CSCI 3230

Fundamentals of Artificial Intelligence

Chapter 18

Learning From Examples

Fall 2013

Outline

- A General Model of Learning Agents
- Inductive Learning
- Learning Decision Trees
- Using Information Theory
- Learning General Logical Descriptions
- Why Learning Works: Computational Learning Theory

Learning from observation

- Learning: Precepts not only for acting, but also for improving the agent's ability to act in the future
- Learning involves the interaction between the agent and the world through observation by the agent of its own decision-making processes.
- To improve their behavior through study of their own experience.
- To acquire new knowledge or refine existing knowledge

Data→ Information→ Knowledge
?Applications

A General Model of Learning Agents

- A learning agent can be divided into four conceptual components, as shown in Fig 18.1. (See chap.2 for details)

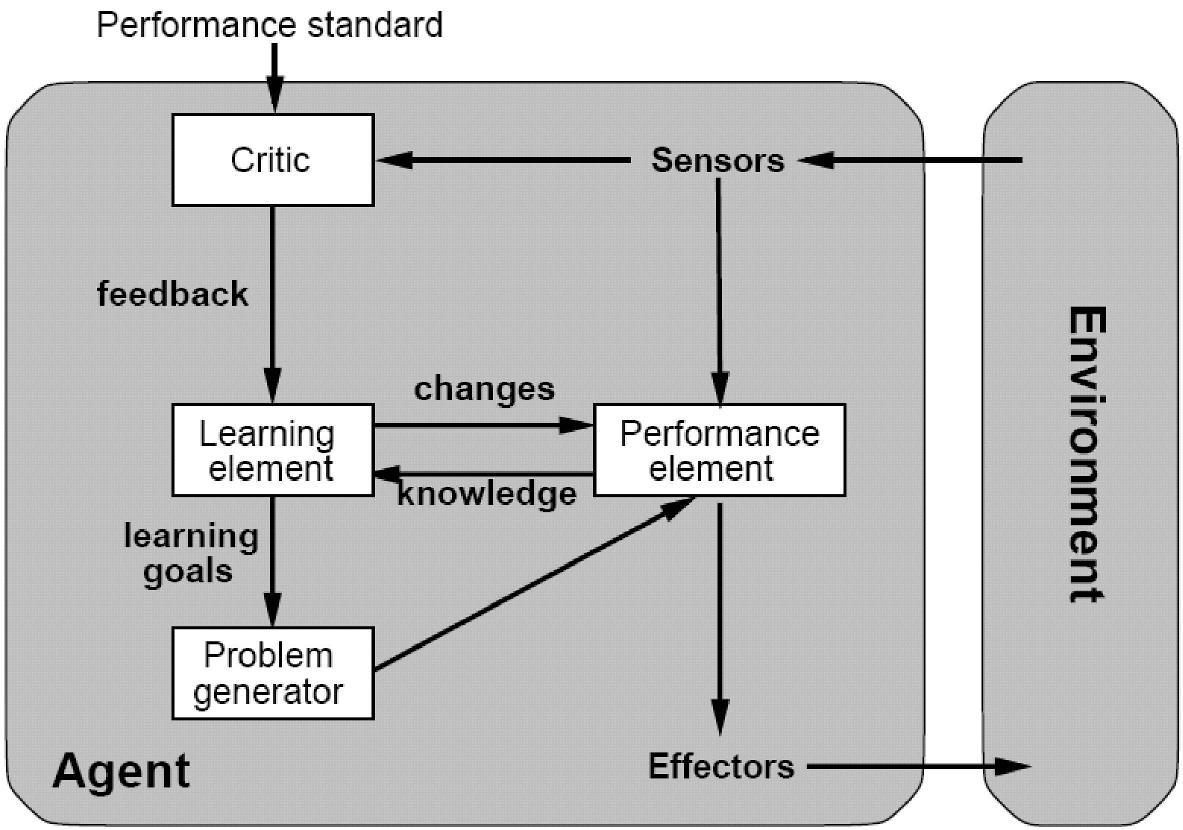


Fig. 18.1 A general model of learning agents

A General Model of Learning Agents

The design of the learning element is affected by 4 major issues:

- Which components of the performance element are to be improved or learnt. (p.6)
- What representation is used for those components (p.7)
- What feedback is available (pp.8-9)
- What prior information is available (p.10)

A General Model of Learning Agents
- Components of the performance element

Many ways to build the performance element. Some of the information/knowledge or components (of the KB) are:

1. A direct mapping from conditions on the current state to actions.
2. A means to infer relevant properties of the world from the percepts sequence.
3. Information about the way the world evolves. (states)model
4. Utility information indicating the desirability of world states.
5. Action-value information indicating the desirability of particular actions in particular states.
6. Goals that describe classes of states whose achievement maximizes the agent's utility.

A General Model of Learning Agents
– Representation of the components

- These components can be represented using any of the representation schemes in this book and learnt
- E.g.
  - deterministic descriptions such as linear weighted polynomials for utility functions in game-playing programs regression
  - propositional and first-order logical sentences for all of the components in a logical agent; and
  - probabilistic descriptions such as belief (Bayesian) networks for the inferential components of a decision-theoretic agent.
  - … Rules; decision trees; NN; SVM; Non-linear integrals; semantic & hierarchy networks; OO;

A General Model of Learning Agents
– Available feedback (c.f. human learning)

- **Unsupervised learning**
  - No hint at all about the correct outputs.
  - It learns patterns in the inputs. E.g.attendence
  - It learns what to do based on a utility function.
  - E.g. reinforcement learning is a form of unsupervised learning; clustering; discovery learning; robot discovers the concept of "door" itself.

- **Reinforcement learning**
  - In learning the condition-action component, the agent receives some evaluation of its action but is not told the correct action.
  - The hefty bill (penalty, e.g. braking, hit the car in front) is called a **reinforcement** – Rewards or punishments

A General Model of Learning Agents
– Available feedback (c.f. human learning)

- Supervised learning
  - In predicting the outcome of an action, the available feedback generally tells the agent what the correct outcome is.
  - Both the inputs & outputs of a component can be perceived (Often, the outputs are provided by a friendly teacher.)
  - E.g. car braking – guess stopping in 10m; supervisor: actually should be 15m. Classification problems– given training examples with class labels.

- Semi-supervised learning
  - Given some labeled examples and some un-labeled examples.

A General Model of Learning Agents
– Available feedback (c.f. human learning) / Prior knowledge

Prior knowledge

- The majority of learning research in AI, CS, and psychology, the agent begins with no knowledge about what it is trying to learn. It only has access to the examples presented by its experience.

- Most human learning takes place with background knowledge. Some psychologists and linguists claim that even newborn babies exhibit knowledge of the world. E.g. Analogy, meta knowledge, problem nature

e.g. discrete vs. continuous; deterministic vs. stochastic; landscape;

Inductive Learning 歸納法

- In supervised learning, the learning element is given the correct (or approximately correct) value of the function for particular inputs, and changes its representation (h) of the function to try to match the information provided by the feedback ($\delta = h - f$).

- More formally, an example is a pair (x, $f(x)$), where x is the input and $f(x)$ is the output of the function applied to x.

- The task of pure inductive inference (or induction):

  given a collection of examples of f, return a function h that approximates f. The function h (e.g.??) is called a hypothesis.

Inductive Learning

Data mining

- The true f is unknown, so there are many choices for h, but without further knowledge, we have no way to prefer (b), (c), or (d). (see Fig 18.2)

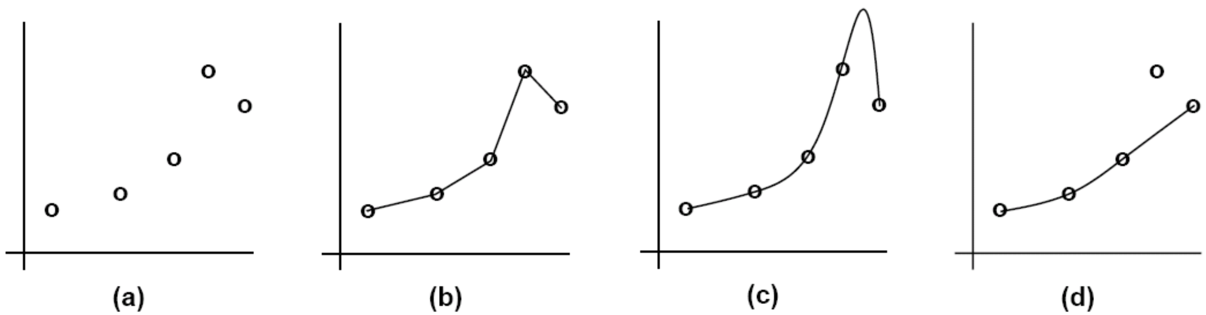


Fig. 18.2 in (a) we have some example (*input*, *output*) pairs. In (b), (c) and (d) we have 3 hypotheses for function from which there example could be drawn. (which one better?)

- Any preference for one hypothesis over another, beyond mere consistency with the examples, is called a bias.
- Because of a large number of possible consistent hypotheses, all learning algorithms exhibit some sort of *bias*:
  - E.g. simplest hypothesis, avoid over-fitting, noise, & outliners.
  - regularized learning

Inductive Learning

- The *choice* of *representation* for the desired function is probably the most important issue facing the designer of a learning agent.

- In learning there is a fundamental trade-off between expressiveness – is the desired function representable in the representation language? – and efficiency –is the learning problem tractable for a given choice of representation.
  - Search space (complexity)? Free lunch? Optimal?
  - (E.g. straight line Vs polynomial).

- i.e. model complexity – effectiveness – efficiency – memory

Learning Decision Trees
– Decision trees as performance elements

- Decision tree induction is one of the simplest and yet most successful forms of learning algorithm in the area of *inductive learning*.

Decision trees as performance elements

- A **decision tree** takes as input an object or situation described by a set of properties (*attributes*), and outputs a yes/no "decision". Decision trees therefore represent **Boolean functions**.
- Functions with a larger range of outputs can also be represented, but for simplicity usually stick to the Boolean case.
- internal node = a test ; branches are labeled with possible values of the test. Each leaf node specifies the Boolean value result if reached. C4.5; See 5;

Learning Decision Trees
– Decision trees as performance elements

- *Example:* To learn a definition for the **goal predicate** (concept) WillWait. 1st: decide *attributes* available to describe the problem domain:

1. Alternate: whether there is a suitable alternative restaurant nearby.
2. Bar: whether the restaurant has a comfortable bar area to wait in.
3. Fri/Sat: true on Fridays and Saturdays.
4. Hungry: whether we are hungry.
5. Patrons: how many people are in the restaurant (values are *None*, *Some*, *Full*).
6. Price: the restaurant's price range ($, $$, $$$).
7. Raining: whether it is raining outside.
8. Reservation: whether we made a reservation.
9. Type: the kind of restaurant (*French*, *Italian*, *Thai or Burger*).
10. WaitEstimate: the wait estimated by the host (0–10 minutes, 10–30, 30–60, >60).

- The decision tree is given in Fig 18.4

Learning Decision Trees
– Decision trees as performance elements

White box: internal node: a test of an attribute;
Line: branch: possible values of the attribute;
Shaded box: leave node: **answer**

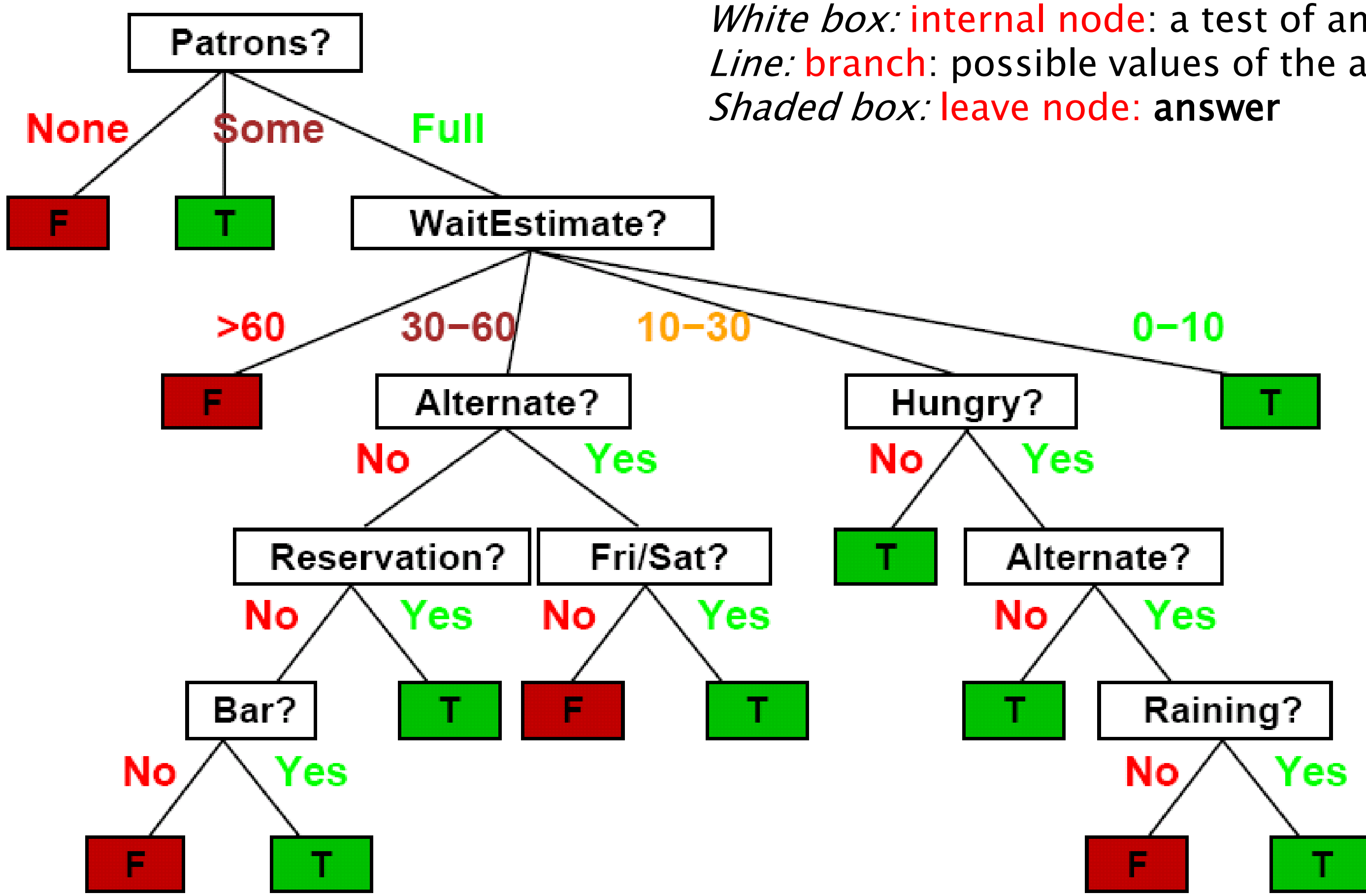


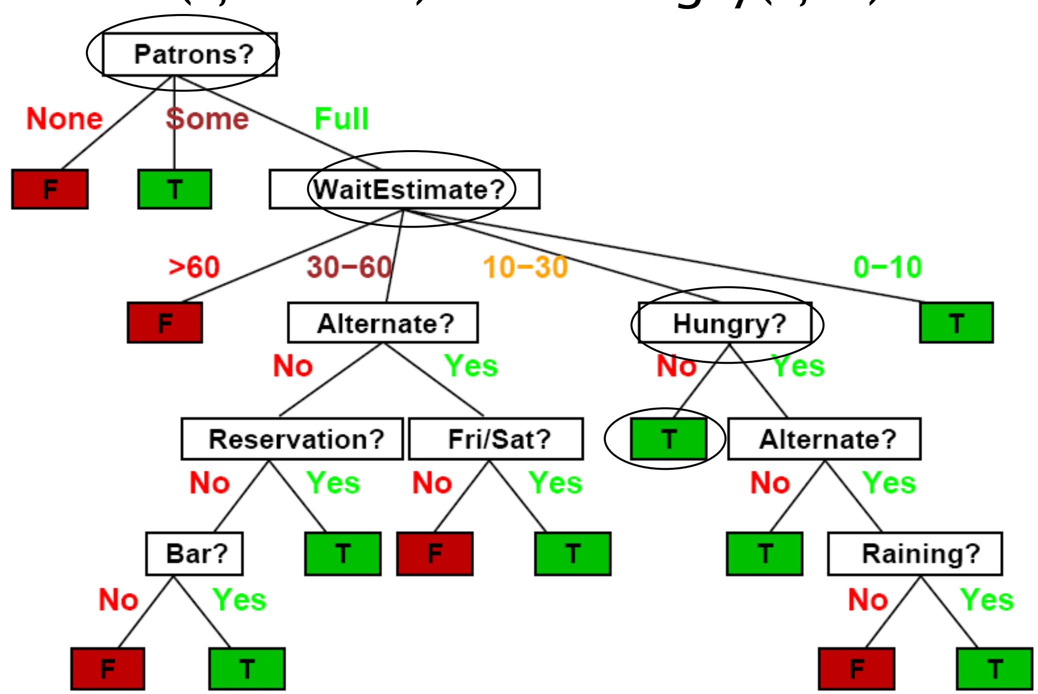
Fig. 18.4 A decision tree for deciding whether to wait for a table

Learning Decision Trees
– Decision trees as performance elements

- A path to a *Yes*–node can be expressed by a conjunction of tests implication.
- E.g., the path for a restaurant full of patrons, with an estimated wait of 10–30 minutes when the agent is not hungry is expressed by the logical sentence:

$\forall r$ Patrons(r, Full) $\wedge$ WaitEstimate(r, 10–30) $\wedge$ $\neg$ Hungry(r, N) $\Rightarrow$ WillWait(r)

r: variable for restaurant



Learning Decision Trees
– Expressiveness of decision trees

- Decision trees are fully expressive within the class of propositional languages, i.e. any Boolean function can be written as a decision tree.
- Trivially done by having each row in the truth table for the function correspond to a path in the tree.
- Not a good way to represent the function, because the truth table is exponentially large in the number of attributes (2^n)
- Clearly, decision trees can represent many functions with much smaller trees.

Learning Decision Trees
– Inducing decision trees from examples

- An **example** is described by the *values* of the *attributes* and the value of the *goal* predicate – called the **classification** of the example. If *true*, we call it a **positive** example; otherwise, a **negative** example.

- A set of examples $X_1, \ldots, X_{12}$ for the restaurant domain is shown in Figure 18.5.
  - Positive examples – the goal WillWait is *true* (X_1, X_3, ...)
  - Negative examples – the goal WillWait is *false* (X_2, X_5,...).
  - The complete set of examples is called the **training set.**

Learning Decision Trees
– 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*

Fig. 18.5 Examples for the restaurant domain. Price? discretization

Learning Decision Trees
– Inducing decision trees from examples

- Extracting a pattern is to describe a large number of cases in a concise way. Not just a correct decision tree that fits the examples, but a concise one.
- This is a general principle of inductive learning often called **Ockham's razor**. The most likely hypothesis is the simplest one that is consistent with all observations. (?)
- There are far fewer simple hypotheses than complex ones, so only a small chance for any wildly incorrect simple hypothesis to be consistent with all observations. Hence, other things being equal, a simple hypothesis consistent with the observations is more likely to be correct than a complex one.
- Unfortunately, finding the smallest decision tree is intractable, but with some **simple heuristics**, we can find a smallish one.

Learning Decision Trees
– Inducing decision trees from examples

- Figure 18.6 shows how the algorithm gets started. Given 12 training examples, classified into positive and negative sets. Then decide which attribute to use as the first test in the tree. (?how) why it is not optimal
- Patrons is a fairly important attribute, because if the value is None or Some, then we are left with example sets for which we can answer definitively (No and Yes, respectively).
- Type is a poor attribute, because it leaves 4 possible outcomes, with the same number of positive and negative answers. (?so)

Learning Decision Trees
– Inducing decision trees from examples

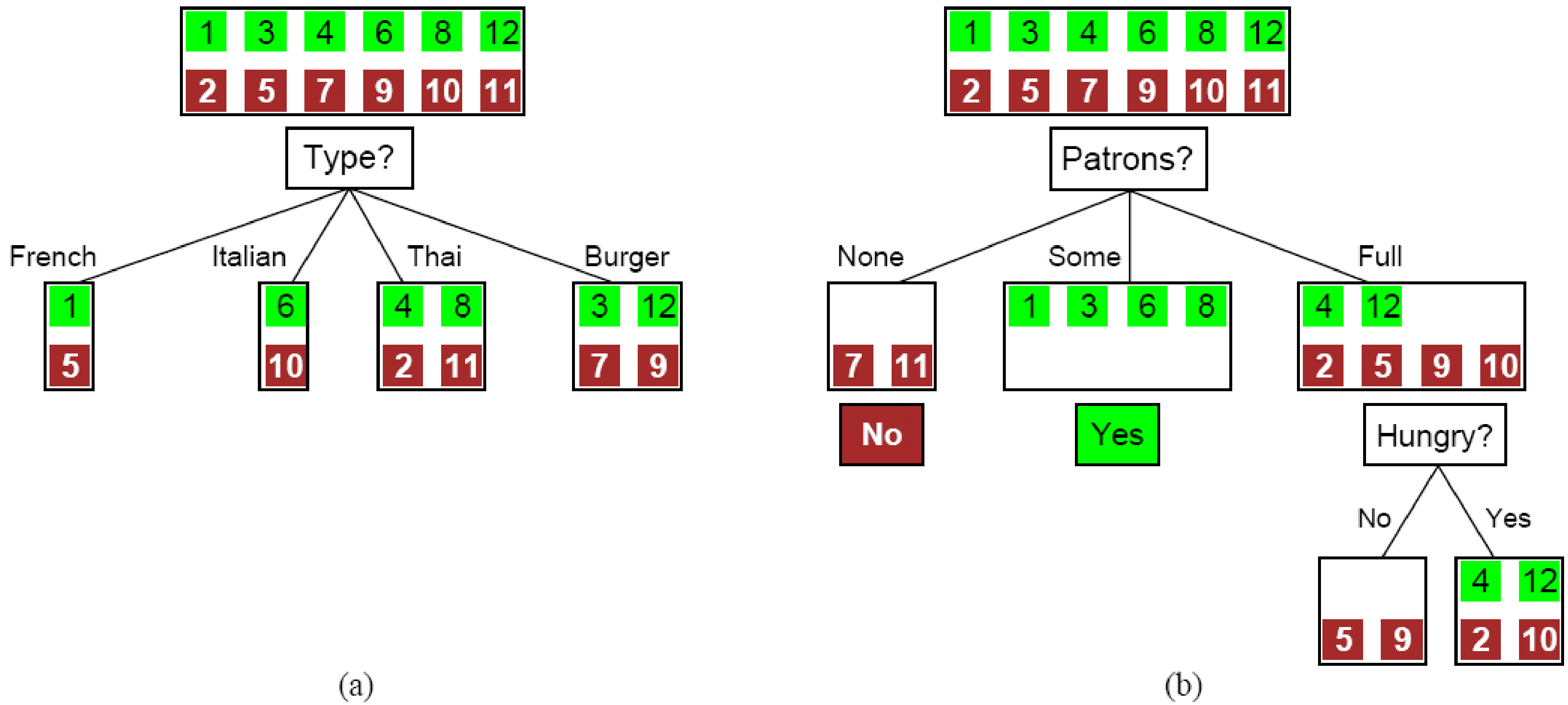


Fig 18.6 Splitting the examples by testing on attributes. (a) Type is a poor choice, no distinction between +ve and –ve examples, and (b) Patrons is a good attribute to test first, and Hungry is a fairly good second test, given that Patrons is the first test.

Learning Decision Trees
– Inducing decision trees from examples

- After the first attribute test splits up the examples, each outcome is a new decision tree learning problem in itself, with fewer examples and one fewer attribute. There are 4 cases to consider for these recursive problems: (see Fig. 18.6(b))

1. If there are some +ve and some –ve examples, then choose the best attribute to split them.
2. If all the remaining examples are +ve (or all –ve), then done: we can answer Yes or No.
3. If there are no examples left, it means that no such example has been observed, and we return the majority classification of the node's parent.

Learning Decision Trees
– Inducing decision trees from examples

4. If there are no attributes left, but both +ve and –ve examples, which means that these examples have exactly the same description, but different classifications. This happens when

   (i) some of the data are incorrect, i.e. noise in the data;

   (ii) the attributes do not give enough information to fully describe the situation; or

   (iii) the domain is truly nondeterministic.

One simple way out: use a majority vote.

Learning Decision Trees
– Inducing decision trees from examples

unuse example, unuse attribute

function Decision-Tree-Learning(*examples*, *attributes*, *default*) **returns** a decision tree

default if from the previous call, implies the parent

inputs: examples, set of examples
attributes, set of attributes
default, default value for the goal predicate

if *examples* is empty **then return** *default* *//majority-value of parent*
else if all *examples* have the same classification **then return** the classification *// leaf node*
else if *attributes* is empty **then return** Majority-Value(*examples*)
else
best ← Choose-Attribute(*attributes*, *examples*) //e.g. info gain
tree ← a new decision tree with root test *best* *//sub-tree root*
for each value v_i *of best* **do**
$examples_i$ ← {elements of examples with $best = v_i$} // less classified examples
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*

Fig 18.7 The decision tree learning algorithm

Is this algo optimal??

Learning Decision Trees
– Inducing decision trees from examples

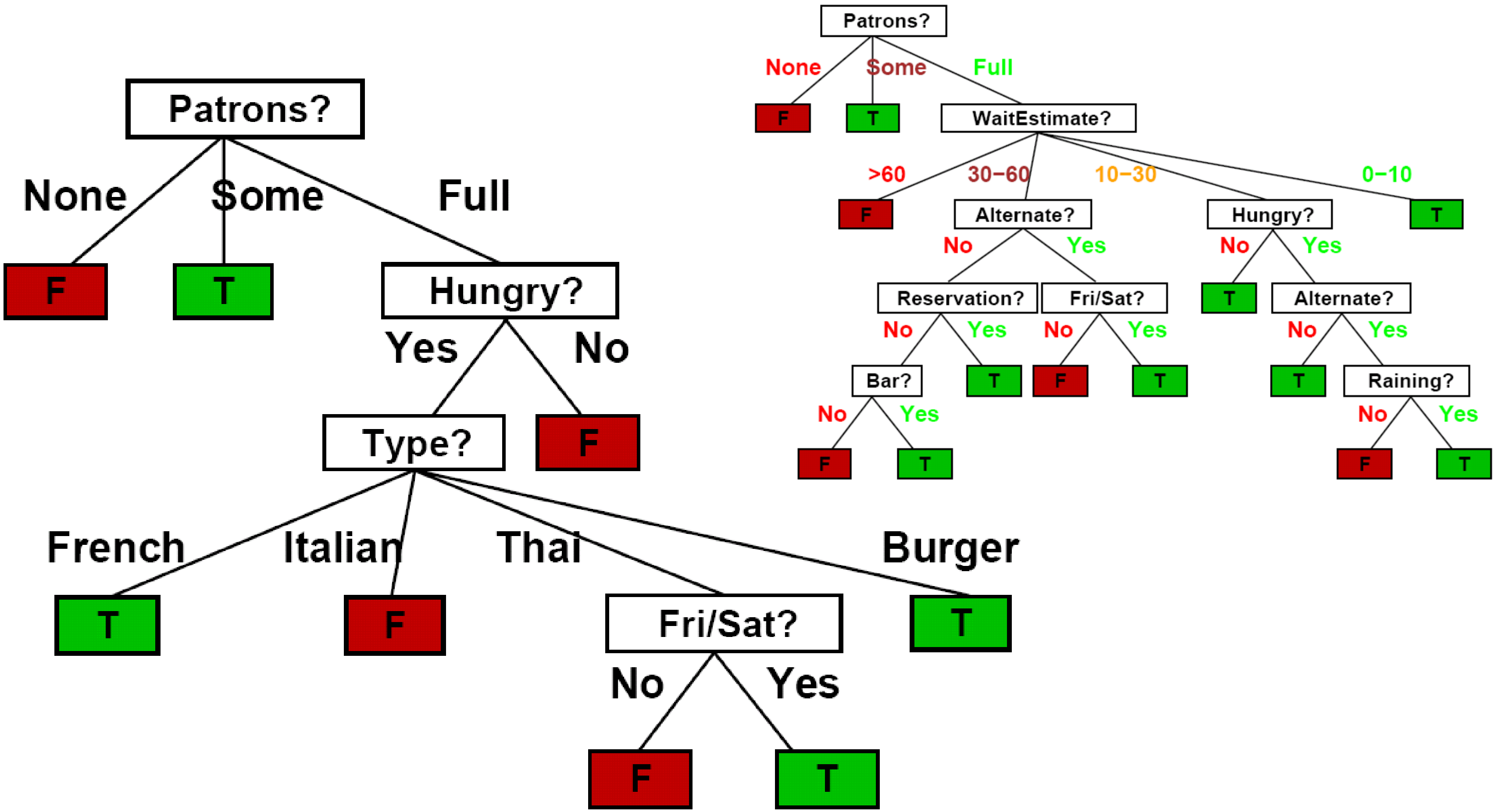


Fig 18.8 The decision tree induced from the 12-example training set.
(different from Fig 18.4 why?)

Learning Decision Trees
– Assessing the performance of the learning algorithm

- Good if it produces hypotheses that predicts accurately the classifications of unseen examples.
- Assess the quality of a hypothesis by checking its predictions against the correct classification.
- We do this on a set of examples known as the **test set**. We can adopt the following methodology:

1. Collect a large set of examples.(may not be possible?)
2. Divide it into two disjoint sets: the **training set** and the **test set**.
3. Use the learning algorithm with the training set as examples to generate a hypothesis h. Then test with test set.
4. Repeat steps 1 to 4 for different sizes of training sets and different randomly selected training sets (say, 20) of each size:

Learning Decision Trees
– Assessing the performance of the learning algorithm

- Take the average prediction quality of these trails as a function of the size of the training set
- Plot the **learning curve** for the algorithm on the particular domain. The learning curve for DECISION-TREE-LEARNING with the restaurant examples is shown in Figure 18.9.
- Notice (next page) that as the training set grows, the prediction quality increases. (Hence, such curves are also called **happy graphs**.) A good sign that there is indeed some pattern in the data and the learning algorithm is picking it up.

Learning Decision Trees
– Assessing the performance of the learning algorithm

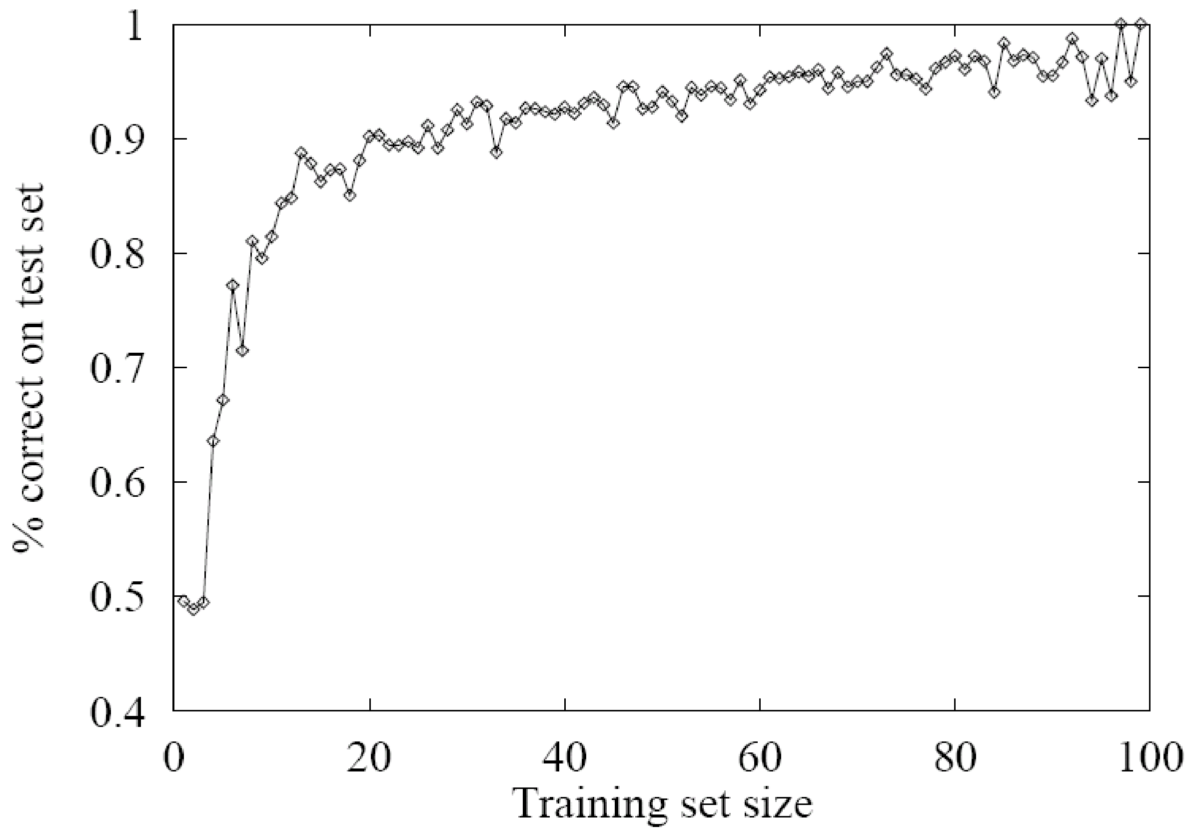


Fig 18.9. A learning curve for the decision tree algorithm on 100 randomly generated examples in the restaurant domain. The graph summarizes 20 trails of each size.

Using Information Theory

Naive Bayes

- In general, for an event to happen, if the possible answers v_i have probabilities $P(v_i)$, then the information content I of the actual answer is given by ??

I means Information gain

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

- This is just the average information content of the n events (the $-\log_2 P$ terms) weighted by the probabilities of the events. probability of it happening To check this equation, for the tossing of a fair coin we get

$$I(\frac{1}{2},\frac{1}{2}) = -\frac{1}{2}\log_2\frac{1}{2} - \frac{1}{2}\log_2\frac{1}{2} = 1 bit$$

0.01x6.64+0.99x0.01=.0664+.0099

- If the coin is loaded to give 99% heads we get $I(1/100, 99/100) = 0.08$ bit, and as the probability of heads go to 1, the information of the actual answer goes to *0*.

$\log_2$ of Probabilities

does not make sense to have a negative information

we need to take log: can show small propability case happen-> it is important finding, big information!

P less than 1 -> always negative

$\log_2(P)$										
P	**0**	**0.01**	**0.02**	**0.03**	**0.04**	**0.05**	**0.06**	**0.07**	**0.08**	**0.09**
0	inf.	-6.64	-5.64	-5.06	-4.64	-4.32	-4.06	-3.84	-3.64	-3.47
0.1	-3.32	-3.18	-3.06	-2.94	-2.84	-2.74	-2.64	-2.56	-2.47	-2.4
0.2	-2.32	-2.25	-2.18	-2.12	-2.06	-2	-1.94	-1.89	-1.84	-1.79
0.3	-1.74	-1.69	-1.64	-1.6	-1.56	-1.51	-1.47	-1.43	-1.4	-1.36
0.4	-1.32	-1.29	-1.25	-1.22	-1.18	-1.15	-1.12	-1.09	-1.06	-1.03
0.5	-1	-0.97	-0.94	-0.92	-0.89	-0.86	-0.84	-0.81	-0.79	-0.76
0.6	-0.74	-0.71	-0.69	-0.67	-0.64	-0.62	-0.6	-0.58	-0.56	-0.54
0.7	-0.51	-0.49	-0.47	-0.45	-0.43	-0.42	-0.4	-0.38	-0.36	-0.34
0.8	-0.32	-0.3	-0.29	-0.27	-0.25	-0.23	-0.22	-0.2	-0.18	-0.17
0.9	-0.15	-0.14	-0.12	-0.1	-0.09	-0.07	-0.06	-0.04	-0.03	-0.01

e.g. $\log_2(0.12) = -3.06$; $\log_2(1) = 0$;

the smaller the *P*, the higher the information content log *P*;

Using Information Theory

- For correct decision tree learning, we need to estimate the information needed for (or contained in) a correct classification.
- An estimate of the probabilities of the possible answers before any attributes tested is given by the proportions of +ve and –ve examples in the training set.
- Suppose the training set contains *p* +ve examples and *n* –ve examples. Then an estimate of the information contained in a correct answer is

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

e.g. Fig 18.6 $\boldsymbol{p = n = 6,\ \ I = 1}$

- Now a test on a single attribute *A* will not usually give us all, but some information. We can measure exactly how much by looking at how much information we still need *after* the attribute test.

Inf given by a test = (Inf needed before) – (Inf needed after the test)

(Inf gain using *A*) $Remiander(A)$

Using Information Theory

To calculate Inf needed after the test of *A*:

- Any attribute *A* divides the training set E into subsets $E_1, \ldots, E_v$ according to their values for *A*, where *A* can have v distinct values. Each subset E_i has p_i +ve examples and n_i –ve examples, so if we go along that **branch** we will need an additional $I(p_i/(p_i+n_i), n_i/(p_i+n_i))$ bits of information to answer the question.

- A random example has the i^{th} value for the attribute with probability $(p_i+ n_i) / (p+n)$, so on average, after testing attribute *A*, we will need

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

bits of information to classify the example.

Using Information Theory

- The information gain from the attribute test is defined as the difference between the original information requirement and the new requirement:

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

  and the heuristic used in the CHOOSE-ATTRIBUTE function is just to choose the attribute with the largest gain.

- Looking at the attributes Patrons and Type and their classifying power, as shown in Figure 18.6 we have

?Why = 1?

$$Gain(Patrons) = 1 - \left[\frac{2}{12} I(0,1) + \frac{4}{12} I(1,0) + \frac{6}{12} I\left(\frac{2}{6}, \frac{4}{6}\right)\right] \approx 0.541 bits$$

(1st 2 terms in [ ] above = 0 ➔ no more inf needed)

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

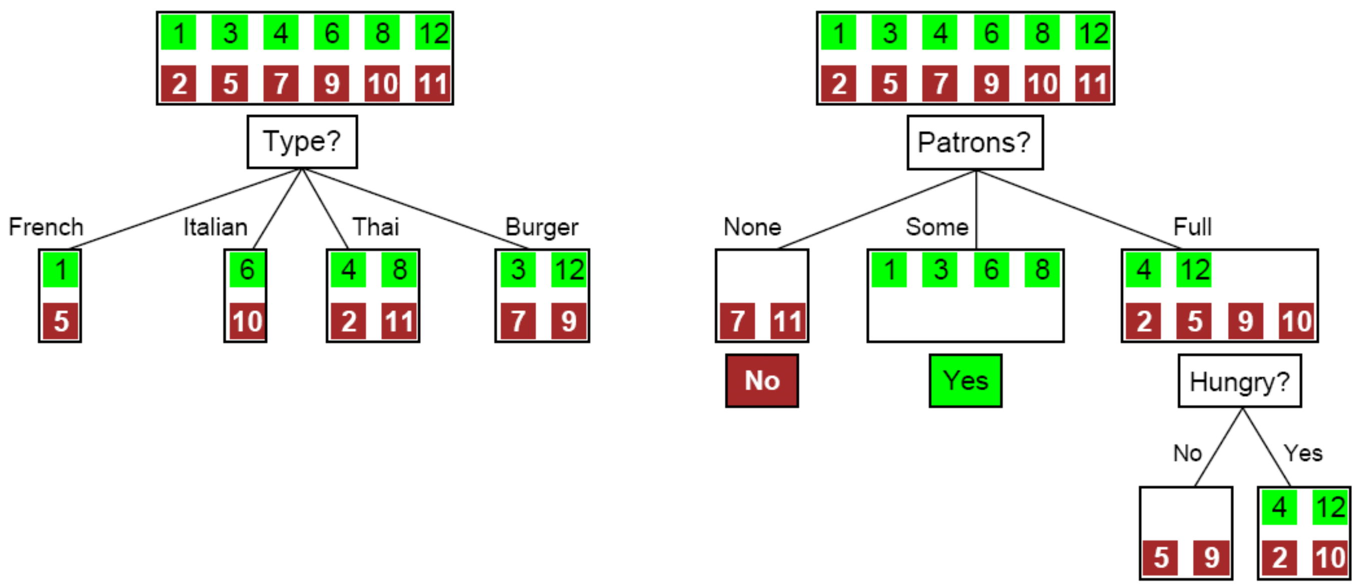
Using Information Theory

- Looking at the attributes Patrons and Type and their classifying power, as shown in Figure 18.6 we have

$$Gain(Patrons) = 1 - [\frac{2}{12} I(0,1) + \frac{4}{12} I(1,0) + \frac{6}{12} I(\frac{2}{6}, \frac{4}{6})] \approx 0.541 bits$$

(1st 2 terms in [ ] above = 0 ➔ no more inf needed)

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



(a) (b)

Using Information Theory
– Noise and overfitting

- If many possible hypotheses, not to use freedom to find meaningless "regularity" in the data.
  - This problem is called overfitting.
  - A general phenomenon, occurs even when the target function is not random.
  - Afflicts every kind of learning algorithm, not just decision trees.
  - It'll fit only training data but not testing data.
- Decision tree pruning: Pruning works by preventing recursive splitting on attributes that are not clearly relevant. E.g. ignore attributes with low Inf gains or use χ^2 measure. dependence test
- With pruning --> smaller tree and learning can tolerate more noise in examples.

Ockham's razor; Feature selection; regularized learning

Using Information Theory
– Noise and overfitting

- **Cross-validation** is another technique that eliminates the dangers of overfitting.
  - It tries to estimate how well the current hypothesis will predict unseen data.
  - Set aside some fraction of the known data, and use it to test the hypothesis induced from the rest of the known data.
  - Do this repeatedly with different subsets of the data, with the results averaged.
    E.g. 10-fold; 5-fold

Using Information Theory
– Broadening the applicability of decision trees

- **Missing data:** In many domains, not all the attribute values are known for every example: not recorded, or too expensive to obtain. 2 problems:
  - (1) Given a complete decision tree, how should one classify an object that is missing one of the test attributes?
  - (2) How should one modify the information gain formula when some examples have unknown values for the attribute?
    E.g. guess by statistics; infer by rules; certainty factors
- **Multivalued attributes:** When an attribute has a large number of possible values, the information gain measure gives an inappropriate indication of the attribute's usefulness.
  E.g. Restaurant Name (Singleton)
- **Continuous-valued attributes:** Attributes such as Height and Weight have a large or infinite set of possible values. Therefore not well-suited for decision-tree learning in raw form.
  - To **discretize** the attribute.
    E.g. Price (continuous) --> $ $$ $$$ (discrete)

Learning General Logical Descriptions
-Hypothesis

- To learn more general kinds of logical representation
- Inductive learning viewed as searching for a good hypothesis in a large hypothesis space – defined by the representation language used.

Hypothesis

- Start with a (unary) goal predicate Q (e.g. in the restaurant domain, Q is WillWait)
- A candidate definition C_i of the goal for each hypothesis H_i is a logical sentence of the form

$$\forall x\ Q(x) \Leftrightarrow C_i(x)$$

Learning General Logical Descriptions
-Hypothesis

- E.g. the decision tree in Fig 18.8 H_r:
  - $\forall r\ WillWait(r) \Leftrightarrow Patron(r, some)$
    $\lor\ (Patrons(r, Full) \land Hungry(r) \land Type(r, French))$
    $\lor\ (Patrons(r, Full) \land Hungry(r) \land Type(r, Thai) \land Fri/Sat(r))$
    $\lor\ (Patrons(r, Full) \land Hungry(r) \land Type(r, Burger))$
- **H** denotes the hypothesis space $\{H_1 \ldots H_n\}$. The learning algorithm believes that one of the hypotheses is correct
- Each hypothesis predicts a certain set of examples– i.e. those satisfy its candidate definition – will be examples of the goal predicate. This set is called the extension of the predicate.

  2 hypotheses with difference sets of extensions are inconsistent.

Learning General Logical Descriptions
–Example

- For example X_i, The classification should be $Q(X_i)$ for a positive example and $\neg Q(X_i)$ for a negative example.
- An example is false negative (FN) for a hypothesis, if the hypothesis says it should be negative but in fact it is positive.
- An example is false positive (FP) for a hypothesis, if the hypothesis says it should be positive but in fact it is negative.
- For false examples (assuming correct): eliminate the hypothesis or change it to accommodate the false example.

*For medical application: Which is more serious?

false negative is serious
the classifier tells you no but this fact is false (that mesns you are sick)

Learning General Logical Descriptions
-Current-best-hypothesis search

- To maintain a single hypothesis and to adjust it as new examples arrive in order to maintain consistency.
- For false negative, the extension of the hypothesis must be increased to include the example, called generalization. (Less restrictive)
- For false positive, the extension of the hypothesis must be decreased to exclude the example, called specialization. (More restrictive)
- Recheck after changes for consistence with other examples, if fail, backtrack

Learning General Logical Descriptions
-Current-best-hypothesis search

+: +ve examples; -: -ve examples ?boundary?

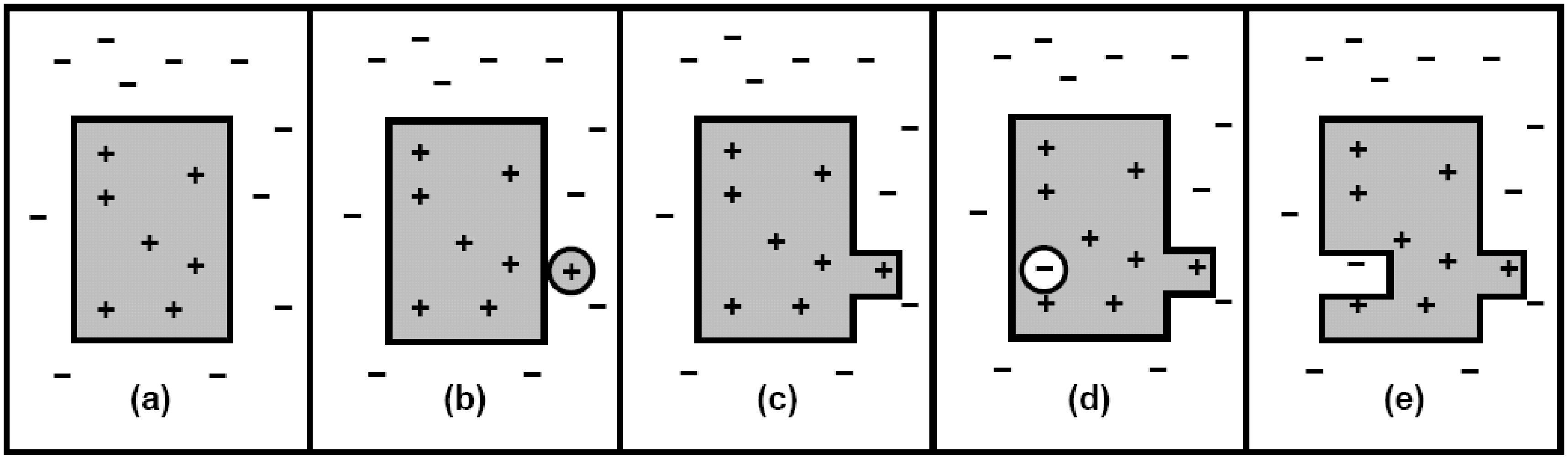


Fig. 18.10 (a) A consistent hypothesis, H (shaded area).
(b) A false negative.
(c) The hypothesis is generalized.
(d) A false positive.
(e) The hypothesis is specialized.

Learning General Logical Descriptions
-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  //all 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
```

Fig 18.11 The current-best-hypothesis learning algorithm.
It searches for a consistent hypothesis and backtracks when no consistent specialization/generalization can be found.

Learning General Logical Descriptions
-Current-best-hypothesis search

- Generalization & specialization are logical relationships between hypotheses. If hypothesis H_1, with definition C_1, is a generalization of H_2, with definition C_2, then we must have:

  $$\forall x\ C_2(x) \Rightarrow C_1(x)$$
  (generalization)

  To generalize H_2, find a definition C_1 that is logically implied by C_2.
- E.g. if $C_2(x)$ is alternate(x) $\wedge$ Patrons(x, Some), then possible generalization: $C_1(x) \equiv$ Patrons(x, Some), called dropping conditions. Or add disjunctive conditions. (?)
- Specialization: add extra conditions to its candidate definition or by removing disjuncts (OR) from a disjunctive definition (?)

Learning General Logical Descriptions
–Some examples from the restaurant example in Fig. 18.5

- Example X_1 is positive. Alternate(X_1) is true, so let us assume an initial hypothesis

$$H_1 : \forall x \; \text{WillWait}(x) \Leftrightarrow \text{Alternate}(x)$$

- X_2 is negative. H_1 predicts it to be positive, so it is a false positive. Therefore, need to specialize H_1. Add an extra condition to rule out X_2. One possibility is

$$H_2 : \forall x \; \text{WillWait}(x) \Leftrightarrow \text{Alternate}(x) \wedge \text{Patrons}(x, \text{Some})$$

(Why not Bar...Hun ?)

Learning General Logical Descriptions
–Some examples from the restaurant example in Fig. 18.5

- X_3 is positive. H_2 predicts it to be negative ⇒ false negative.
  - Therefore, need to generalize H_2.
  - This can be done by dropping the Alternate condition, yielding

    H_3: $\forall x$ WillWait(x) ⇔ Patrons(x, Some)

- X_4 is positive, H_3 predicts it to be negative ⇒ false negative.
  - Therefore need to generalize H_3.
  - We cannot drop Patron condition, because it would yield an all-inclusive hypothesis that is inconsistent with X_2.
  - One possibility is to add a disjunct:

    H_4: $\forall x$ WillWait(x) ⇔ Patrons(x, Some) ∨ (Patrons(x, Full) ∧ Fri/Sat(x))

Why Learning Works: Computational Learning Theory

- Answers provided by **computational learning theory**, the intersection of AI and theoretical computer science.
- The underlying principle: any hypothesis that is seriously wrong will almost certainly be "found out" with high probability after a small number of examples, because it will make an incorrect prediction.
- Thus, any hypothesis that is consistent with a sufficiently large set of training examples m is unlikely to be seriously wrong – i.e., it must be Probably Approximately Correct (PAC).
- We'll prove it and find m.
- PAC-learning is a subfield of computational learning theory.

Why Learning Works: Computational Learning Theory
– How many examples are needed?

- Let X be the set of all possible examples.
- Let D be the distribution from which examples are drawn.
- Let H be the set of possible hypotheses.
- Let m be the number of examples in the training set.

Initially, we'll assume that the true function f is a member of **H**. Now we can define the "**error** of a hypothesis h" with respect to the true function f given a distribution D over the examples as the probability P that h is different from f on an example x:

$$error(h) = \mathrm{P}\,(h(\mathrm{x}) \neq f\,(\mathrm{x}) \underset{\text{(given that)}}{\mid} \mathrm{x}\ \text{drawn from D})$$

Why Learning Works: Computational Learning Theory
– How many examples are needed?

- A hypothesis h is called **approximately correct** if (probability) $error(h) \leq \varepsilon$ (epsilon), where ε is a "small (+ve) constant".
- To show that after seeing m examples, with high probability, all consistent hypotheses will be approximately correct.
- Think of an approximately correct hypothesis as being "close" to the true function in hypothesis space – it lies inside what is called the **ϵ-ball** around the true function f.

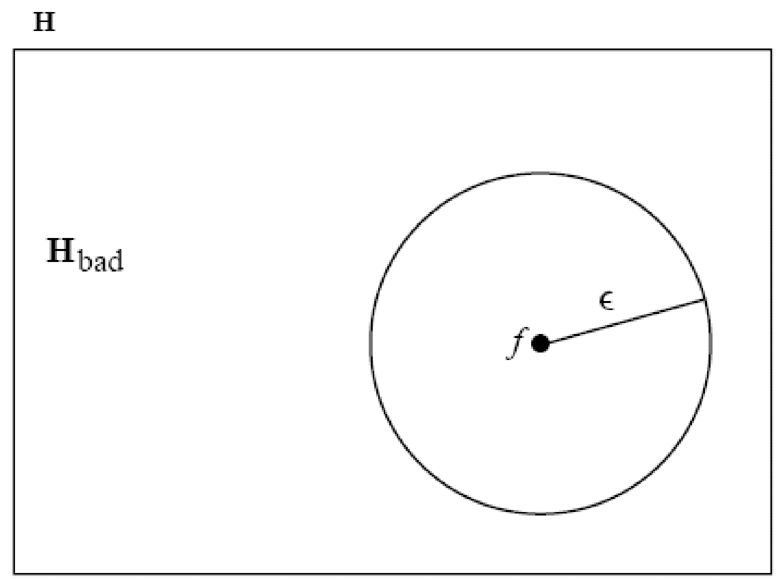


Fig 18.15 Schematic diagram of hypothesis space, showing the ε-ball around the true function f

Why Learning Works: Computational Learning Theory
– How many examples are needed?

- We can calculate the probability that a "seriously wrong" hypothesis $h_b \in H_{bad}$ is consistent with the first m examples as follows:
  - $error(h_b) > \epsilon$. $(error(h) \leq \varepsilon)$
  - Thus, the probability that h_b agrees with any given example is (at most) $\leq (1-\epsilon)$. $(> (1-\epsilon)$ for h)
  - The bound for m examples is

$$P(h_b \text{ agrees with m examples}) \leq (1-\epsilon)^m$$

- For $\mathbf{H_{bad}}$ to contain a consistent hypothesis, at least one of the hypotheses in $\mathbf{H_{bad}}$ must be consistent. The probability of this occurring is bounded by the sum of the individual probabilities:

$$P(\mathbf{H_{bad}} \text{ contains a consistent hypothesis}) \leq |\mathbf{H_{bad}}|(1-\epsilon)^m \leq |\mathbf{H}|(1-\epsilon)^m$$

where |H| is the total hypothesis space; |*|: size;

  - e.g. Boolean Function of n attributes: $|H| = 2^{2^n}$

Why Learning Works: Computational Learning Theory
– How many examples are needed?

- We would like to reduce the probability of this event below some small number δ:

$$|H|(1-\epsilon)^m \leq \delta$$

- We can achieve this if we allow the algorithm to see

$$m \geq \frac{1}{\varepsilon}\left(\ln\frac{1}{\delta} + \ln|H|\right)$$

- Examples needed: (trend only)
  - –Thus, if a hypothesis is consistent with m *examples*, then with probability at least $1-\delta$, it has error at most ϵ. In other words, it is probably approximately correct (*PAC*).
  - –The number, m, of required examples, as a function of ϵ and δ, is called the **sample complexity** of the hypothesis space.
  - –E.g. If $|H| = 2^{2^n}$ for Boolean functions, sample complexity, m, grows with 2^n. n: # of attributes
  - –Smaller ϵ, δ and larger H apace $\Rightarrow$ higher m required

Sample Complexity:
$|H| = 2^{2^n}$ for Boolean functions

Consider n=2: $2^{2^2} = 16$ possible binary functions (**outputs**)

n = 2 attributes

16 possible functions defined by 16 different outputs

x_1	x_2	h_1	h_2	h_3	h_4	h_5	h_6	h_7	h_8	h_9	h_{10}	h_{11}	h_{12}	h_{13}	h_{14}	h_{15}	h_{16}
0	0	0	0	0	0	0	0	0	0	1	1	1	1	1	1	1	1
0	1	0	0	0	0	1	1	1	1	0	0	0	0	1	1	1	1
1	0	0	0	1	1	0	0	1	1	0	0	1	1	0	0	1	1
1	1	0	1	0	1	0	1	0	1	0	1	0	1	0	1	0	1