The learning problem is called **realizable** if the hypothesis space contains the true function; otherwise it is **unrealizable**.

On the other hand, in the name of better generalization ability it may be sensible to trade off exactness of fitting to simplicity of the hypothesis.

In other words, it may be sensible to be content with a hypothesis fitting the data less perfectly as long as it is simple.

The hypothesis space needs to be restricted so that finding a hypothesis that fits the data stays computationally efficient.

Machine learning concentrates on learning relatively simple knowledge representations.

---

Supervised learning can be done by choosing the hypothesis $h^*$ that is most probable given the data:

$$h^* = \arg\max_{h \in \mathcal{H}} P(h | \text{data})$$

By Bayes’ rule this is equivalent to

$$h^* = \arg\max_{h \in \mathcal{H}} P(\text{data} | h) P(h)$$

Then we can say that the prior probability $P(h)$ is high for a degree-1 or -2 polynomial, lower for degree-7 polynomial, and especially low for degree-7 polynomial with large, sharp spikes.

There is a tradeoff between the expressiveness of a hypothesis space and the complexity of finding a good hypothesis within that space.
18.3 Learning Decision Trees

- A decision tree takes as input an object or situation described by a set of attributes.
- It returns a decision — the predicted output value for the input.
- If the output values are discrete, then the decision tree classifies the inputs.
- Learning a continuous function is called regression.
- Each internal node in the tree corresponds to a test of the value of one of the properties, and the branches from the node are labeled with possible values of the test.
- Each leaf node in the tree specifies the value to be returned if the leaf is reached.
- To process an input, it is directed from the root of the tree through internal nodes to a leaf, which determines the output value.
A decision tree (of reasonable size) is an easy to comprehend way of representing knowledge.

Important in practice, heuristically learnable.

The previous decision tree corresponds to the goal predicate \( \textit{WillWait} \) – whether to wait for a table in a restaurant.

Its goal predicate can be seen as an assertion of the form

\[ \forall s : \textit{WillWait}(s) \iff (P_1(s) \lor \cdots \lor P_n(s)), \]

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

An exponentially large decision tree can express any Boolean function.

Typically, decision trees can represent many functions with much smaller trees.

For some kinds of functions this, however, is a real problem, e.g., \( \text{xor} \) and \( \text{maj} \) need exponentially large decision trees.

Decision trees, like any other knowledge representation, are good for some kinds of functions and bad for others.

Consider the set of all Boolean functions on \( n \) attributes.

How many different functions are in this set?

The truth table has \( 2^n \) rows, so there are \( 2^{2^n} \) different functions.

For example, when \( n = 6 \Rightarrow 2^{2^n} > 18 \times 10^{18} \), \( n = 10 \Rightarrow 2^{1024} \approx 10^{308} \), and \( n = 20 \Rightarrow > 10^{300000}\).

We will need some ingenious algorithms to find consistent hypotheses in such a large space.
Top-down induction of decision trees

- The input to the algorithm is a *training set*, which consists of examples \((x, y)\), where \(x\) is a vector of input attribute values and \(y\) is the single output value (class value) attached to them.
- We could simply construct a consistent decision tree that has one path from the root to a leaf for each example.
- Then we would be able to classify all training examples correctly, but the tree would not be able to generalize at all.
- Applying Occam’s razor, we should find the *smallest* decision tree that is consistent with the examples.
- Unfortunately, for any reasonable definition of “smallest,” finding the smallest tree is an intractable problem.

Successful decision tree learning algorithms are based on simple heuristics and do a good job of finding a smallish tree.
- The basic idea is to test the most important attribute first.
- Because the aim is to classify instances, “most important” attribute is the one that makes the most difference to the classification of an example.
- Actual decision tree construction happens with a recursive algorithm:
  - First the most important attribute is chosen to the root of the tree,
  - the training data is divided according to the values of the chosen attribute, and
  - (sub)tree construction continues using the same idea.
GROWCONSTREE(A, S)

Input: A set of training examples S on attributes A
Output: A decision tree that is consistent with S

1. if all examples in S have class C then
2. return an one-leaf tree labeled by C
3. else
4. select an attribute a from A
5. partition S into S₁, …, Sₖ by the value of a
6. for i = 1 to k do
7. Tᵢ = GROWCONSTREE(A \− \{a\}, Sᵢ)
8. return a tree T that has a in its root and
9. Tᵢ as its i-th subtree

If there are no examples left
⇒ no such example has been observed, and we return a default value calculated from the majority classification at the node’s parent (or the majority classification at the root)

If there are no attributes left but still instances of several classes in the remaining portion of the data,
⇒ these examples have exactly the same description, but different classification

Then we say that there is noise in the data
Noise may follow either when the attributes do not give enough information to describe the situation fully, or when the domain is truly nondeterministic
One simple way out of this problem is to use a majority vote
Choosing attribute tests

- The idea is to pick the attribute that goes as far as possible toward providing an exact classification of the examples
- A perfect attribute divides the examples into sets that contain only instances of one class
- A really useless attribute leaves the example sets with roughly the same proportion of instances of all classes as the original set
- To measure the usefulness of attributes we can use, for instance, the expected amount of information provided by the attribute — i.e., its Shannon entropy
- Information theory measures information content in bits
- One bit of information is enough to answer a yes/no question about which one has no idea, such as the flip of a fair coin

In general, if the $n$ possible answers $v_i$ have probabilities $P(v_1)$, then the entropy $H$ of the actual answer is

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

For example, $H(0.5, 0.5) = 2(-0.5 \log_2 0.5) = 1$ bit

In choosing attribute tests, we want to calculate the change of the value distribution $P(C)$ of the class attribute $C$, if the training set $S$ is divided into subsets according to the value of attribute $a$

$$H_S(P(C)) - Gain_S(P(C) \mid a),$$

where

$$Gain_S(P(C) \mid a) = \sum_{S_i} \frac{|S_i|}{|S|} \cdot H_{S_i}(P(C)),$$

when $a$ divides $S$ in subsets $S_i$.
Let the training set $S$ contain 14 apples and 6 oranges.

Hence,
$$H_S(P(C)) = H(0.7, 0.3) \approx 0.7 \times 0.515 + 0.3 \times 1.737 \approx 0.881$$

Suppose that attribute $a$ divides the data s.t.
$$S_1 = \{7 \text{ apples}, 3 \text{ oranges}\},$$
$$S_2 = \{7 \text{ apples}\}, \text{and}$$
$$S_3 = \{3 \text{ oranges}\}$$

then
$$Gain_S(P(C) \mid a) = \sum_{S_i} \frac{|S_i|}{|S|} \cdot H_{S_i}(P(C))$$
$$= \left(\frac{10}{20}\right) \times H(0.7, 0.3) + 0 + 0$$
$$\approx 0.881 \approx 0.441$$

Assessing performance of learning algorithms

- Divide the set of examples into disjoint training set and test set.
- Apply the training algorithm to the training set, generating a hypothesis $h$.
- Measure the percentage of examples in the test set that are correctly classified by $h$: $h(x) = y$ for an $(x, y)$ example.
- Repeat the above-mentioned steps for different sizes of training sets and different randomly selected training sets of each size.
- The result of this procedure is a set of data that can be processed to give the average prediction quality as a function of the size of the training set.
- Plotting this function on a graph gives the learning curve.
- An alternative (better) approach to testing is cross-validation.
The idea in \textit{k-fold cross-validation} is that each example serves double duty as training data and test data. First we split the data into \( k \) equal subsets. We then perform \( k \) rounds of learning; on each round \( 1/k \) of the data is held out as a test set and the remaining examples are used as training data. The average test set score of the \( k \) rounds should then be a better estimate than a single score. Popular values for \( k \) are 5 and 10 — enough to give an estimate that is statistically likely to be accurate, at the cost of 5 to 10 times longer computation time. The extreme is \( k = n \), also known as leave-one-out cross-validation (LOO[CV], or jackknife).

\section*{Generalization and overfitting}

- If there are two or more examples with the same description (in terms of attributes) but different classifications \( \Rightarrow \) no consistent decision tree exists.
- The solution is to have each leaf node report either:
  - The majority classification for its set of examples, if a deterministic hypothesis is required, or
  - the estimated probabilities of each classification using the relative frequencies.
- It is quite possible, and in fact likely, that even when vital information is missing, the learning algorithm will find a consistent decision tree.
- This is because the algorithm can use \textit{irrelevant} attributes, if any, to make spurious distinctions among the examples.
Consider trying to predict the roll of a die on the basis of
- The day and
- The month in which the die was rolled, and
- Which is the color of the die,
then as long as no two examples have identical descriptions, the
learning algorithm will find an exact hypothesis

- Such a hypothesis will be totally spurious
- The more attributes there are, the more likely it is that an exact
  hypothesis will be found
- The correct tree to return would be a single leaf node with
  probabilities close to 1/6 for each roll
- This problem is an example of overfitting, a very general
  phenomenon afflicting every kind of learning algorithm and target
  function, not only random concepts

Decision tree pruning

- A simple approach to deal with overfitting is to prune the
decision tree
- Pruning works by preventing recursive splitting on attributes
  that are not clearly relevant

- Suppose we split a set of examples using an irrelevant
  attribute
- Generally, we would expect the resulting subsets to have
  roughly the same proportions of each class as the original set
- In this case, the information gain will be close to zero
- How large a gain should we require in order to split on a
  particular attribute?
A statistical **significance test** begins by assuming that there is no underlying pattern (the so-called **null hypothesis**) and then analyzes the actual data to calculate the extent to which they deviate from a perfect absence of pattern. If the degree of deviation is statistically unlikely (usually taken to mean a 5% probability or less), then that is considered to be good evidence for the presence of a significant pattern in the data.

The probabilities are calculated from standard distributions of the amount of deviation one would expect to see in random sampling. Null hypothesis: the attribute at hand is irrelevant and, hence, its information gain for an infinitely large sample is zero. We need to calculate the probability that, under the null hypothesis, a sample of size \( v = n + p \) would exhibit the observed deviation from the expected distribution of examples.

Let the numbers positive and negative examples in each subset be \( p_i \) and \( n_i \), respectively. Their expected values, assuming true irrelevance, are

\[
\begin{align*}
  p_i' &= p \cdot (p_i + n_i)/(p + n) \\
  n_i' &= n \cdot (p_i + n_i)/(p + n)
\end{align*}
\]

where \( p \) and \( n \) are the total numbers of positive and negative examples in the training set.

A convenient measure for the total deviation is given by

\[
D = \sum_{i=1}^{v} \frac{(p_i - p_i')^2}{p_i'} + \frac{(n_i - n_i')^2}{n_i'}
\]

Under the null hypothesis, the value of \( D \) is distributed according to the \( \chi^2 \) (chi-squared) distribution with \( (v - 1) \) degrees of freedom. The probability that the attribute is really irrelevant can be calculated with the help of standard \( \chi^2 \) tables.
The above method is known as $\chi^2$ (pre-)pruning.

Pruning allows the training examples to contain noise and it also reduces the size of the decision trees and makes them more comprehensible.

More common than pre-pruning are post-pruning methods in which:

- One first constructs a decision tree that is as consistent as possible with the training data and
- Then removes those subtrees that have likely been added due to the noise.

In cross-validation the known data is divided in $k$ parts, each of which is used as a test set in its turn for a decision tree that has been grown on the other $k-1$ subsets.

Thus one can approximate how well each hypothesis will predict unseen data.

**Broadening the applicability of decision trees**

- In practice decision tree learning has to answer also the following questions:
  - Missing attribute values: while learning and in classifying instances
  - Multivalued discrete attributes: value subsetting or penalizing against too many values
  - Numerical attributes: split point selection for interval division
  - Continuous-valued output attributes

- Decision trees are used widely and many good implementations are available (for free).

- Decision trees fulfill understandability, contrary to neural networks, which is a legal requirement for financial decisions.