Supervised Learning

Most forms of machine learning can be viewed as *function approximation*:

Given: A set of **training examples**: $\{(x_1, f(x_1)), \ldots, (x_m, f(x_m))\}$

A space of **hypotheses** H

Find: An hypothesis $h \in H$ that is a good approximation to f.

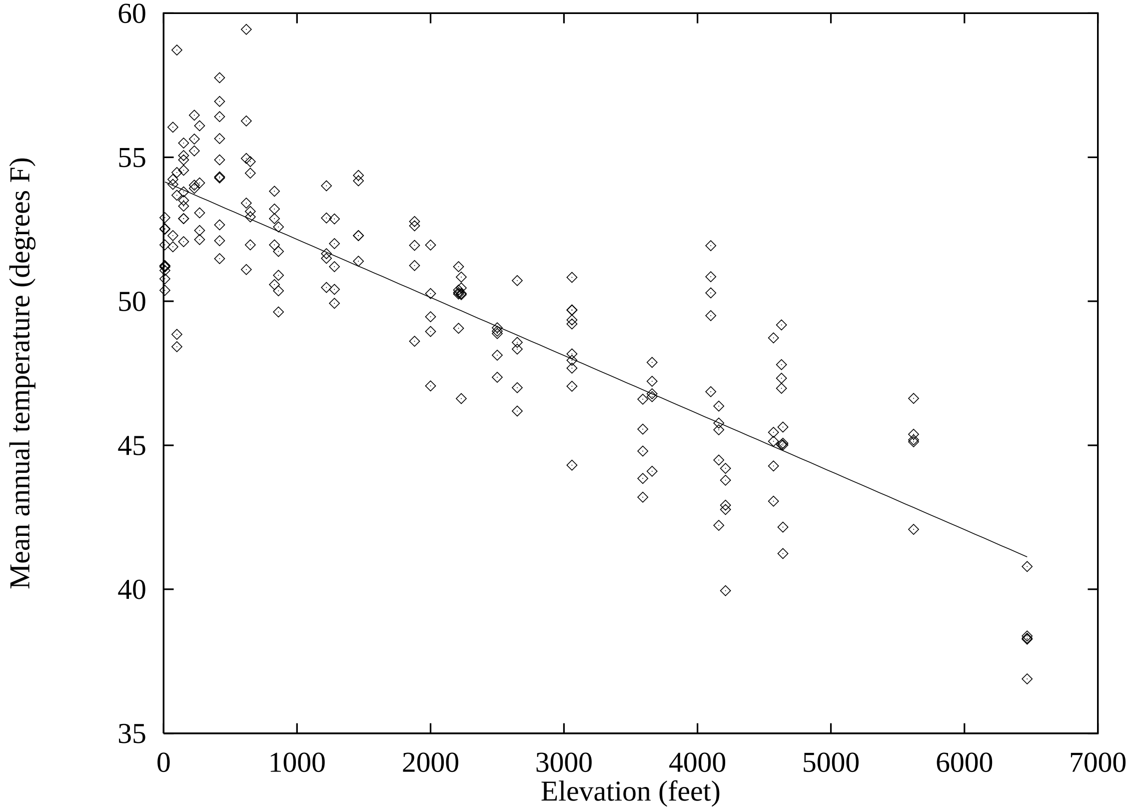
Each example x_i is typically represented by a vector of **feature values**:

$$x_i = (x_{i1}, \ldots, x_{in}).$$

Kinds of Functions

- **Real-valued Functions**. $f(x) \in \mathcal{R}$.
- **Classifiers**. $f(x) \in \{c_1, \ldots, c_k\}$.
- **Probability Distributions**. $P_f(y|x) = P(f(x)|x)$.

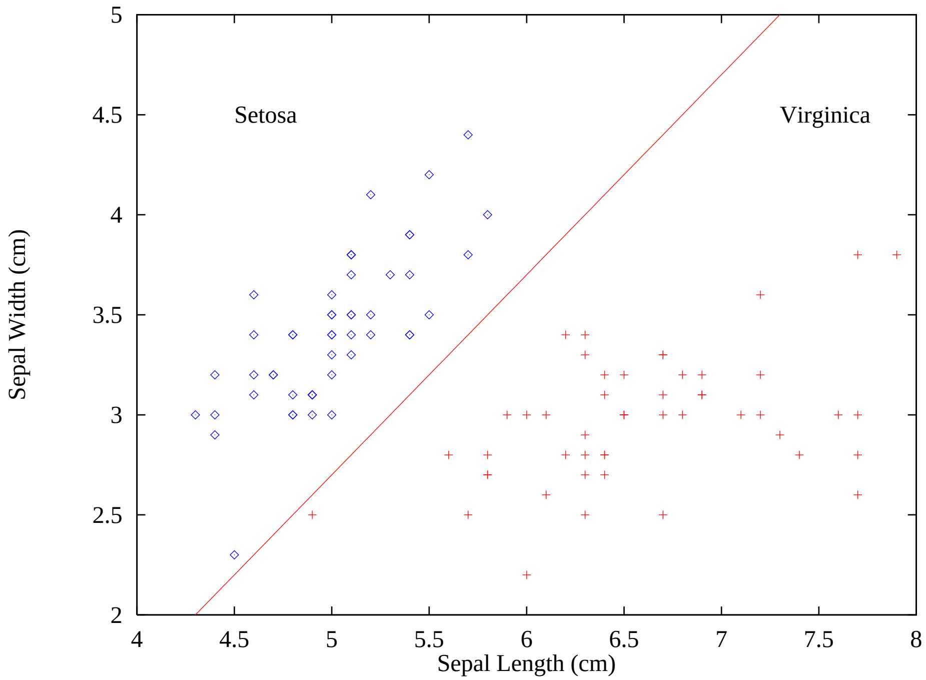
Real-Valued Functions: Regression



- **Unknown function:** maps elevation to mean annual temperature.
- **Examples:** 175 weather stations with known elevation and temperature.
- **Hypothesis space:** All functions of the form

$$temp = w_0 + w_1 \cdot elev.$$

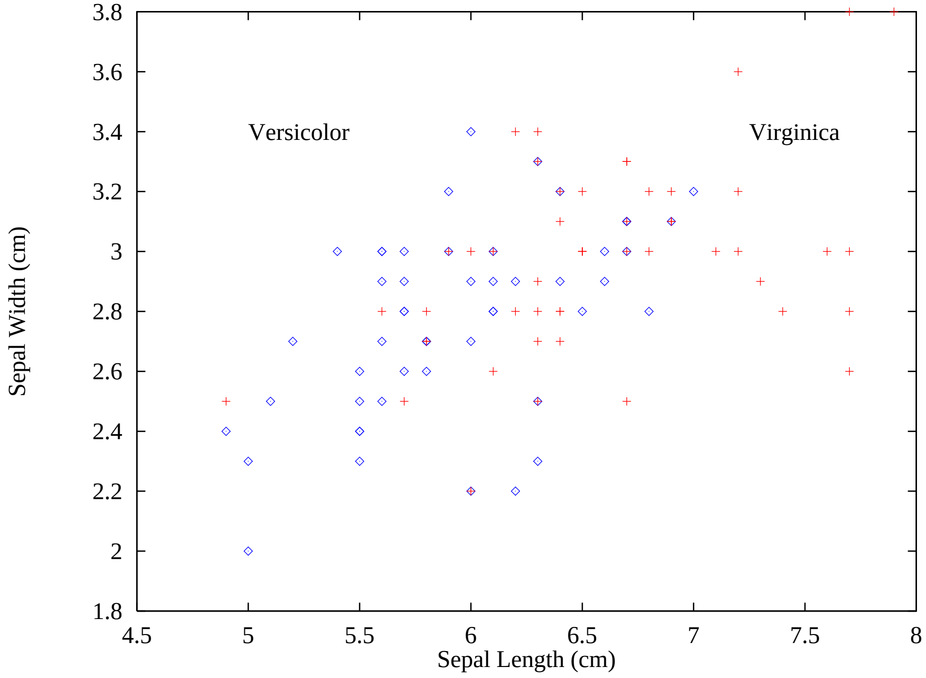
Discrete-Valued Functions: Classification



- **Unknown function:** maps from flower measurements to species of flower
- **Examples:** 100 flowers measured and classified by R. A. Fisher.
- **Hypothesis Space:** All linear discriminators of the form

$$h(x) = \begin{cases} \mathit{Setosa} & \text{if } w_0 + w_1 \cdot x.\textsc{SepalWidth} + w_2 \cdot x.\textsc{SepalLength} > 0 \\ \mathit{Virginica} & \text{otherwise} \end{cases}$$

Probability Distributions

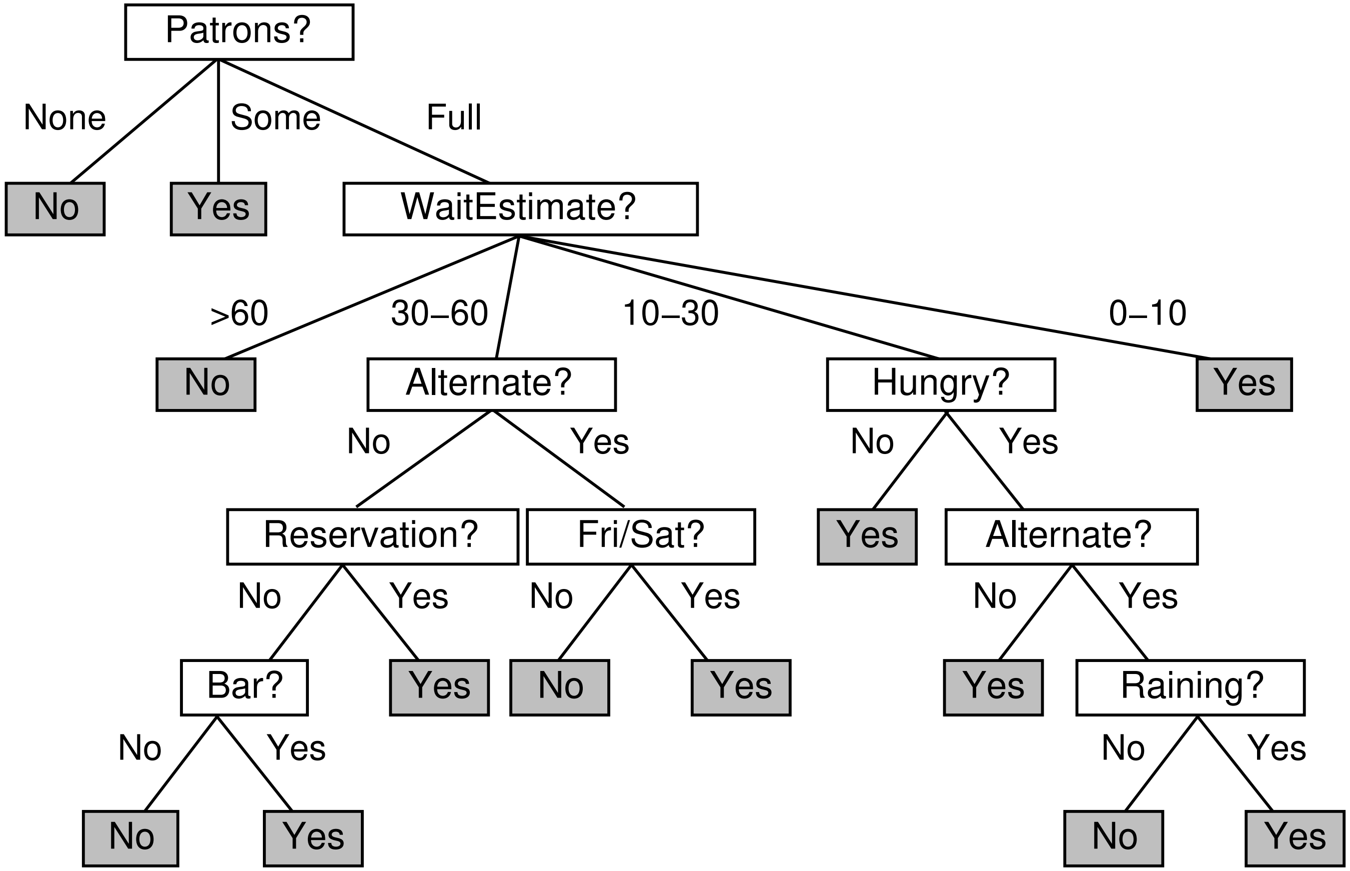


- **Unknown function:** maps from flower measurements to probability of Versicolor or Virginica.
- **Examples:** 100 flowers measured and classified by R. A. Fisher.
- **Hypothesis Space:** Normal (gaussian) distributions with mean μ_i and covariance matrix Σ_i.

Decision Trees

Decision trees are the most widely-used representations for unknown functions in classification problems.

Example Decision Tree: "Waiting for a Restaurant"



Waiting for a Restaurant: Training Examples

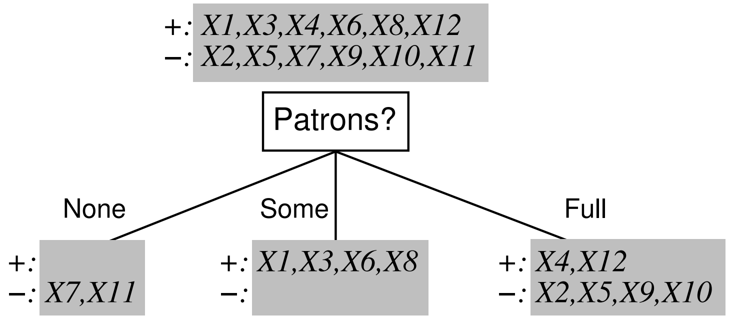
Example	Attributes										Goal
	Alt	*Bar*	*Fri*	*Hun*	*Pat*	*Price*	*Rain*	*Res*	*Type*	*Est*	*WillWait*
X_1	*Yes*	*No*	*No*	*Yes*	*Some*	*$$$*	*No*	*Yes*	*French*	*0–10*	*Yes*
X_2	*Yes*	*No*	*No*	*Yes*	*Full*	*$*	*No*	*No*	*Thai*	*30–60*	*No*
X_3	*No*	*Yes*	*No*	*No*	*Some*	*$*	*No*	*No*	*Burger*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

Learning a Decision Tree

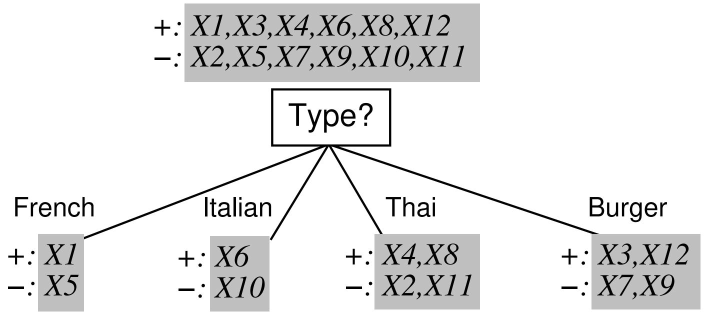
```
function DECISION-TREE-LEARNING(examples, attributes, default) returns a decision tree
   inputs: examples, set of examples
           attributes, set of attributes
           default, default value for the goal predicate

   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
```

Learning a Decision Tree (2)

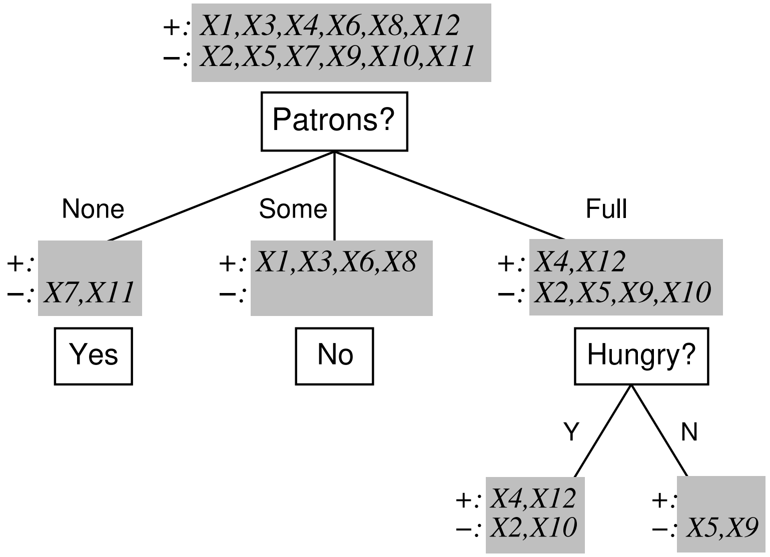
(a)



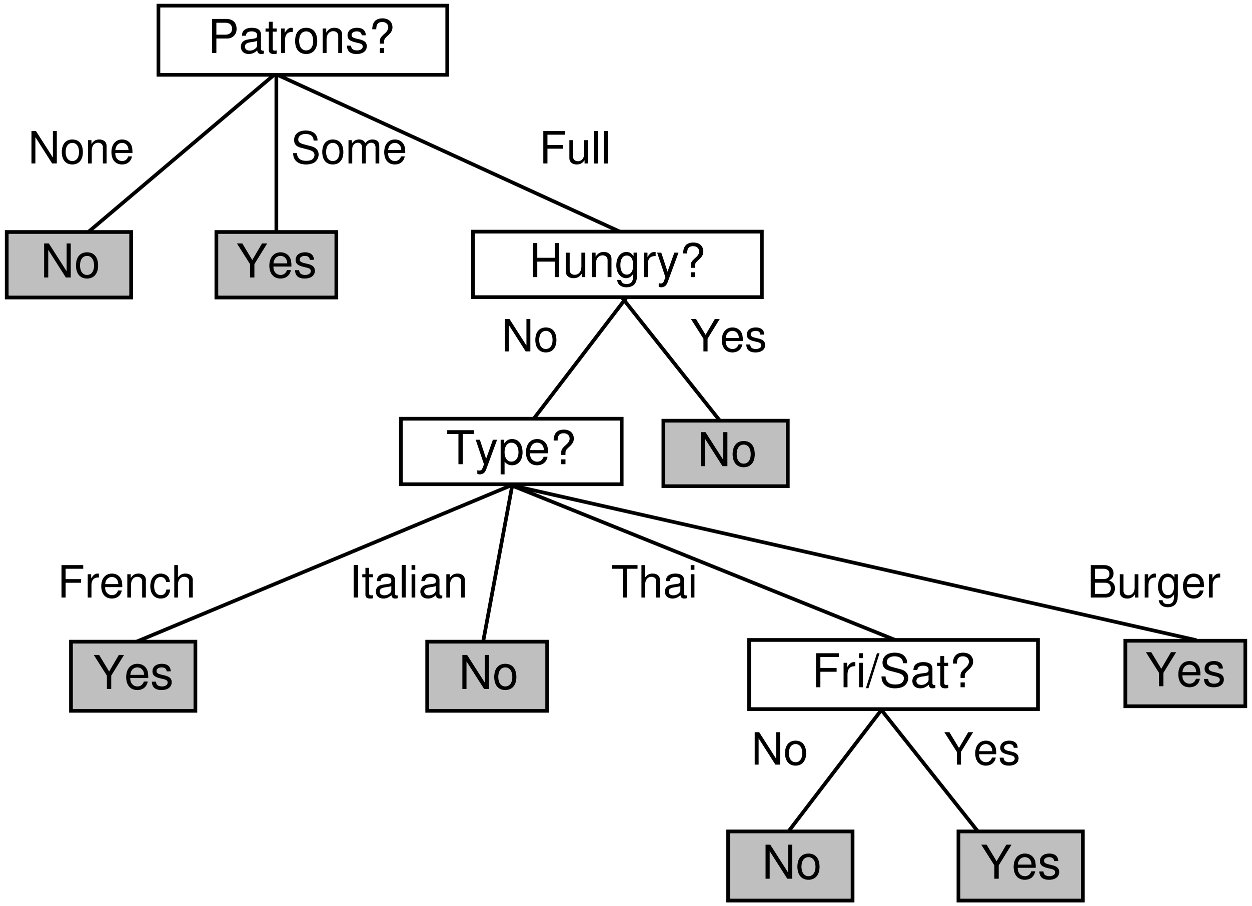
(b)



(c)



Learning a Decision Tree (3)



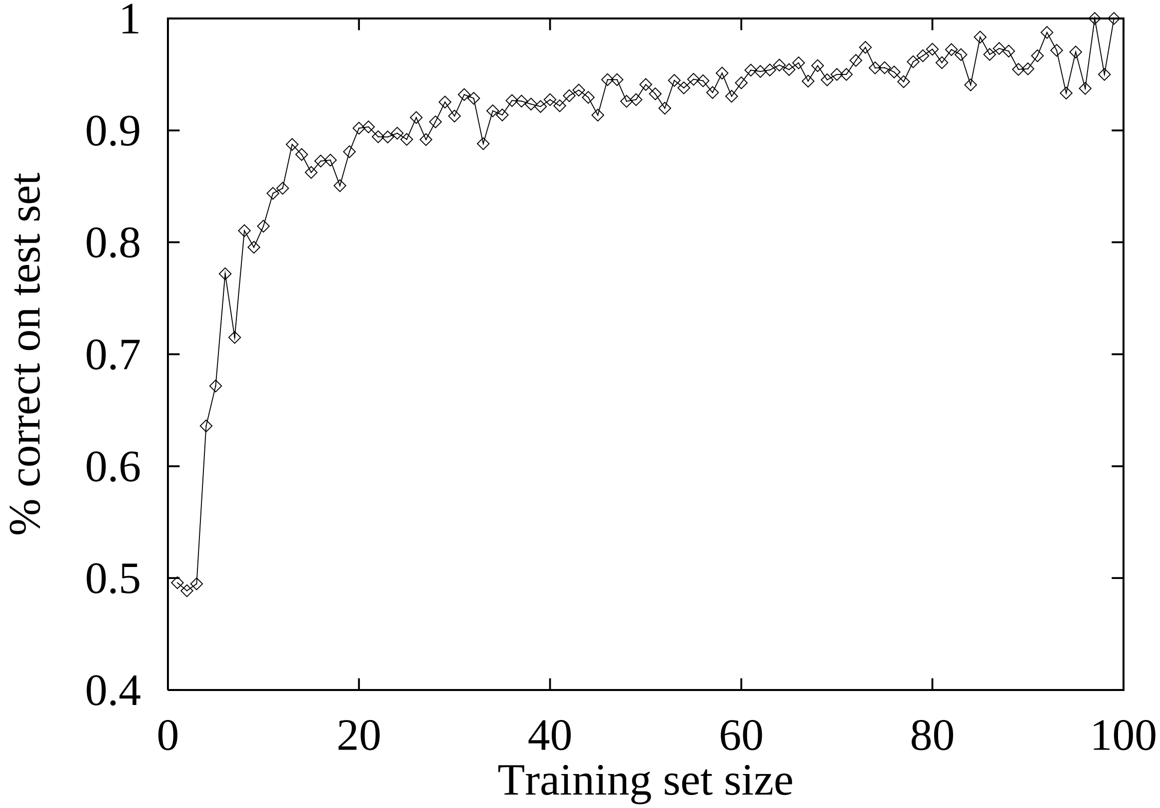
Assessing the Performance of a Learning Algorithm

- Collect a large set of examples.
- Randomly divide it into two disjoint sets: the **training set** and the **test set**.
- Run the learning algorithm on the training set to generate hypothesis h.
- Measure the percentage of examples in the test set that are correctly classified by h.

If we just measured performance on the training set, it would be too optimistic! The test set must be independent!

Learning Curves

Accuracy usually improves with more training examples.



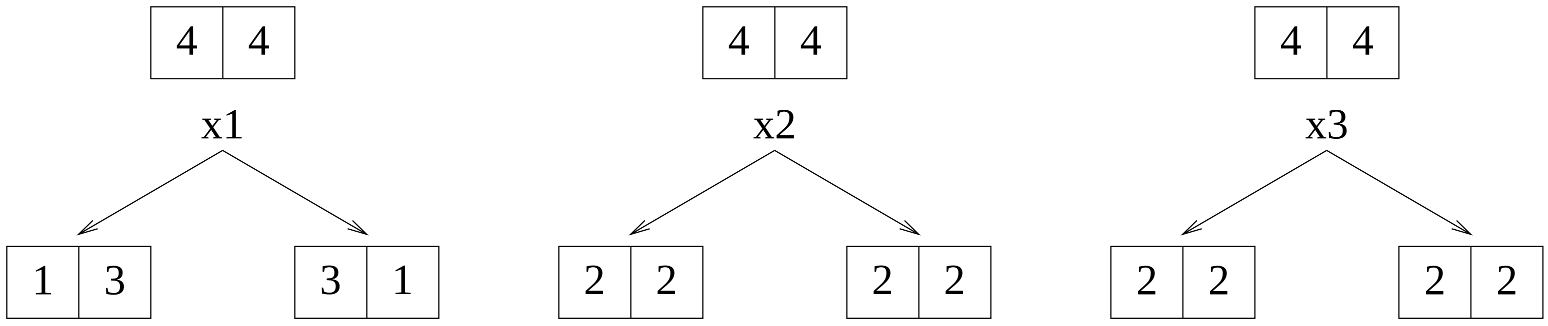
Choosing the Best Splitting Test

One way to choose the best feature is to perform a 1-step lookahead search and choose the feature that gives the lowest error rate on the training data. Let J_j be the number of errors on the training set after splitting on boolean feature j.

ChooseBestFeature(training set S)
choose j to minimize J_j, computed as follows:
    $S_0 =$ all $\langle \mathbf{x}, y \rangle \in S$ with $x_j = 0$;
    $S_1 =$ all $\langle \mathbf{x}, y \rangle \in S$ with $x_j = 1$;
    $y_0 =$ the most common value of y in S_0
    $y_1 =$ the most common value of y in S_1
    $J_0 =$ number of examples $\langle \mathbf{x}, y \rangle \in S_0$ with $y \neq y_0$
    $J_1 =$ number of examples $\langle \mathbf{x}, y \rangle \in S_1$ with $y \neq y_1$
    $J_j = J_0 + J_1$ (total errors if we split on this feature)
return j

Choosing the Best Feature—An Example

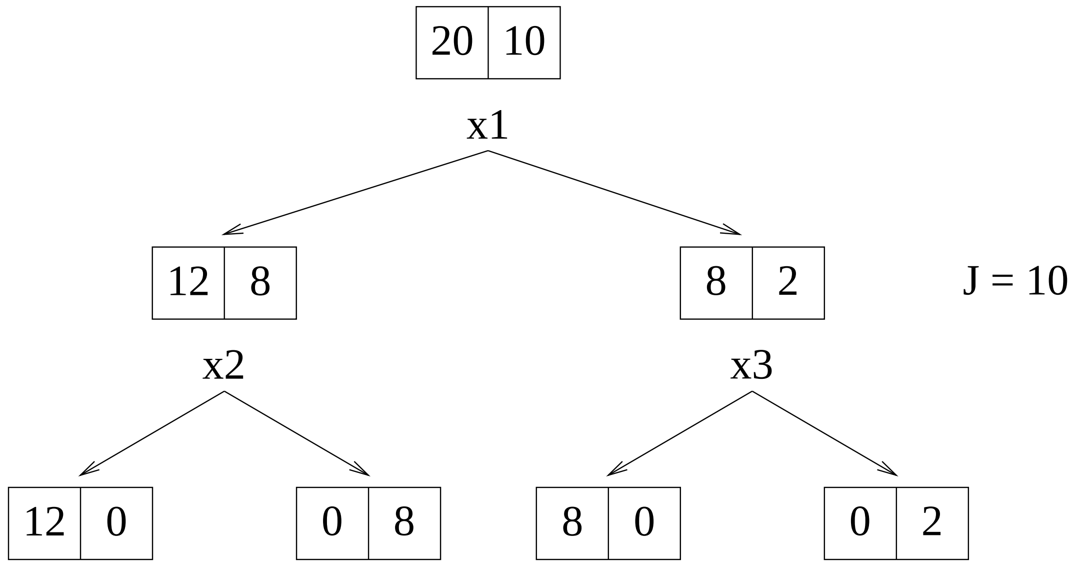
x_1	x_2	x_3	y
0	0	0	1
0	0	1	0
0	1	0	1
0	1	1	1
1	0	0	0
1	0	1	1
1	1	0	0
1	1	1	0



J=2 J=4 J=4

Choosing the Best Feature (3)

Unfortunately, this measure does not always work well, because it does not detect cases where we are making "progress" toward a good tree.



A Better Heuristic From Information Theory

Let V be a random variable with the following probability distribution:

$P(V=0)$	$P(V=1)$
0.2	0.8

The *surprise*, $S(V=v)$ of each value of V is defined to be

$$S(V=v) = -\lg P(V=v).$$

An event with probability 1 gives us zero surprise.

An event with probability 0 gives us infinite surprise!

It turns out that the surprise is equal to the number of bits of information that need to be transmitted to a recipient who knows the probabilities of the results.

This is also called the *description length* of $V=v$.

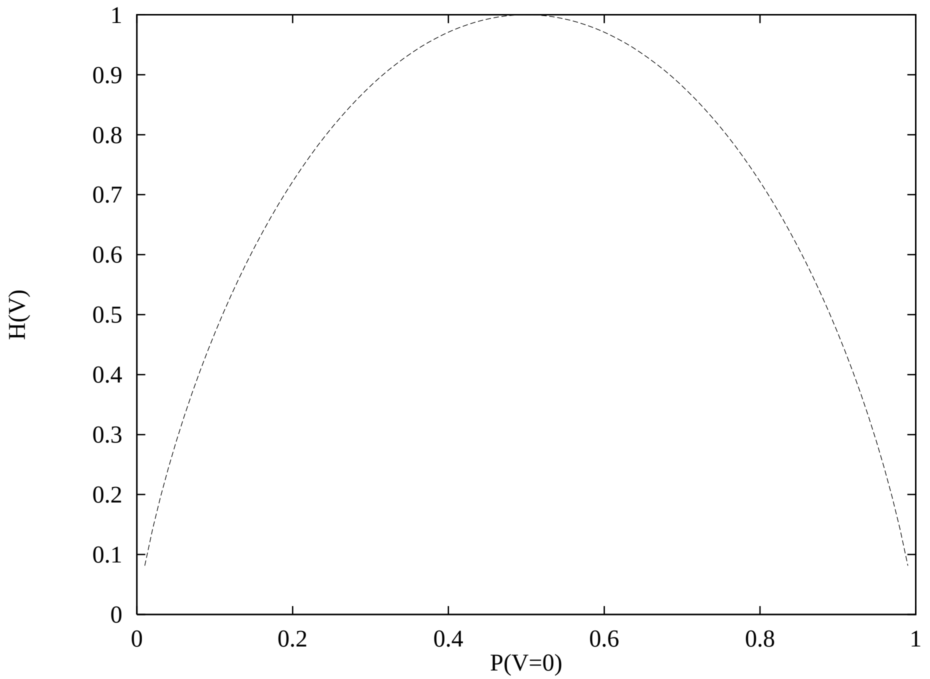
Fractional bits only make sense if they are part of a longer message (e.g., describe a whole sequence of coin tosses).

Entropy

The *entropy* of V, denoted $H(V)$ is defined as follows:

$$H(V) = \sum_{v=0}^{1} -P(H = v) \lg P(H = v).$$

This is the average surprise of describing the result of one "trial" of V (one coin toss).



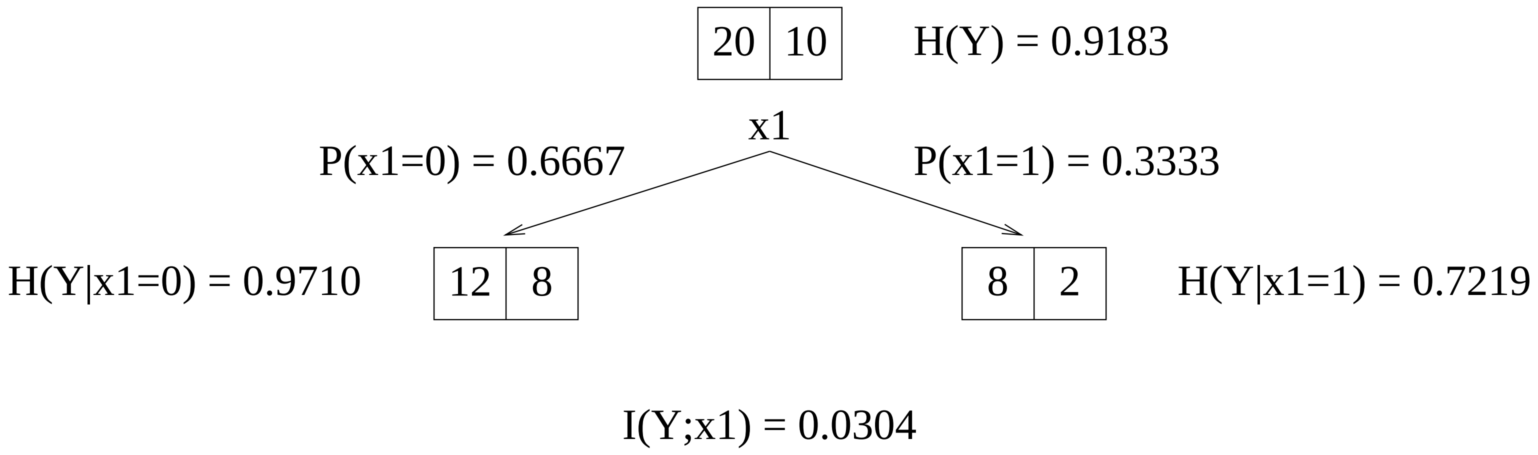
Entropy can be viewed as a measure of ignorance.

Mutual Information

Now consider two random variables A and B that are not necessarily independent. The *mutual information* between A and B is the amount of information we learn about B by knowning the value of A (and vice versa—it is symmetric). It is computed as follows:

$$I(A;B) = H(B) - \sum_b P(B=b) \cdot H(A|B=b)$$

In particular, consider the class Y of each training example and the value of feature x_1 to be random variables. Then the mutual information quantifies how much x_1 tells us about the value of the class Y.



Non-Boolean Features

- **Features with multiple discrete values**

  Construct a multiway split?

  Test for one value versus all of the others?

  Group the values into two disjoint subsets?

- **Real-valued features**

  Consider a threshold split using each observed value of the feature.

Whichever method is used, the mutual information can be computed to choose the best split.

Fundamental Tradeoff in Machine Learning

- **Hypothesis Space Size** (expressive power)
- **Number of Examples** (sample size)
- **Error Rate of Resulting Hypothesis** (accuracy)

$$\epsilon \cdot m = \ln |H| + \ln \frac{1}{\delta}$$

If $h \in H$ is *consistent* with m examples, then with probability $1 - \delta$ the error rate of h will be less than ϵ.

Larger H requires more examples or gives higher error rates.
If we want to learn from small sample sizes, then $|H| \approx e^{\epsilon m}$.

How Decision Tree Algorithms Manage This Tradeoff

- Start with a very small hypothesis space (single leaf node).
- Grow the hypothesis space until a (nearly) consistent hypothesis is found.

As long as the space grows slowly with m, the trees will be fairly accurate. (Number of leaves should be $< \ln m$ roughly.)

Corollary: Functions that cannot be represented approximately by small trees cannot be learned efficiently by decision tree algorithms.

Ockham's Razor: The most likely hypothesis is the simplest one that is consistent with all observations.

Choosing a Learning Algorithm

Question: How should we choose a learning algorithm for a new problem?

Answer: Choose an H that is likely to contain (a good approximation to) the unknown function.

Learning research seeks to find good flexible hypothesis spaces H and algorithms for searching them efficiently.

Brute force search is impossible, because H is typically exponential in size.