Foundations of AI

11. Machine Learning

Learning from Observations

Wolfram Burgard, Andreas Karwath, Bernhard Nebel, and Martin Riedmiller

Learning

- What is learning?
  An agent learns when it improves its performance w.r.t. a specific task with experience.
  → E.g., game programs
- Why learn?
  → Engineering, philosophy, cognitive science
  → Data Mining (discovery of new knowledge through data analysis)

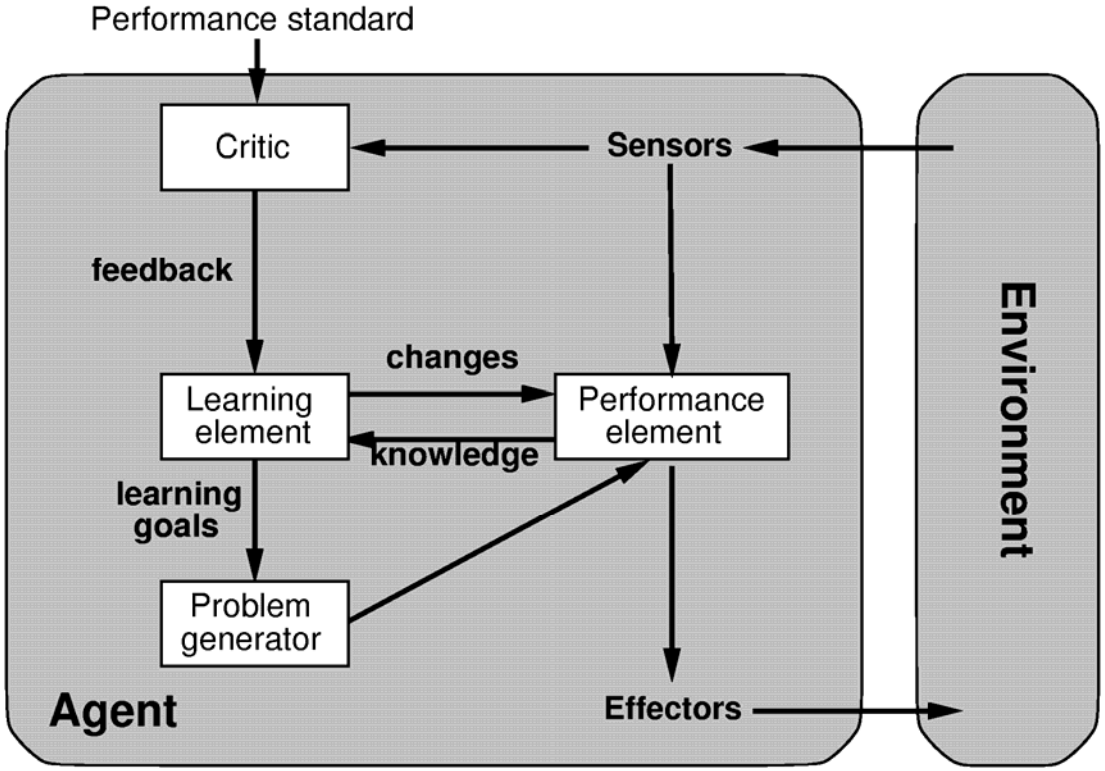
No intelligence without learning!

Contents

- The learning agent
- Types of learning
- Decision trees

The Learning Agent

So far an agent's percepts have only served to help the agent choose its actions. Now they will also serve to improve future behavior.



Building Blocks of the Learning Agent

Performance element: Processes percepts and chooses actions.
→ Corresponds to the agent model we have studied so far.

Learning element: Carries out improvements
→ requires self knowledge and feedback on how the agent is doing in the environment.

Critic: Evaluation of the agent's behaviour based on a given external behavioral measure
→ feedback.

Problem generator: Suggests explorative actions that lead the agent to new experiences.

The Learning Element

Its design is affected by four major issues:

- Which components of the performance element are to be learned?
- What representation should be chosen?
- What form of feedback is available?
- Which prior information is available?

Types of Feedback During Learning

The type of feedback available for learning is usually the most important factor in determining the nature of the learning problem.

Supervised learning: Involves learning a function from examples of its inputs and outputs.

Unsupervised learning: The agent has to learn patterns in the input when no specific output values are given.

Reinforcement learning: The most general form of learning in which the agent is not told what to do by a teacher. Rather it must learn from a reinforcement or reward. It typically involves learning how the environment works.

Inductive Learning

An example is a pair (x, f(x)). The complete set of examples is called the training set.

Pure inductive inference: for a collection of examples for f, return a function h (hypothesis) that approximates f.

The function h typically is member of a hypothesis space H.

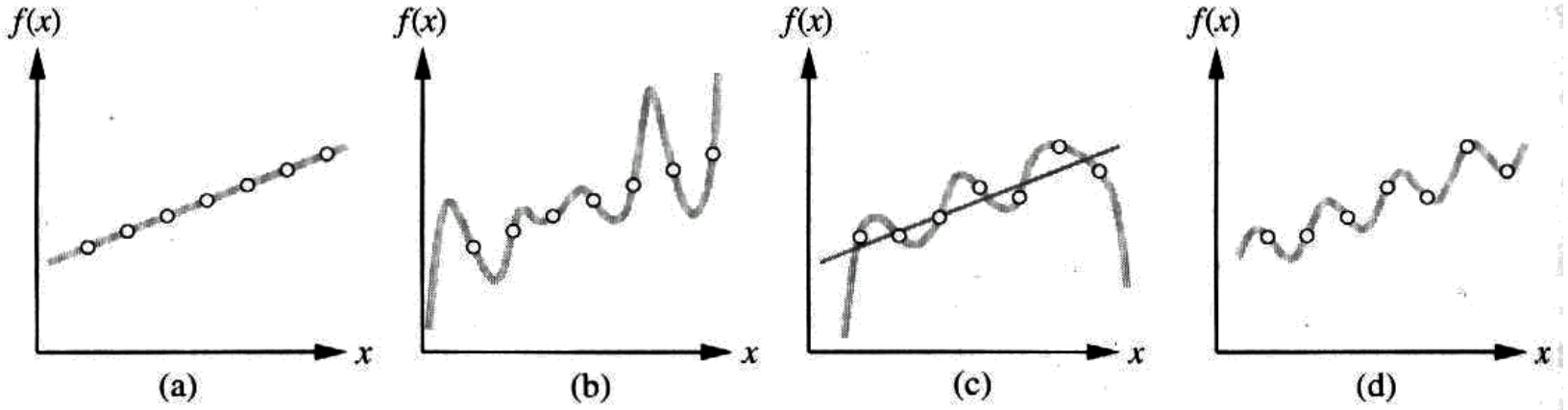
A good hypothesis should generalize the data well, i. e., will predict unseen examples correctly.

A hypothesis is consistent with the data set if it agrees with all the data.

How do we choose from among multiple consistent hypotheses?

Ockham's razor: prefer the simplest hypothesis consistent with the data.

Example: Fitting a Function to a Data Set



a) consistent hypothesis that agrees with all the data

b) degree-7 polynomial that is also consistent with the data set

c) data set that can be approximated consistently with a degree-6 polynomial

d) sinusoidal exact fit to the same data

Decision Trees

Input: Description of an object or a situation through a set of attributes.

Output: a decision, that is the predicted output value for the input.

Both, input and output can be discrete or continuous.

Discrete-valued functions lead to classification problems.

Learning a continuous function is called regression.

Boolean Decision Tree

Input: set of vectors of input attributes X and a single Boolean output value y (goal predicate).

Output: Yes/No decision based on a goal predicate.

Goal of the learning process: Definition of the goal predicate in the form of a decision tree.

Boolean decision trees represent Boolean functions.

Properties of (Boolean) Decision Trees:

- An internal node of the decision tree represents a test of a property.
- Branches are labeled with the possible values of the test.
- Each leaf node specifies the Boolean value to be returned if that leaf is reached.

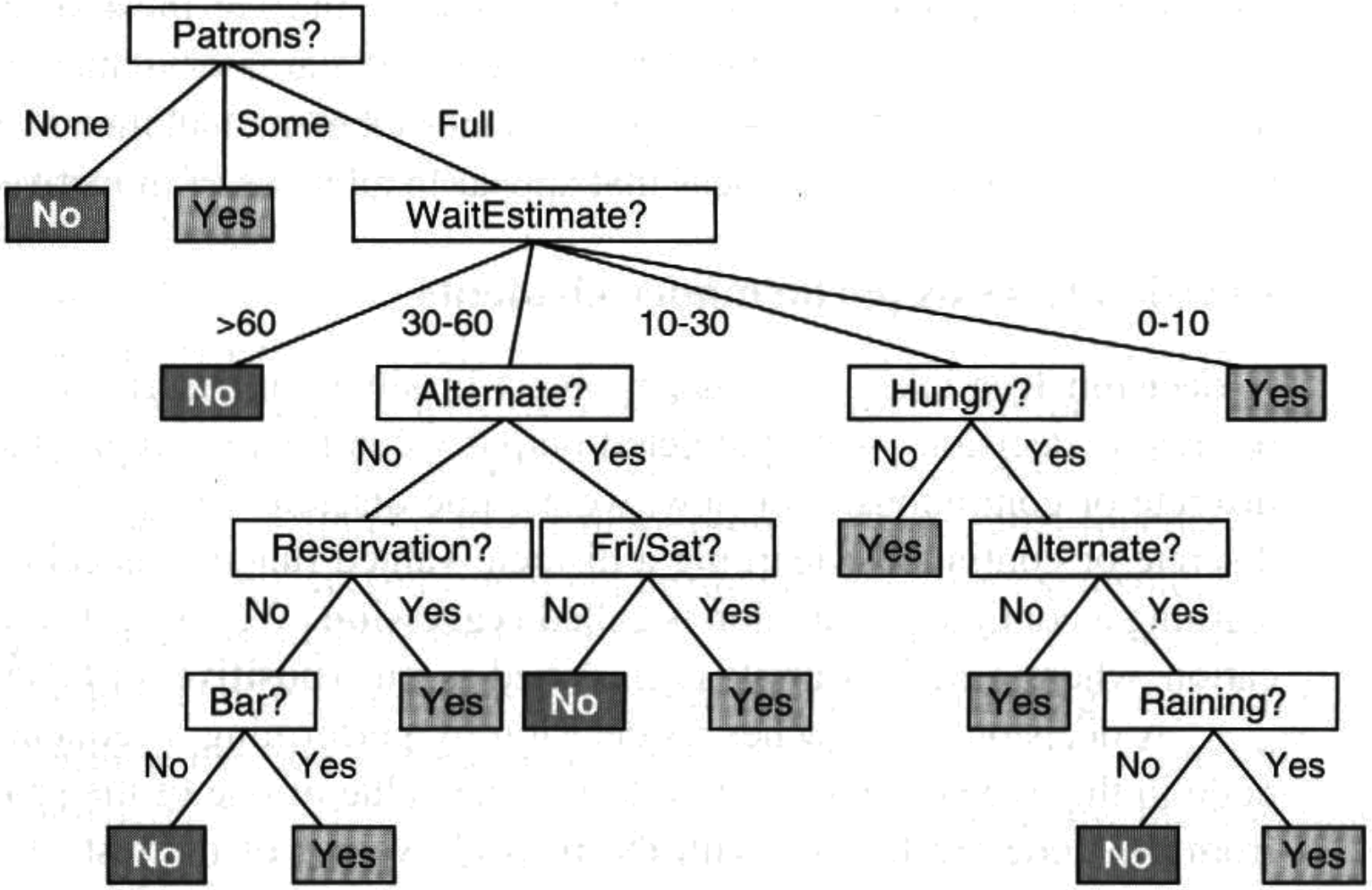
When to Wait for Available Seats at a Restaurant

Goal predicate: *WillWait*

Test predicates:

- *Patrons*: How many guests are there? (none, some, full)
- *WaitEstimate*: How long do we have to wait? (0-10, 10-30, 30-60, >60)
- *Alternate*: Is there an alternative? (T/F)
- *Hungry*: Am I hungry? (T/F)
- *Reservation*: Have I made a reservation? (T/F)
- *Bar*: Does the restaurant have a bar to wait in? (T/F)
- *Fri/Sat*: Is it Friday or Saturday? (T/F)
- *Raining*: Is it raining outside? (T/F)
- *Price*: How expensive is the food? ($, $$, $$$)
- *Type*: What kind of restaurant is it? (French, Italian, Thai, Burger)

Restaurant Example (Decision Tree)



Expressiveness of Decision Trees

Each decision tree hypothesis for the WillWait goal predicate can bee seen as an assertion of the form

$$\forall s \, WillWait(s) \Leftrightarrow (P_1(s) \vee P_2(s) \vee \ldots \vee P_n(s))$$

where each $P_i(s)$ is the conjunction of tests along a path from the root of the tree to a leaf with a positive outcome.

Any Boolean function can be represented by a decision tree.

Limitation: All tests always involve only one object and the language of traditional decision trees is inherently propositional.

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

cannot be represented as a test.

We could always add another test called CheaperRestaurantNearby, but a decision tree with all such attributes would grow exponentially.

Compact Representations

For every Boolean function we can construct a decision tree by translating every row of a truth table to a path in the tree.

This can lead to a tree whose size is exponential in the number of attributes.

Although decision trees can represent functions with smaller trees, there are functions that require an exponentially large decision tree:

Parity function: $p(x) = \begin{cases} 1 & \text{even number of inputs are 1} \\ 0 & \text{otherwise} \end{cases}$

Majority function: $m(x) = \begin{cases} 1 & \text{half of the inputs are 1} \\ 0 & \text{otherwise} \end{cases}$

There is no consistent representation that is compact for all possible Boolean functions.

The Training Set of the Restaurant Example

Classification of an example = Value of the goal predicate

TRUE → positive example

FALSE → negative example

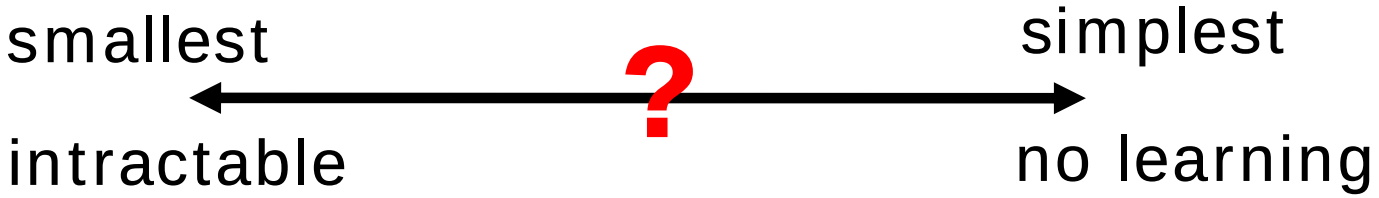
Example	Attributes										Goal
	Alt	*Bar*	*Fri*	*Hun*	*Pat*	*Price*	*Rain*	*Res*	*Type*	*Est*	*WillWait*
X_1	*Yes*	*No*	*No*	*Yes*	*Some*	*$$$*	*No*	*Yes*	*French*	*0–10*	*Yes*
X_2	*Yes*	*No*	*No*	*Yes*	*Full*	*$*	*No*	*No*	*Thai*	*30–60*	*No*
X_3	*No*	*Yes*	*No*	*No*	*Some*	*$*	*No*	*No*	*Burger*	*0–10*	*Yes*
X_4	*Yes*	*No*	*Yes*	*Yes*	*Full*	*$*	*No*	*No*	*Thai*	*10–30*	*Yes*
X_5	*Yes*	*No*	*Yes*	*No*	*Full*	*$$$*	*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*

Inducing Decision Trees from Examples

- Naïve solution: we simply construct a tree with one path to a leaf for each example.
- In this case we test all the attributes along the path and attach the classification of the example to the leaf.
- Whereas the resulting tree will correctly classify all given examples, it will not say much about other cases.
- It just memorizes the observations and does not generalize.

Inducing Decision Trees from Examples

- Smallest solution: applying Ockham's razor we should instead find the smallest decision tree that is consistent with the training set.
- Unfortunately, for any reasonable definition of smallest finding the smallest tree is intractable.
- Dilemma:



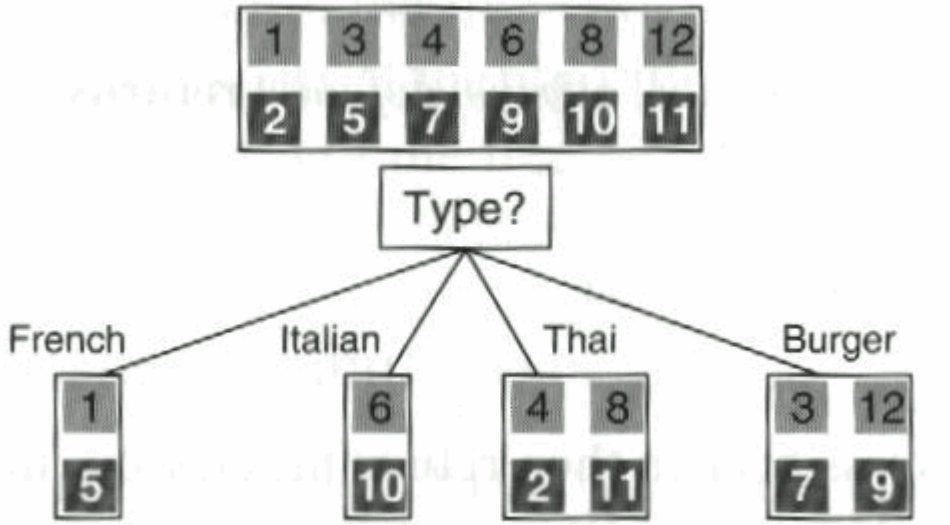
- We can give a decision tree learning algorithm that generates "smallish" trees.

Idea of Decision Tree Learning

Divide and Conquer approach:

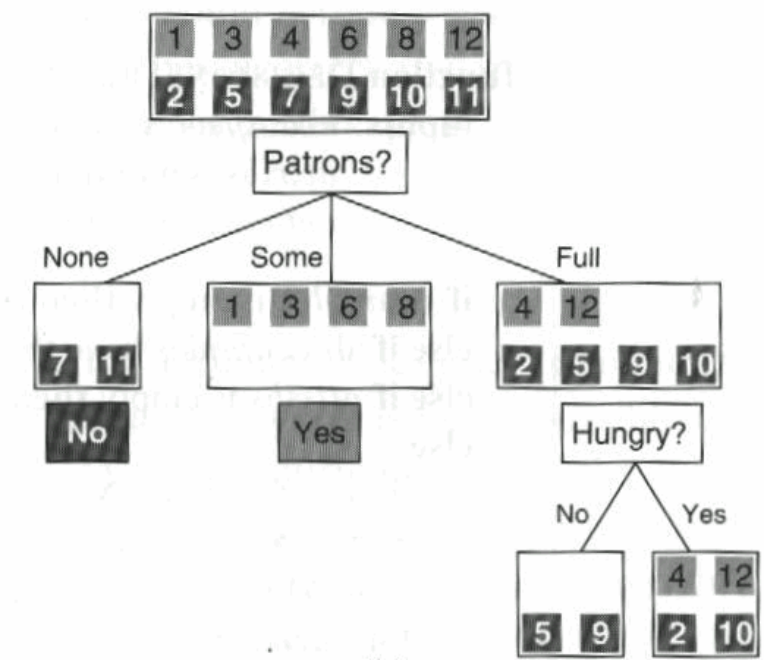
- Choose an (or better: the best) attribute.
- Split the training set into subsets each corresponding to a particular value of that attribute.
- Now that we have divided the training set into several smaller training sets, we can recursively apply this process to the smaller training sets.

Splitting Examples (1)



- Type is a poor attribute, since it leaves us with four subsets each of them containing the same number of positive and negative examples.
- It does not reduce the problem complexity.

Splitting Examples (2)



- Patrons is a better choice, since if the value is None or Some, then we are left with example sets for which we can answer definitely (Yes or No).
- Only for the value Full we are left with am mixed set of examples.
- One potential next choice is Hungry.

Recursive Learning Process

In each recursive step there are four cases to consider:

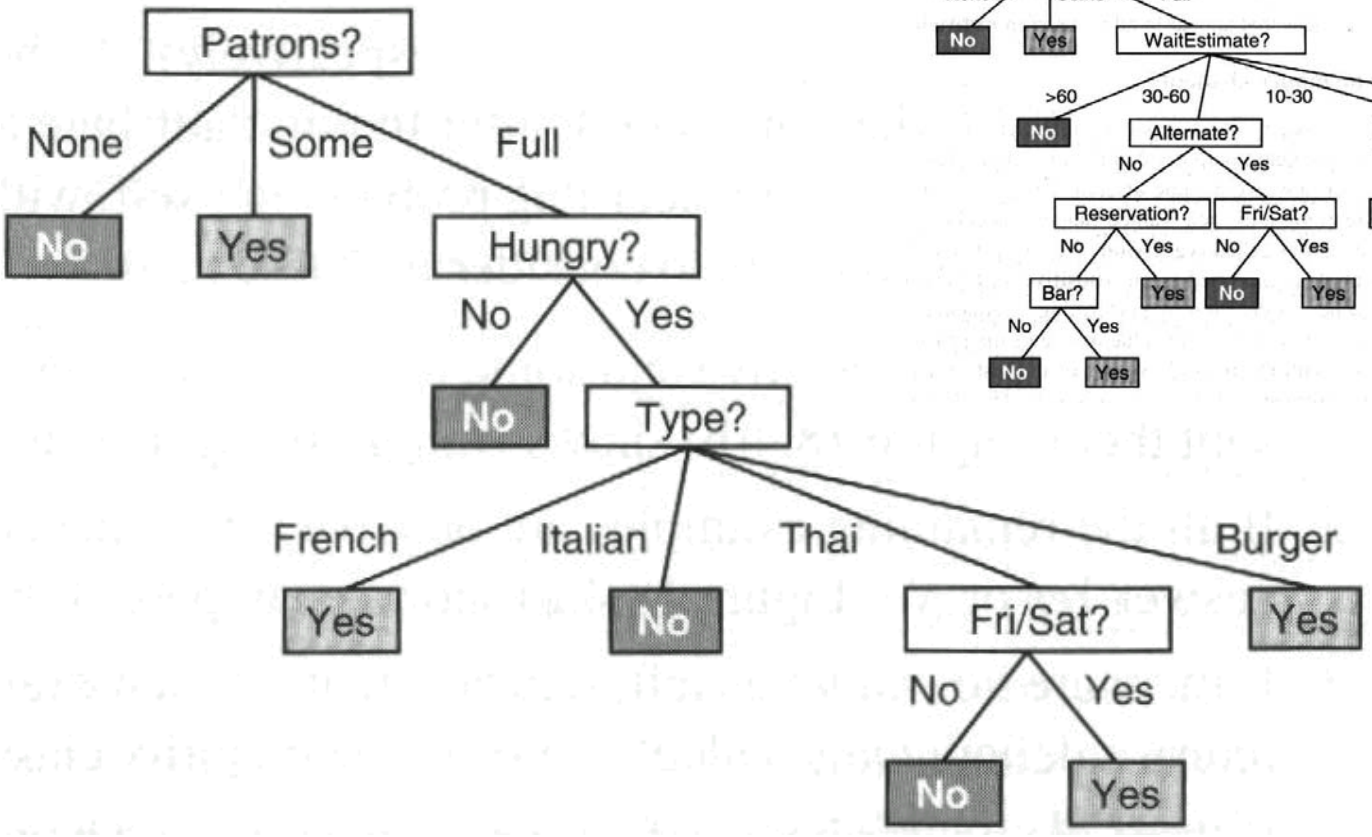
- Positive and negative examples: choose a new attribute.
- Only positive (or only negative) examples: done (answer is Yes or No).
- No examples: there was no example with the desired property. Answer Yes if the majority of the parent node's examples is positive, otherwise No.
- No attributes left, but there are still examples with different classifications: there were errors in the data (→ NOISE) or the attributes do not give sufficient information. Answer Yes if the majority of examples is positive, otherwise no.

The Decision Tree Learning Algorithm

```
function DECISION-TREE-LEARNING(examples, attribs, default) returns a decision tree
  inputs: examples, set of examples
          attribs, 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 attribs is empty then return MAJORITY-VALUE(examples)
  else
      best ← CHOOSE-ATTRIBUTE(attribs, examples)
      tree ← a new decision tree with root test best
      m ← MAJORITY-VALUE(examples_i)
      for each value v_i of best do
          examples_i ← {elements of examples with best = v_i}
          subtree ← DECISION-TREE-LEARNING(examples_i, attribs − best, m)
          add a branch to tree with label v_i and subtree subtree
      return tree
```

Application to the Restaurant Data



Original tree:

Properties of the Resulting Tree

- The resulting tree is considerably simpler than the one originally given (and from which the training examples were generated).
- The learning algorithm outputs a tree that is consistent with all examples it has seen.
- The tree does not need to agree with the correct function.
- For example, it suggests not to wait if we are not hungry. If we are, there are cases in which it tells us to wait.
- Some tests (Raining, Reservation) are not included since the algorithm can classify the examples without them.

Choosing Attribute Tests

`choose-attribute(attribs, examples)`

- One goal of decision tree learning is to select attributes that minimize the depth of the final tree.
- The perfect attribute divides the examples into sets that are all positive or all negative.
- Patrons is not perfect but fairly good.
- Type is useless since the resulting proportion of positive and negative examples in the resulting sets are the same as in the original set.
- What is a formal measure of “fairly good” and “useless?”

Evaluation of Attributes

Tossing a coin: What value has prior information about the outcome of the toss when the stakes are $1 and the winnings $1?

- Rigged coin with 99% heads and 1% tails. (average winnings per toss = $0.98)
  - → Worth of information about the outcome is less than $0.02.
- Fair coin
  - → Value of information about the outcome is less than $1.
  - → The less we know about the outcome, the more valuable the prior information.

Information Provided by an Attribute

- One suitable measure is the expected amount of information provided by the attribute.
- Information theory measures information content in bits.
- One bit is enough to answer a yes/no question about which one has no idea (fair coin flip).
- In general, if the possible answers v_i have probabilities $P(v_i)$, the information content is given as

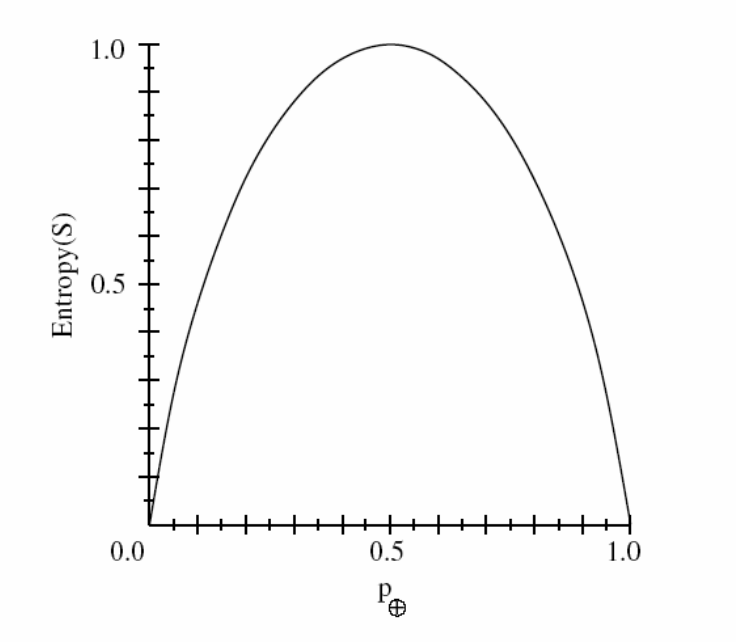
$$I(P(v_1),..,P(v_n)) = \sum_{i=1}^{n} -P(v_i)\log_2(P(v_i))$$

Examples

$$I\left(\frac{1}{2},\frac{1}{2}\right)=$$

$$I(1,0)=$$

$$I(0,1)=$$



Attribute Selection (1)

Suppose training set E consists of p positive and n negative examples:

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

The value of an attribute A depends on the additional information that we still need to collect after we selected it.

Suppose A divides the training set E into subsets E_i, $i = 1,\ldots, v$.

Every subset has $I\left(\frac{p_i}{p_i+n_i},\frac{n_i}{p_i+n_i}\right)$

A random example has value i with probability $\frac{p_i+n_i}{p+n}$.

Attribute Selection (2)

→ The average information content after choosing A is

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

→ The information gain from choosing A is

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

Heuristic in `choose-attribute` is to select the attribute with the larges gain.

Examples:

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

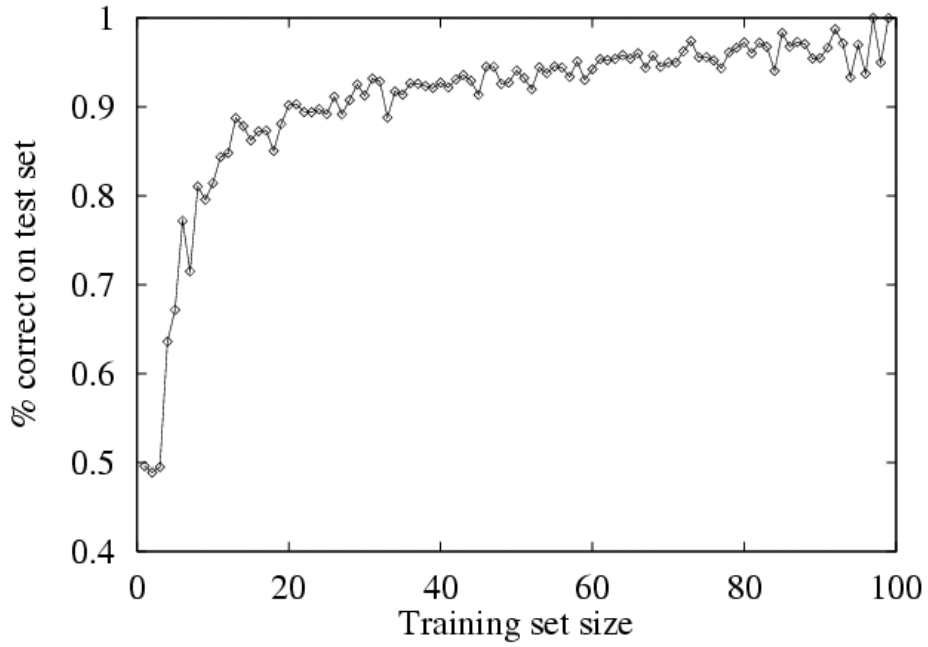
$$Gain(Type)=1-\left[\frac{2}{12}I\left(\frac{1}{2},\frac{1}{2}\right)+\frac{2}{12}I\left(\frac{1}{2},\frac{1}{2}\right)+\frac{4}{12}I\left(\frac{2}{4},\frac{2}{4}\right)+\frac{4}{12}I\left(\frac{2}{4},\frac{2}{4}\right)\right]=0$$

Assessing the Performance of the Learning Algorithm

Methodology for assessing the power of prediction:

- Collect a large number of examples.
- Divide it into two disjoint sets: the training set and the test set.
- Use the training set to generate *h*.
- Measure the percentage of examples of the test set that are correctly classified by *h*.
- Repeat the process for randomly-selected training sets of different sizes.

Learning Curve for the Restaurant Example



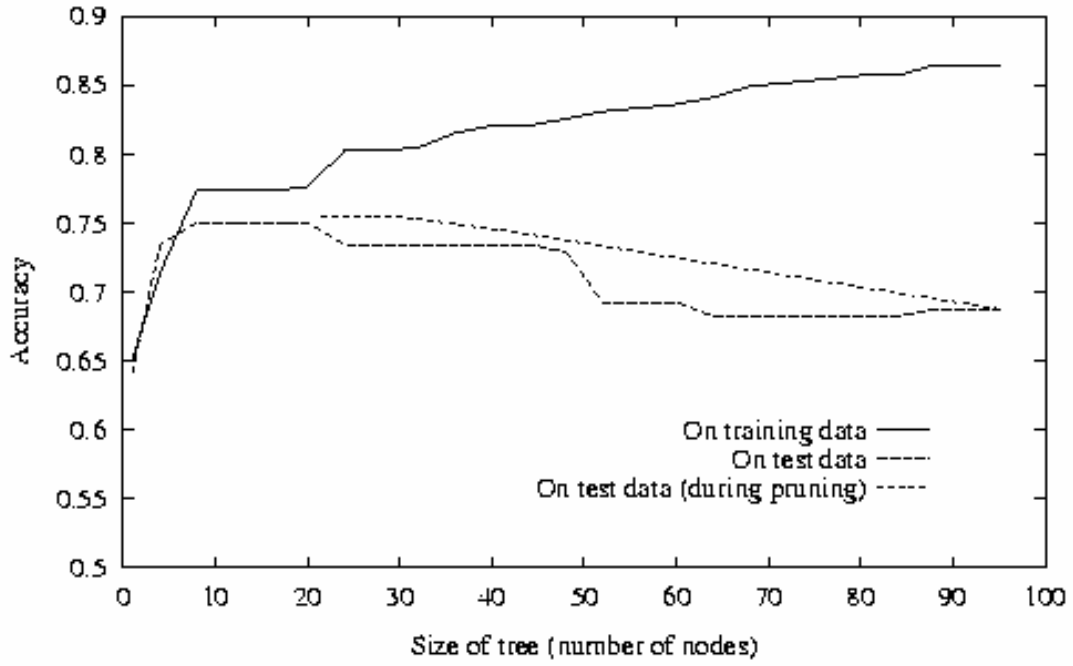
As the training set grows, the prediction quality increases.

Important Strategy for Designing Learning Algorithms

- The training and test sets must be kept separate.
- Common error: Changing the algorithm after running a test, and then testing it with training and test sets from the same basic set of examples. By doing this, knowledge about the test set gets stored in the algorithm, and the training and test sets are no longer independent.

Noise

- What is noise? Random errors in the learning data
- Effect: Larger trees make more mistakes on new data (overfitting)
- Avoidance of overfitting by means of a "validation set": The training set is divided into two groups; 70% of the training set is used to build the tree, and the remaining 30% to define an appropriate tree size ("pruning")



Post-pruning

- One way to deal with noise
  - Split training set into e.g.
    - 70 % for learning
    - 30 % for validation
  - First build the tree as usual on learning set
  - Then iterate as long as accuracy on validation set increases
    - Turn each subtree into leaf and measure accuracy on validation set
    - Select that new tree that increases the accuracy the most on the validation set

Summary: Decision Trees

- One possibility for representing (Boolean) functions.
- Decision trees can be exponential in the number of attributes.
- It is often too difficult to find the minimal DT.
- One method for generating DTs that are as flat as possible is based on ranking the attributes.
- The ranks are computed based on the information gain.