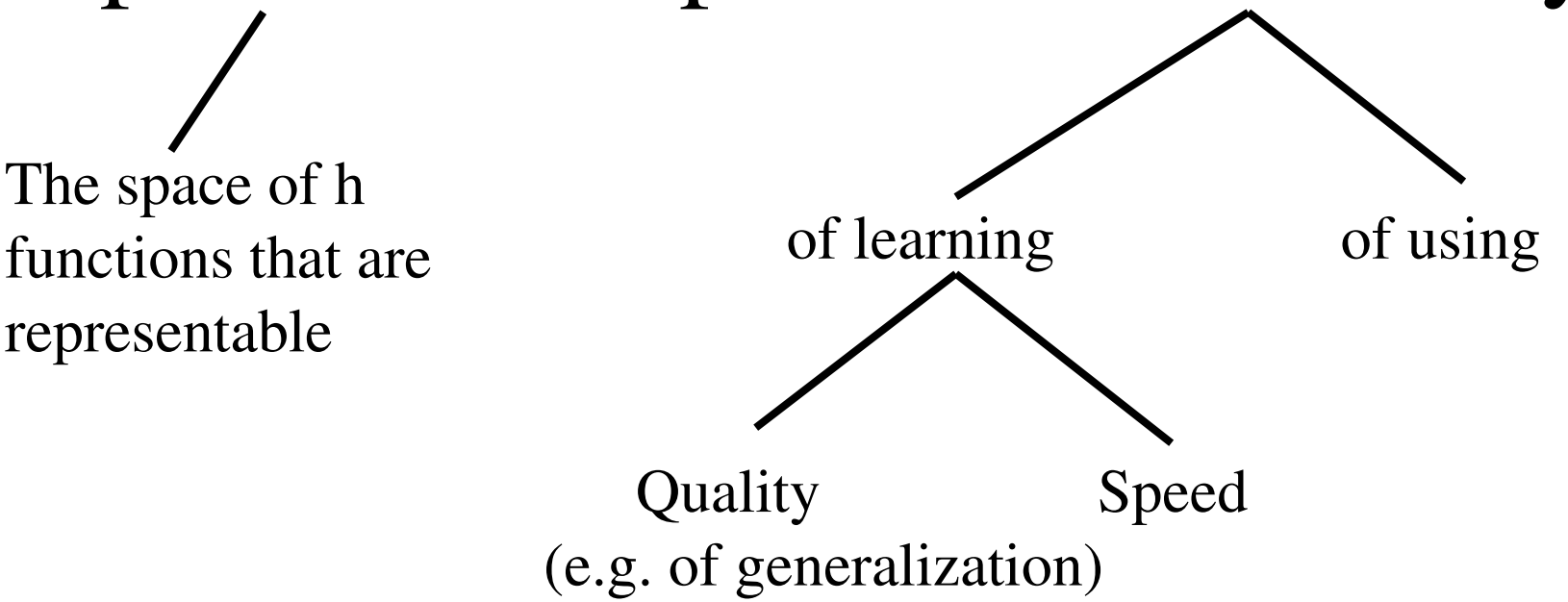
function REFLEX-PERFORMANCE-ELEMENT(*percept*) **returns** an action

  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*)}

Representation power vs. efficiency



Accuracy on

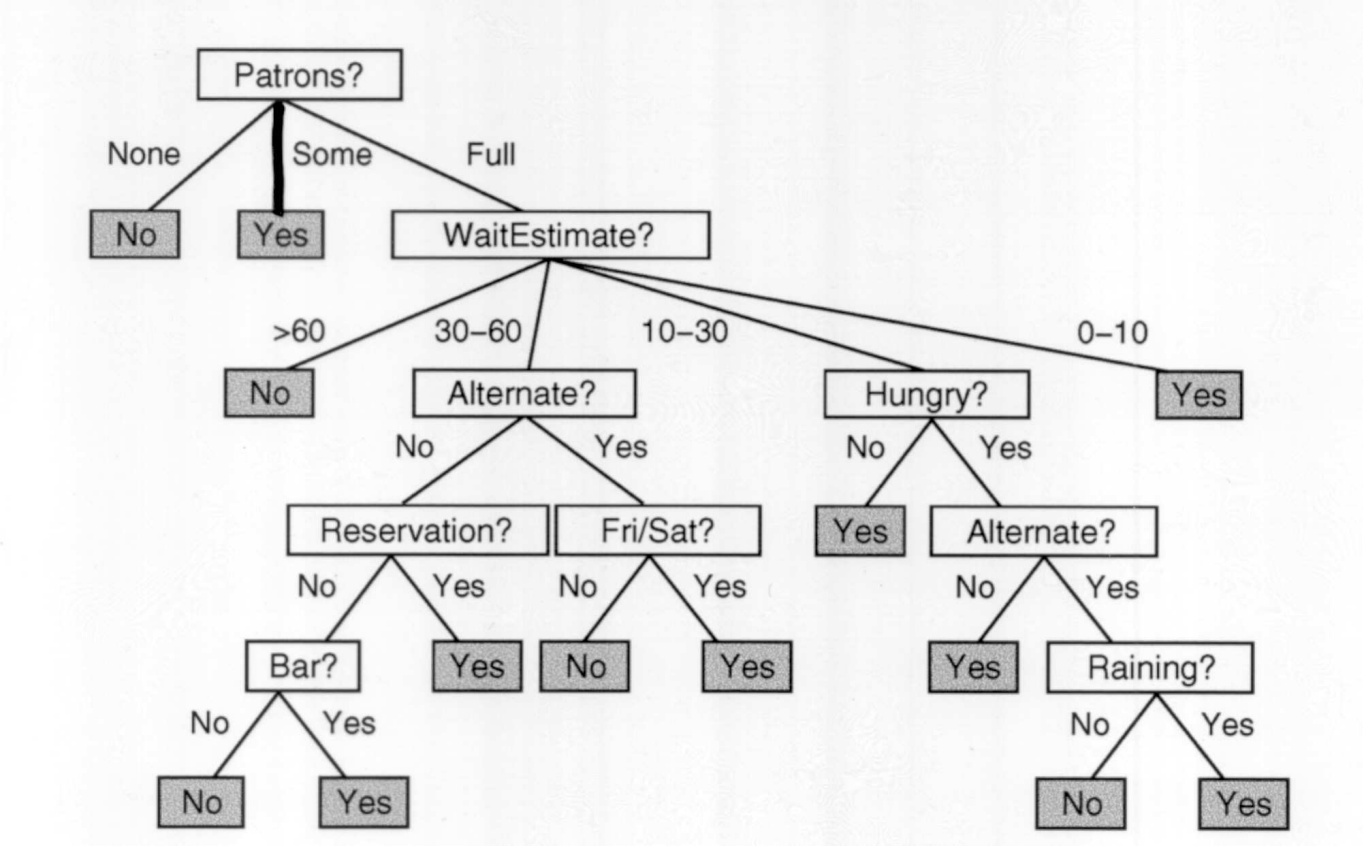
- Training set
- Test set (generalization accuracy)
- combined

We will cover the following supervised learning techniques

- Decision trees
- Instance-based learning
- Learning general logical expressions
- Decision lists
- Neural networks

e.g. want to wait?

Decision Tree



Features: Alternate?, Bar?, Fri/Sat?, Hungry?, Patrons, Price, Raining? Reservations?, Type?, WaitEstimate?

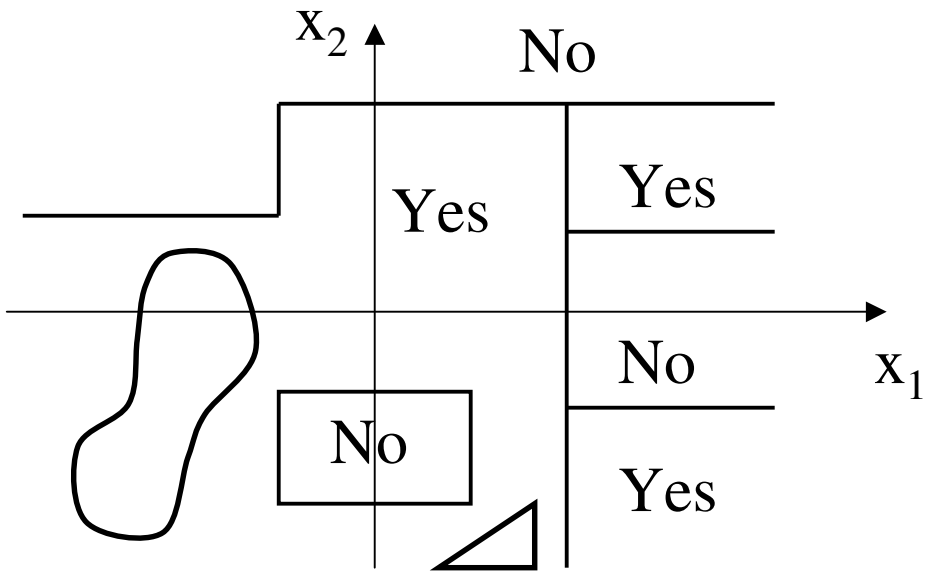
x = list of feature values

E.g. x=(Yes, Yes, No, Yes, Some, $$, Yes, No, Thai, 10-30)

Wait? Yes

Representation power of decision trees

Any Boolean function can be written as a decision tree.



Cannot represent tests that refer to 2 or more objects, e.g.

$\exists r_2$ Nearby(r_2,r) $\wedge$ Price(r,p) $\wedge$ Price(r_2,r_2) $\wedge$ Cheaper (p_2,p)

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*

Trivial solution: one path in the tree for each example

- Bad generalization

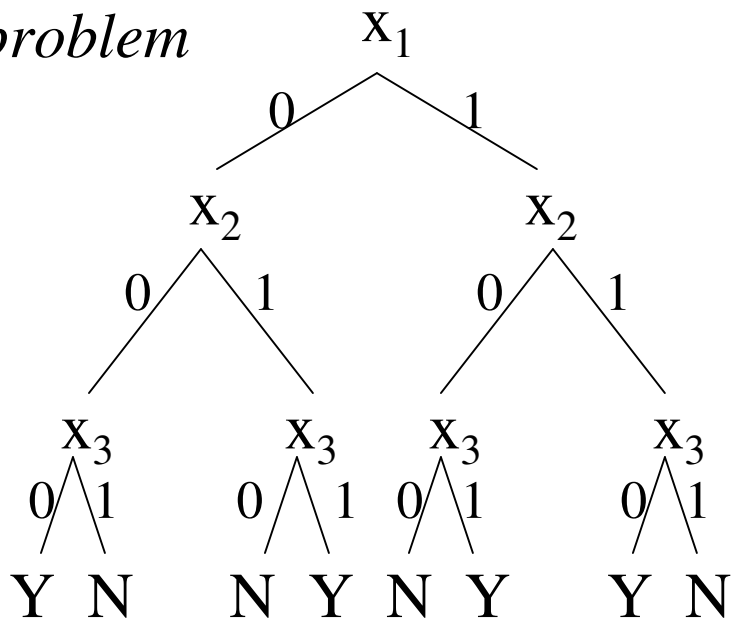
Ockham's razor principle (assumption):

- The most likely hypothesis is the simplest one that is consistent with training examples.

Finding the smallest decision tree that matches training examples is NP-hard.

Representation with decision trees…

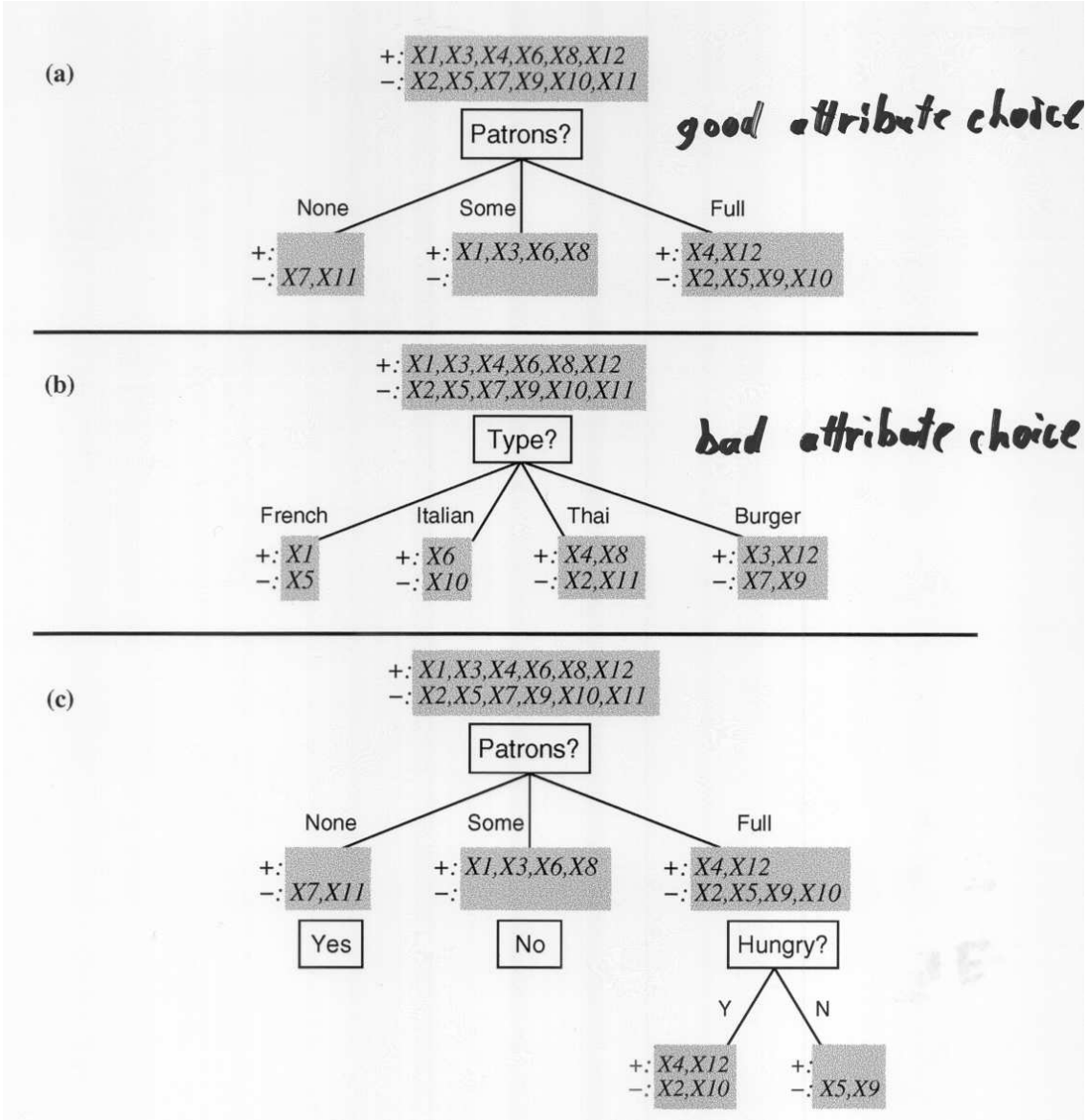
Parity problem



Exponentially large tree.
Cannot be compressed.

n features (aka attributes).
2^n rows in truth table.
Each row can take one of 2 values.
So there are 2^{2^n} Boolean functions of n attributes.

Decision Tree Learning



Decision Tree Learning

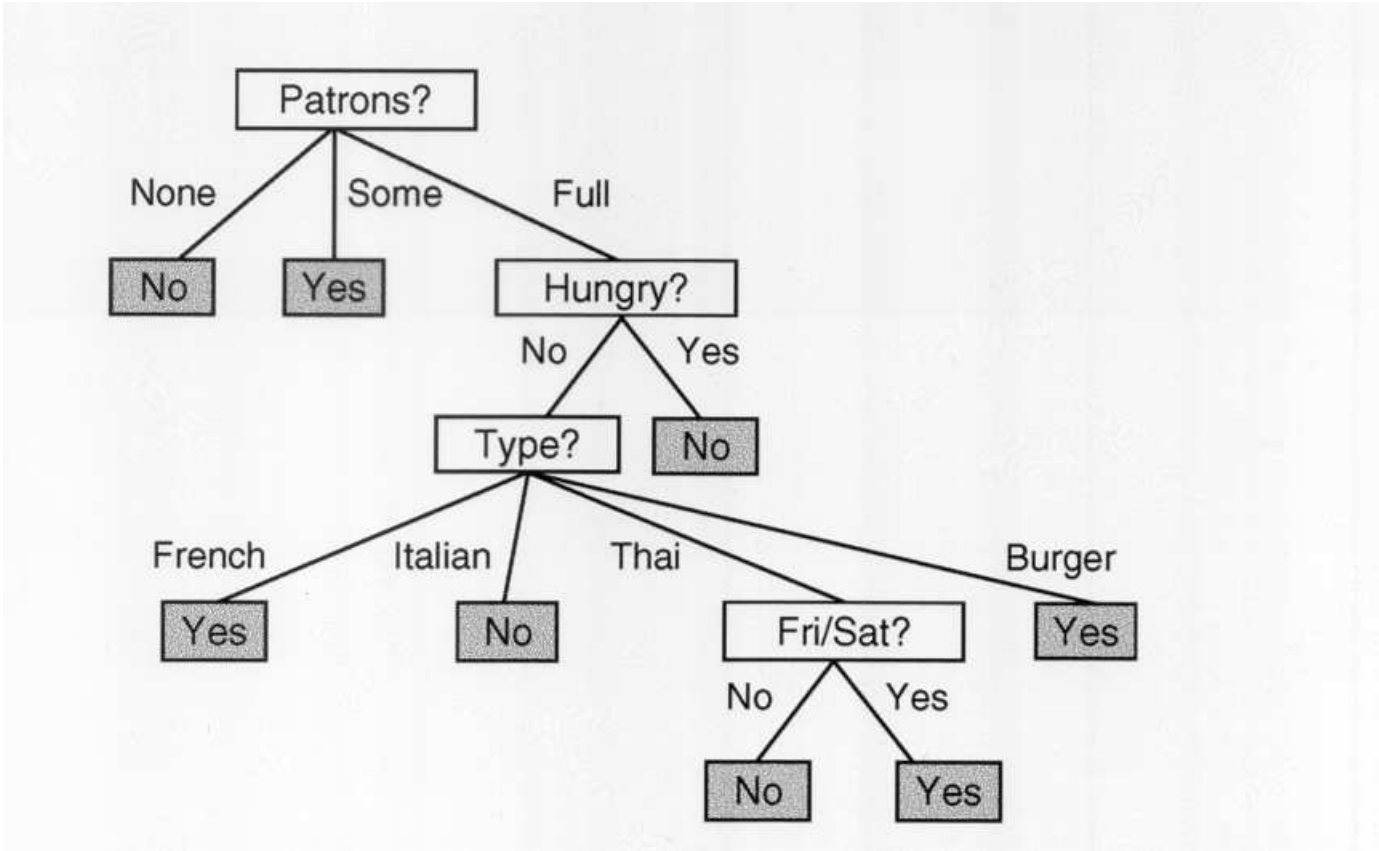
```
function DECISION-TREE-LEARNING(examples, attributes, default) returns a decision tree
  inputs: examples, set of examples
          attributes, set of attributes
          default, default value for the goal predicate

  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
```

= majority classification at node's parent

← noise

Decision Tree Learning



Not the same as original tree even though this was generated from the same examples!

Q: How come?

A: Many hypothesis match the examples.

Using information theory

Bet $1 on the flip of a coin

1. P(heads) = 0.99 bet heads
   E = 0.99 * $1 – 0.01 * $1 = $0.98
   Would never pay more than $0.02 for info.
2. P(heads) = 0.5
   Would be willing to pay up to $1 for info.

Measure info value in bits instead of $: info content is:

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

i.e. average info content weighted by the probability of the events

e.g. fair coin = $I(\frac{1}{2},\frac{1}{2}) = -\frac{1}{2}\log_2\frac{1}{2} - \frac{1}{2}\log_2\frac{1}{2} = 1bit$

loaded coin = $I(\frac{1}{100},\frac{99}{100}) = 0.08\text{bits}$

Choosing decisions tree attributes based on information gain

p = number of positive training examples
n = number of negative training examples
Estimate of how much information is in a correct answer:

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

Any attribute A divides the training set E into subsets $E_1 \ldots E_v$

Remaining info needed after splitting on attribute A

Probability of a random instance having value i for attribute A

Amount of information still needed (in the case where value of A was i)

$$\text{Remainder}(A) = \sum_{i=1}^{v} \overbrace{\frac{p_i+n_i}{p+n}}^{\text{Probability of a random instance having value i for attribute A}} \overbrace{I(\frac{p_i}{p_i+n_i},\frac{n_i}{p_i+n_i})}^{\text{Amount of information still needed (in the case where value of A was i)}}$$

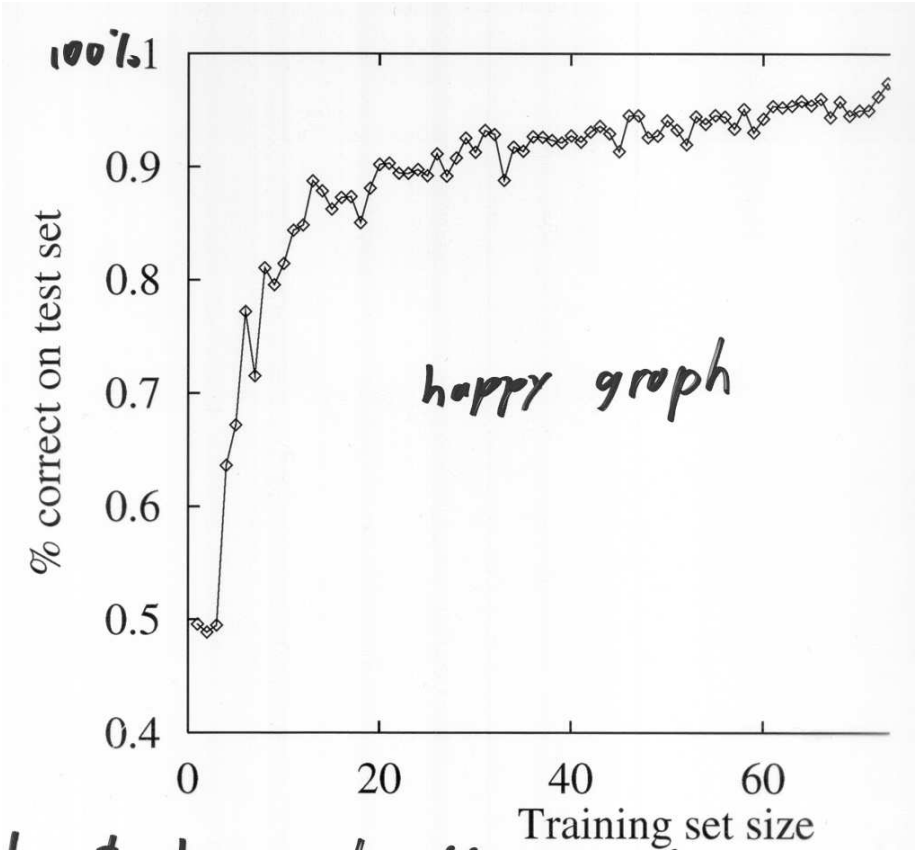
$$\text{Gain}(A) = I(\frac{p}{p+n},\frac{n}{p+n}) - \text{Remainder}(A)$$

Choose attribute with highest gain (among remaining training examples at that node of the tree).

Evaluating learning algorithms

Training set
Test set

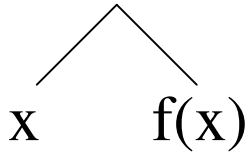
Redividing and altering proportions



Should not change algorithm based on performance on test set!

Algorithms with many variants have an unfair advantage?

Noise & overfitting in decision trees



E.g. rolling die with 3 features: day, month, color

1. χ^2 pruning

Assume (Null Hypothesis) that test gives no info

Expected:

$$\hat{p}_i = p\frac{p_i + n_i}{p+n} \quad \hat{n}_i = n\frac{p_i + n_i}{p+n}$$

$$D = \sum_{i=1}^{v} \frac{(p_i - \hat{p}_i)^2}{\hat{p}_i} + \frac{(n_i - \hat{n}_i)^2}{\hat{n}_i}$$

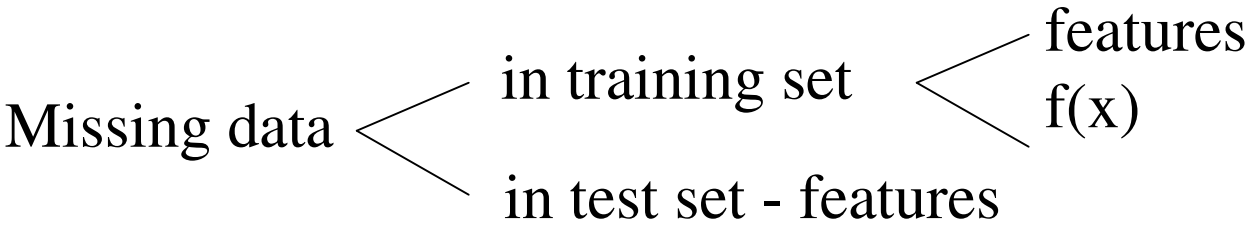
compare to χ^2 table

2. Cross-validation

Split training set into two parts, one for training, one for choosing the hypothesis with highest accuracy.

Pruning also gives smaller, more understandable trees.

Broadening the applicability of decision trees



Multivalued attributes

Info gain gives unfair advantage to attributes with many values → use gain ratio

Continuous-valued attributes

Manual vs. automatic discretization

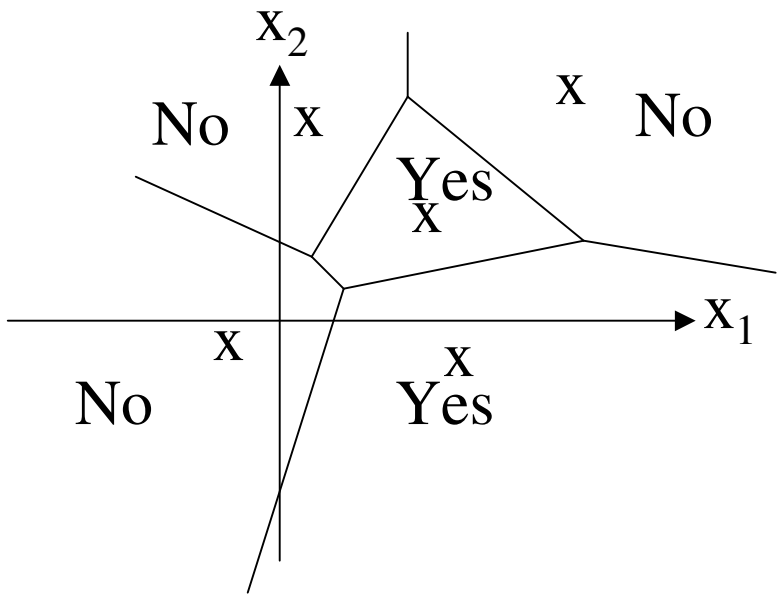
Incremental algorithms.

Instance-based learning

k-nearest neighbor classifier:

For a new instance to be classified, pick k "nearest" training instances and let them vote for the classification (majority rule)

E.g. k=1



Fast learning time (CPU cycles)

Learning general logical descriptions

Goal predicate Q e.g. WillWait
Candidate (definition hypothesis) C_i
Hypothesis: $\forall$instances x, $Q(x) \Leftrightarrow C_i(x)$

E.g. $\forall x$ WillWait(x) $\Leftrightarrow$ Patrons(x,Some)
$\vee$ Patrons (x, Full) $\wedge \neg$Hungry(x) $\wedge$ Type(x,Thai)
$\vee$ Patrons (x, Full) $\wedge$ Hungry(x)

Example Xi
First example:

Alternate $(X_1) \wedge \neg Bar(X_1) \wedge \neg Fri/Sat(X_1) \wedge Hungry(X_1) \wedge \ldots$

and the classification

$WillWait(X_1)$

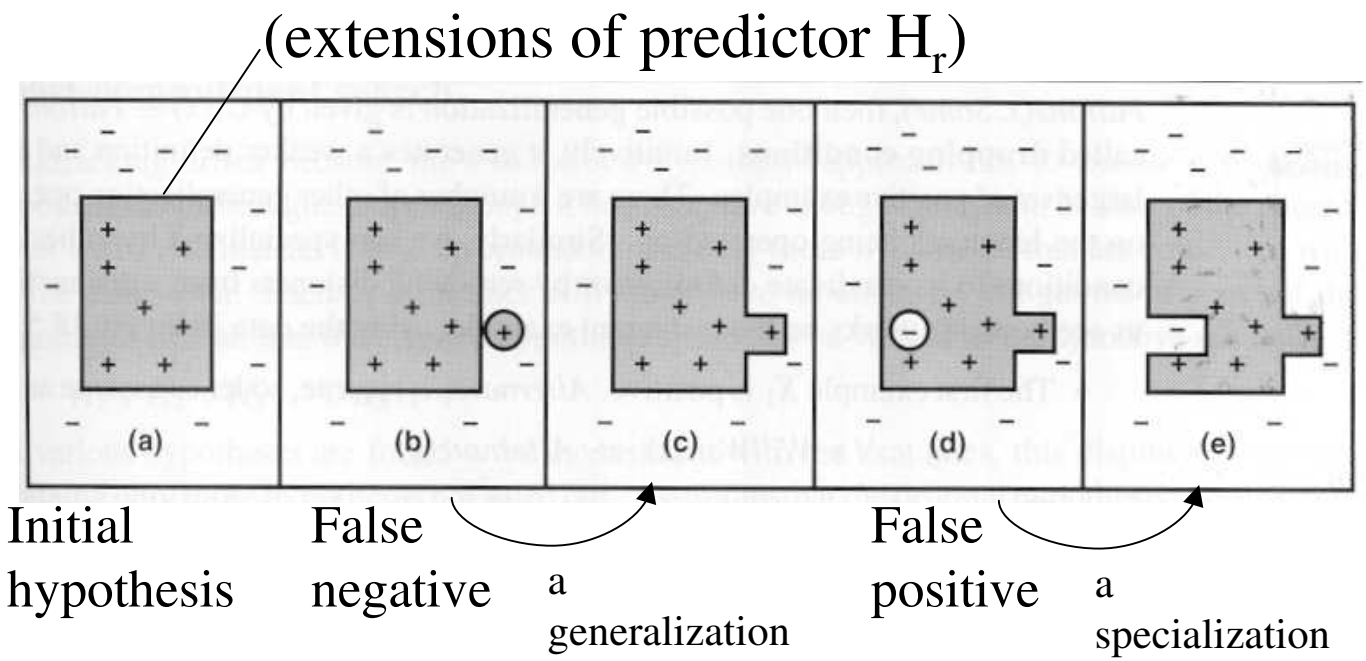
Would like to find a hypothesis that is consistent with training examples.

False negative: hypothesis says it should be negative but it is positive.
False positive: hypothesis says it should be positive but it is negative.

Remove hypothesis that are inconsistent.

In practice, do not use resolution via enumeration of hypothesis space…

Current-best-hypothesis search



Generalization e.g. via dropping conditions

$$\text{Alternate}(x) \wedge \text{Patrons}(x,\text{Some}) \rightarrow \text{Patrons}(x,\text{Some})$$

Specialization e.g. via adding conditions or via removing disjuncts

$$\text{Alternate}(x) \vee \text{Patrons}(x,\text{Some}) \rightarrow \text{Patrons}(x,\text{Some})$$

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
```

But

1. Checking all previous instances over again is expensive.
2. Difficult to find good heuristics, and backtracking is slow in the hypothesis space (which is doubly exponential)

Version Space Learning

Least commitment:
Instead of keeping around one hypothesis and using backtracking, keep all consistent hypotheses (and only those).

aka *candidate elimination*

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

  $V \leftarrow$ the set of all hypotheses
  for each example e in *examples* **do**
    if V is not empty **then** $V \leftarrow$ VERSION-SPACE-UPDATE(V, e)
  end
  return V

---

function VERSION-SPACE-UPDATE(V, e) **returns** an updated version space

  $V \leftarrow \{h \in V : h \text{ is consistent with } e\}$

Incremental: old instances do not have to be rechecked

Version Space Learning

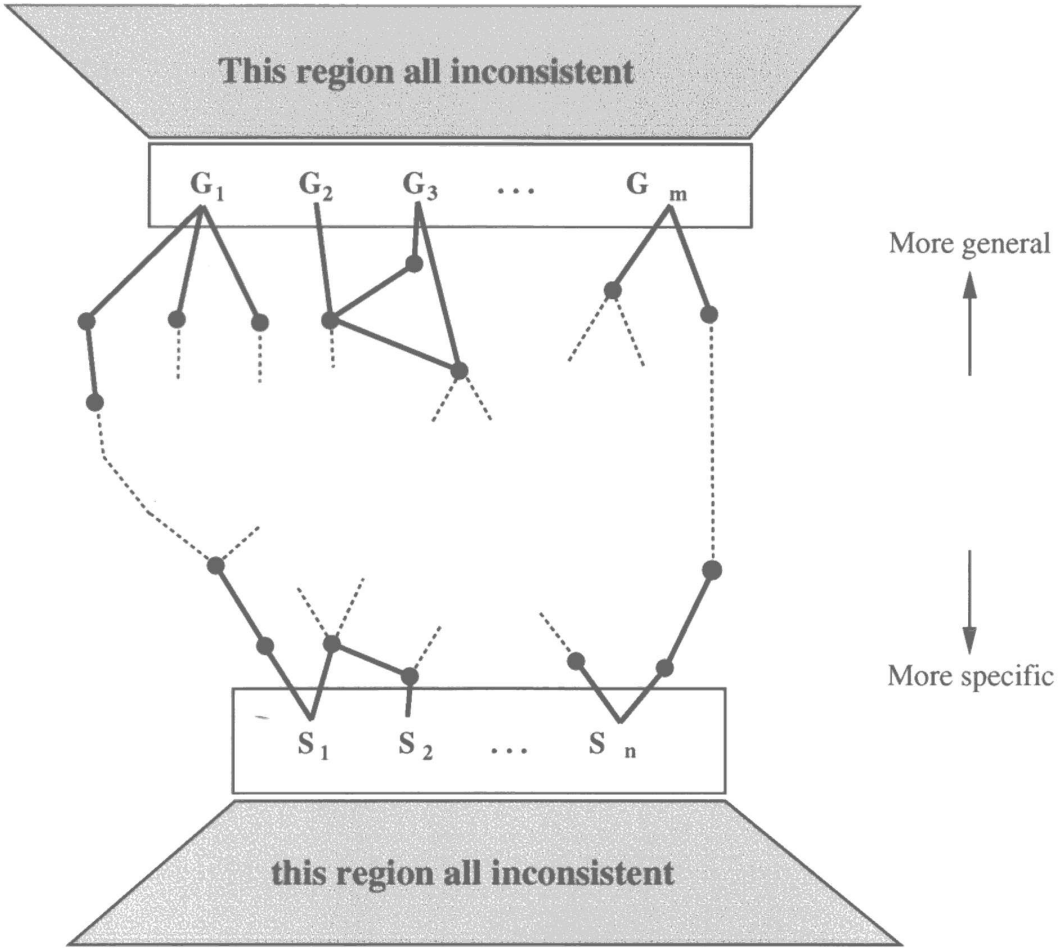
No need to list all consistent hypotheses:

Keep - most general boundary (G-Set)
- most specific boundary (S-Set)

Everything in between is consistent.
Everything outside is inconsistent.

Initialize: G-Set={True}
S-Set={False}

Version Space Learning



Algorithm:

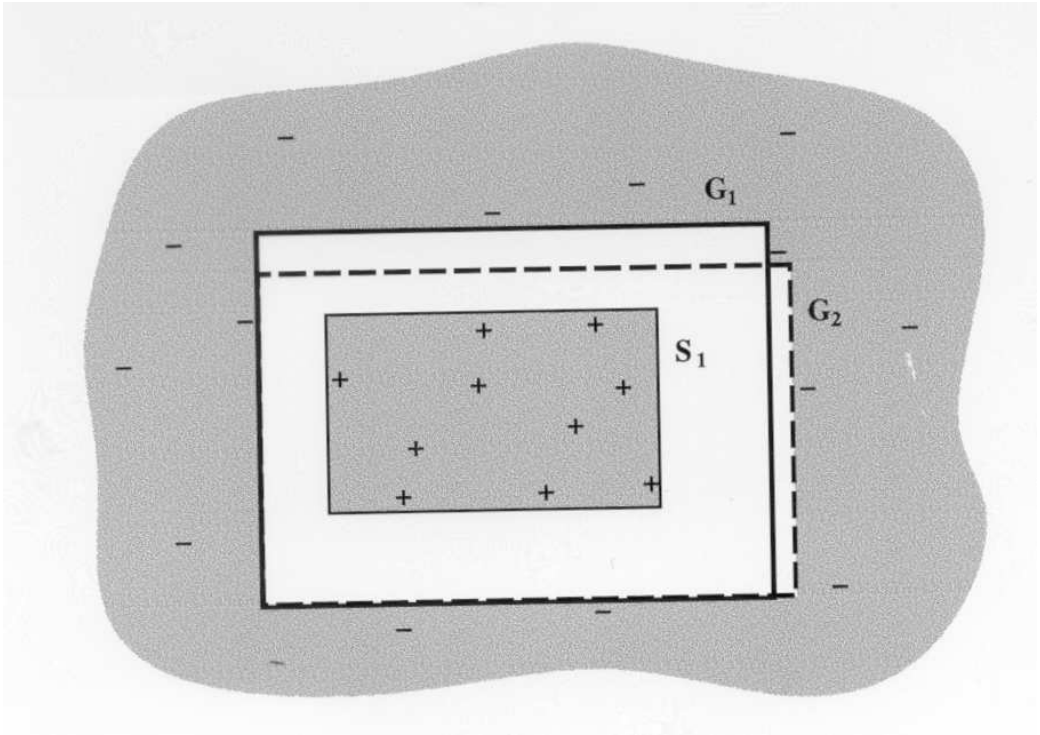
1. False positive for Si:
Si is too general, and there are no consistent specializations for Si, so throw Si out of S-Set

2. False negative for Si:
Si is too specific,
so replace it with all its immediate generalizations.

3. False positive for Gi:
Gi is too general,
so replace it with all its immediate specializations.

4. False negative for Gi:
Gi is too specific, but there are no consistent generalizations of Gi, so throw Gi out of G-Set

Version Space Learning



The extensions of the members of G and S. No known examples lie in between.

Version Space Learning

Stop when:

1. One concept left
2. S-set of G-Set becomes empty, i.e. no consistent hypothesis.
3. No more training examples, i.e. more than one hypothesis is left.

Problems:

1. If there is noise or insufficient attributes for correct classification, the version space collapses.
2. If we allow unlimited disjunction, then

   S-Set will contain a single most specific hypothesis, i.e., the disjunction of the positive training examples.

   G-set will contain just the negation of the disjunction of the negative examples.

   - Use limited forms of disjunction
   - Use generalization hierarchy

   e.g. WaitEsitmate(x,30-60)$\vee$WaitEstimate(x,>60) $\rightarrow$ LongWait(x)

Computational learning theory

Tuomas Sandholm
Carnegie Mellon University
Computer Science Department

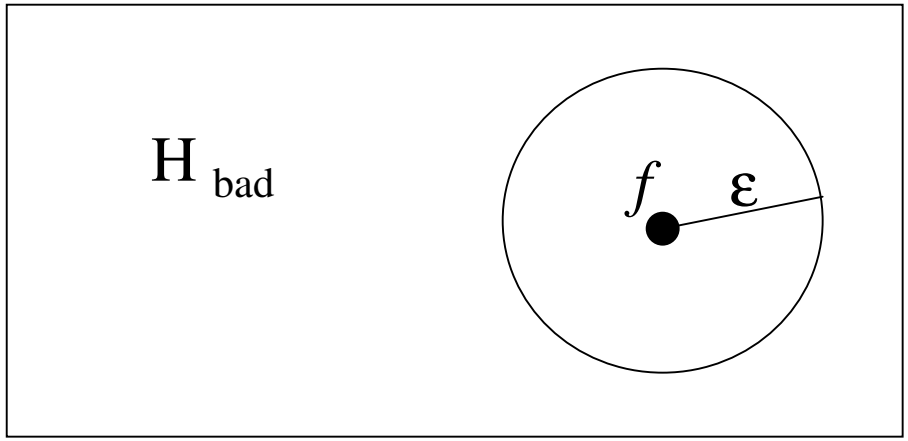
How many examples are needed?

X = set of all possible examples
D = probability distribution from which examples are drawn,
assumed same for training and test sets.
H = set of possible hypotheses
m = number of training examples

$$error(h) = P(h(x) \neq f(x) \mid x \text{ drawn from D})$$

H is approximately correct if error(h) ≤ ε

Hypothesis space:



How many examples are needed?

Calculate the probability that a wrong $h_b \in H_{bad}$ is consistent with the first m training examples as follows.

We know error(h_b) > ε by definition of H_{bad}.
So the probability that h_b agrees with any given example is $\leq (1- \varepsilon)$.
P(h_b agrees with m examples) $\leq (1- \varepsilon)^m$
P(H_{bad} contains a consistent hypothesis) $\leq |H_{bad}|(1- \varepsilon)^m \leq |H|(1- \varepsilon)^m \leq \delta$

Because $1-\varepsilon \leq e^{-\varepsilon}$, we can achieve this by seeing
$m \geq (1/ \varepsilon) (\ln (1/ \delta) + \ln|H|)$ training examples

Sample complexity of the hypothesis space.
Probably approximately correct (PAC).

PAC learning

If H is the set of all Boolean fns of n attributes, then $|H| = 2^{2^n}$

So m grows as 2^n
#possible examples is also 2^n

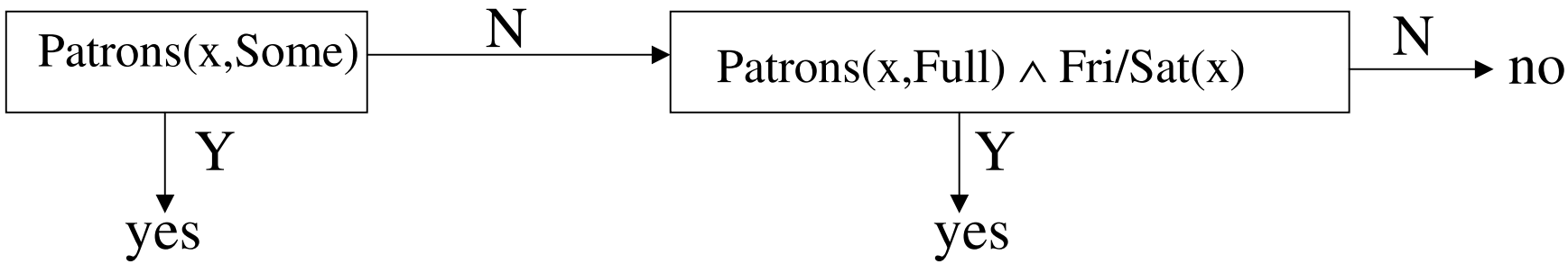
i.e. no learning algorithm for the space of all Boolean fns will do better than a lookup table that merely returns a hypothesis that is consistent with all the training examples.

i.e. for any unseen example, H will contain as many consistent examples predicting a positive outcome as predict a negative outcome.

Dilemma: restrict H to make it learnable?
-might exclude the correct hypothesis

1. Bias toward small hypotheses within H
2. Restrict H (restrict language)

Learning decision lists



Can represent any Boolean function if tests are unrestricted.
But: restrict every test to at most k literals:

k-DL (k-DT ⊂ k-DL)

↑ decision trees of depth k

k-DL(n) ← n attributes

Conj(n,k) = conjunctions of at most k literals using n attributes

Each test can be attached with 3 possible outcomes:
Yes, No, TestNotIncludedInDecisionList
So there are $3^{|Conj(n,k)|}$ sets of component tests.
Each of these sets can be in any order: $|k\text{-DL}(n)| \leq 3^{|Conj(n,k)|}|Conj(n,k)|!$

Learning decision lists

$$\left|Conj(n,k)\right| = \sum_{i=0}^{k} \binom{2n}{i} = O(n^k)$$

So, $\left|k - DL(n)\right| = 2^{O(n^k \log_2(n^k))}$

↓ lots of work

Plug this into $m \geq (1/\varepsilon)(\ln(1/\delta)+\ln|H|)$
to get ↓

$$m \geq \frac{1}{\varepsilon}\left(\ln(\frac{1}{\delta}) + O(n^k \log_2(n^k))\right)$$

This is polynomial in n

So, any algorithm that returns a consistent decision list will PAC-learn in a reasonable #examples (for small k).

Learning decision lists

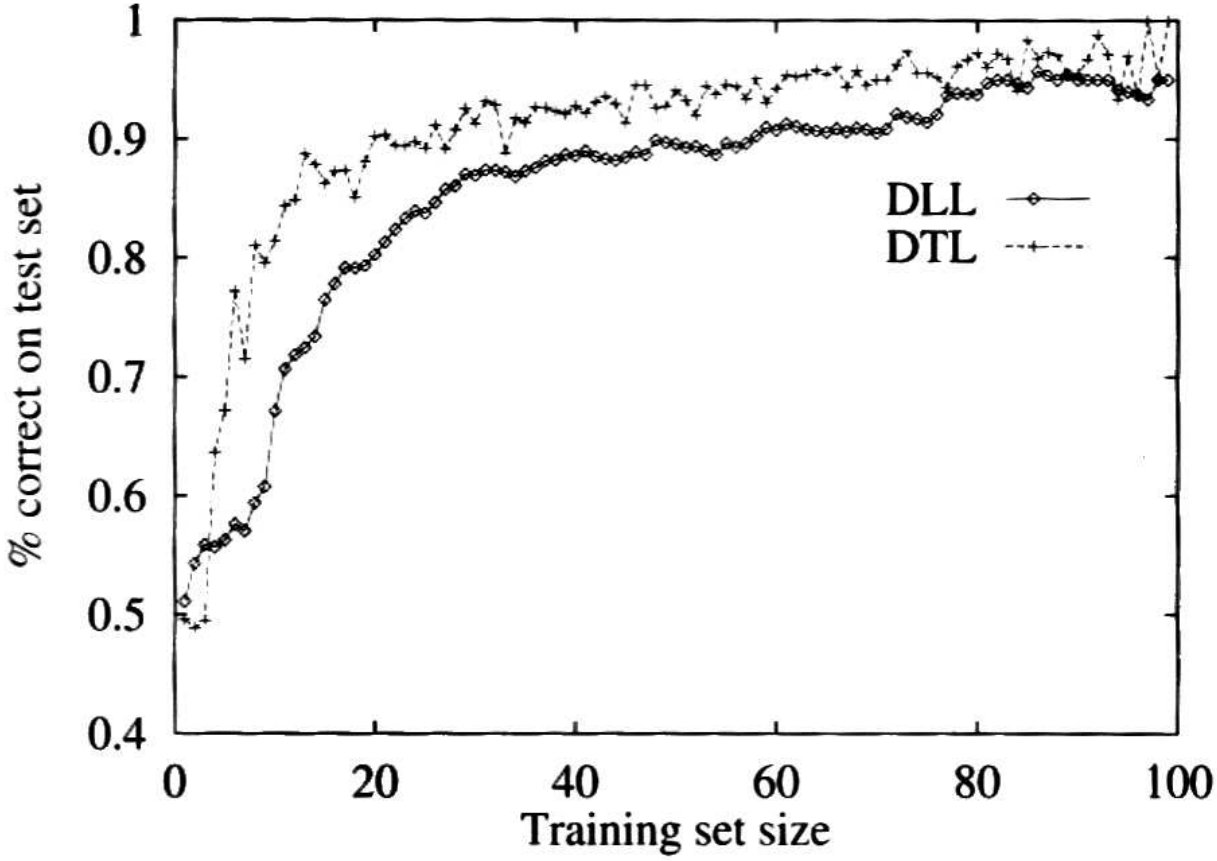
An algorithm for finding a consistent decisions list:
Greedily add one test at a time

```
function DECISION-LIST-LEARNING(examples) returns a decision list, No or failure

  if examples is empty then return the value No
  t ← a test that matches a nonempty subset examples_t of examples
       such that the members of examples_t are all positive or all negative
  if there is no such t then return failure
  if the examples in examples_t are positive then o ← Yes
  else o ← No
  return a decision list with initial test t and outcome o
          and remaining elements given by DECISION-LIST-LEARNING(examples − examples_t)
```

The theoretical results do not depend on how the tests are chosen.

Decision list learning vs. decision tree learning



In practice, prefer simple (small) tests. Simple approach: pick smallest test, no matter how small the set (>0) of examples is that it matters for.