2

Occam's Razor

The PAC model introduced in Chapter 1 defined learning directly in terms of the predictive power of the hypothesis output by the learning algorithm. It was possible to apply this measure of success to a learning algorithm because we made the assumption that the instances are drawn independently from a *fixed* probability distribution $\mathcal{D}$, and then measured predictive power with respect to this same distribution.

In this chapter, we consider a rather different definition of learning that makes no assumptions about how the instances in a labeled sample are chosen. (We still assume that the labels are generated by a target concept chosen from a known class.) Instead of measuring the *predictive* power of a hypothesis, the new definition judges the hypothesis by how succinctly it explains the *observed* data (a labeled sample). The crucial difference between PAC learning and the new definition is that in PAC learning, the random sample drawn by the learning algorithm is intended only as an aid for reaching an accurate model of some external process (the target concept and distribution), while in the new definition we are concerned only with the fixed sample before us, and not any external process.

This new definition will be called *Occam learning*, because it formalizes a principle that was first expounded by the theologian William

of Occam, and which has since become a central doctrine of scientific methodology. The principle is often referred to as *Occam's Razor* to indicate that overly complex scientific theories should be subjected to a simplifying knife.

If we equate "simplicity" with representational succinctness, then another way to interpret Occam's principle is that learning is the act of finding a pattern in the observed data that facilitates a compact representation or compression of this data. In our simple concept learning setting, succinctness is measured by the size of the representation of the hypothesis concept. Equivalently, we can measure succinctness by the cardinality of the hypothesis class used by the algorithm, for if this class is small then a typical hypothesis from the class can be represented by a short binary string, and if this class is large then a typical hypothesis must be represented by a long string. Thus an algorithm is an *Occam algorithm* if it finds a short hypothesis consistent with the observed data.

Despite its long and illustrious history in the philosophy of science and its extreme generality, there is something unsatisfying about the notion of an Occam algorithm. After all, the primary goal of science (or more generally, of the learning process) is to formulate theories that accurately predict future observations, not just to succinctly represent past observations. In this chapter, we will prove that when restricted to the probabilistic setting of the PAC model, Occam algorithms do indeed have predictive power. This provides a formal justification of the Occam principle, albeit in a restricted setting.

Thus, under appropriate conditions, any algorithm that always finds a succinct hypothesis that is consistent with a given input sample is automatically a PAC learning algorithm. In addition to the philosophical interpretation we have just discussed, this reduction of PAC learning to Occam learning provides a new method of designing PAC learning algorithms.

2.1 Occam Learning and Succinctness

As in Chapter 1, let $X = \cup_{n \geq 1} X_n$ be the instance space, let $\mathcal{C} = \cup_{n \geq 1} \mathcal{C}_n$ be the target concept class, and let $\mathcal{H} = \cup_{n \geq 1} \mathcal{H}_n$ be the hypothesis representation class. In this chapter we will assume, unless explicitly stated otherwise, that the hypothesis representation scheme of $\mathcal{H}$ uses a binary alphabet, and we define *size*(h) to be the length of the bit string h. Also, recall that for a concept $c \in \mathcal{C}$, *size*(c) denotes the size of the smallest representation of c in $\mathcal{H}$.

Let $c \in \mathcal{C}_n$ denote the target concept. A labeled sample S of cardinality m is a set of pairs:

$$S = \{\langle x_1, c(x_1)\rangle, \ldots, \langle x_m, c(x_m)\rangle\}.$$

An Occam algorithm L takes as input a labeled sample S, and outputs a "short" hypothesis h that is consistent with S. By consistent we mean that $h(x_i) = c(x_i)$ for each i, and by "short" we mean that *size*(h) is a sufficiently slowly growing function of n, *size*(c) and m. This is formalized in the following definition.

Definition 6 *Let $\alpha \geq 0$ and $0 \leq \beta < 1$ be constants. L is an (α, β)-*
Occam algorithm for $\mathcal{C}$ using $\mathcal{H}$ *if on input a sample S of cardinality m labeled according to $c \in \mathcal{C}_n$, L outputs a hypothesis $h \in \mathcal{H}$ such that:*

- *h is consistent with S.*
- *size$(h) \leq (n \cdot size(c))^{\alpha} m^{\beta}$.*

We say that L is an **efficient** *(α, β)-Occam algorithm if its running time is bounded by a polynomial in n, m and size(c).*

In what sense is the output h of an Occam algorithm succinct? First let us assume that $m >> n$, so that the above bound can be effectively

simplified to $size(h) < m^\beta$ for some $\beta < 1$. Since the hypothesis h is consistent with the sample S, h allows us to reconstruct the m labels $c(x_1) = h(x_1), \ldots, c(x_m) = h(x_m)$ given only the unlabeled sequence of instances $x_1, \ldots, x_m$. Thus the m bits $c(x_1), \ldots, c(x_m)$ have been effectively *compressed* into a much shorter string h of length at most m^β. Note that the requirement $\beta < 1$ is quite weak, since a consistent hypothesis of length $O(mn)$ can always be achieved by simply storing the sample S in a table (at a cost of $n + 1$ bits per labeled example) and giving an arbitrary (say negative) answer for instances that are not in the table. We would certainly not expect such a hypothesis to have any predictive power.

Let us also observe that even in the case $m << n$, the shortest consistent hypothesis in $\mathcal{H}$ may in fact be the target concept, and so we must allow $size(h)$ to depend at least linearly on $size(c)$. The definition of succinctness above is considerably more liberal than this in terms of the allowed dependence on n, and also allows a generous dependence on the number of examples m. We will see cases where this makes it easier to efficiently find a consistent hypothesis — by contrast, computing the shortest hypothesis consistent with the data is often a computationally hard problem.

The next theorem, which is the main result of this chapter, states that any efficient Occam algorithm is also an efficient PAC learning algorithm.

Theorem 2.1 *(Occam's Razor) Let L be an efficient (α, β)-Occam algorithm for C using $\mathcal{H}$. Let $\mathcal{D}$ be the target distribution over the instance space X, let $c \in C_n$ be the target concept, and $0 < \epsilon, \delta \leq 1$. Then there is a constant $a > 0$ such that if L is given as input a random sample S of m examples drawn from $EX(c, \mathcal{D})$, where m satisfies*

$$m \geq a\left(\frac{1}{\epsilon}\log\frac{1}{\delta} + \left(\frac{(n \cdot size(c))^\alpha)}{\epsilon}\right)^{\frac{1}{1-\beta}}\right)$$

then with probability at least $1-\delta$ the output h of L satisfies $error(h) \leq \epsilon$. Moreover, L runs in time polynomial in n, $size(c)$, $1/\epsilon$ and $1/\delta$.

Notice that as β tends to 1, the exponent in the bound for m tends to infinity. This corresponds with our intuition that as the length of the hypothesis approaches that of the data itself, the predictive power of the hypothesis is diminishing.

For the applications we give later, it turns out to be most convenient to state and prove Theorem 2.1 in a slightly more general form, in which we measure representational succinctness by the cardinality of the hypothesis class rather than by the bit length *size*(h). We then prove Theorem 2.1 as a special case. To make this precise, let $\mathcal{H}_n = \cup_{m \geq 1} \mathcal{H}_{n,m}$. Consider a learning algorithm for $\mathcal{C}$ using $\mathcal{H}$ that on input a labeled sample S of cardinality m outputs a hypothesis from $\mathcal{H}_{n,m}$. The following theorem shows that if $|\mathcal{H}_{n,m}|$ is small enough, then the hypothesis output by L has small error with high confidence.

Theorem 2.2 *(Occam's Razor, Cardinality Version) Let $\mathcal{C}$ be a concept class and $\mathcal{H}$ a representation class. Let L be an algorithm such that for any n and any $c \in \mathcal{C}_n$, if L is given as input a sample S of m labeled examples of c, then L runs in time polynomial in n, m and size(c), and outputs an $h \in \mathcal{H}_{n,m}$ that is consistent with S. Then there is a constant $b > 0$ such that for any n, any distribution $\mathcal{D}$ over X_n, and any target concept $c \in \mathcal{C}_n$, if L is given as input a random sample from $EX(c, \mathcal{D})$ of m examples, where $|\mathcal{H}_{n,m}|$ satisfies*

$$\log |\mathcal{H}_{n,m}| \leq b\epsilon m - \log \frac{1}{\delta}$$

(or equivalently, where m satisfies $m \geq (1/b\epsilon)(\log |\mathcal{H}_{n,m}| + \log(1/\delta))$) then L is guaranteed to find a hypothesis $h \in \mathcal{H}_n$ that with probability at least $1 - \delta$ obeys error$(h) \leq \epsilon$.

Note that here we do not necessarily claim that L is an *efficient* PAC learning algorithm. In order for the theorem to apply, we must (if possible) pick m large enough so that $b\epsilon m$ dominates $\log |\mathcal{H}_{n,m}|$. Moreover, since the running time of L has a polynomial dependence on m, in order

to assert that L is an efficient PAC algorithm, we also have to bound m by some polynomial in n, $size(c)$, $1/\epsilon$ and $1/\delta$. The proof of Theorem 2.1 relies on the fact that in the case of an (α, β)-Occam algorithm, $\log |\mathcal{H}_{n,m}|$ grows only as m^β, and therefore given any ϵ, this is smaller than $b\epsilon m$ for a small value of m.

We first give a proof of Theorem 2.2.

Proof: We say that a hypothesis $h \in \mathcal{H}_{n,m}$ is *bad* if $error(h) > \epsilon$, where the error is of course measured with respect to the target concept c and and the target distribution $\mathcal{D}$. Then by the independence of the random examples, the probability that a fixed bad hypothesis h is consistent with a randomly drawn sample of m examples from $EX(c, \mathcal{D})$ is at most $(1 - \epsilon)^m$. Using the union bound, this implies that if $\mathcal{H}' \subseteq \mathcal{H}_{n,m}$ is the set of *all* bad hypotheses in $\mathcal{H}_{n,m}$, then the probability that some hypothesis in $\mathcal{H}'$ is consistent with a random sample of size m is at most $|\mathcal{H}'|(1 - \epsilon)^m$. We want this to be at most δ; since $|\mathcal{H}'| \leq |\mathcal{H}_{n,m}|$ we get a stronger condition if we solve for $|\mathcal{H}_{n,m}|(1 - \epsilon)^m \leq \delta$. Taking logarithms, we obtain $\log |\mathcal{H}_{n,m}| \leq m \log(1/(1 - \epsilon)) - \log(1/\delta)$. Using the fact that $\log(1/(1 - \epsilon)) = \Theta(\epsilon)$, we get the statement of the theorem.

□(Theorem 2.2)

We now prove Theorem 2.1:

Proof: Let $\mathcal{H}_{n,m}$ denote the set of all possible hypothesis representations that the (α, β)-Occam algorithm L might output when given as input a labeled sample S of cardinality m. Since L is an (α, β)-Occam algorithm, every such hypothesis has bit length at most $(n \cdot size(c))^\alpha m^\beta$, thus implying that $|\mathcal{H}_{n,m}| \leq 2^{(n \cdot size(c))^\alpha m^\beta}$. By Theorem 2.2, the output of L has error at most ϵ with confidence at least $1 - \delta$ provided

$$\log |\mathcal{H}_{n,m}| \leq b\epsilon m - \log \frac{1}{\delta}.$$

Transposing, we want m such that

$$m \geq \frac{1}{b\epsilon} \log |\mathcal{H}_{n,m}| + \frac{1}{b\epsilon} \log \frac{1}{\delta}$$

The above condition can be satisfied by picking m such that both $m \geq (2/b\epsilon)\log|\mathcal{H}_{n,m}|$ and $m \geq (2/b\epsilon)\log(1/\delta)$ hold. Choosing $a = 2/b$ yields the statement of the theorem. □(Theorem 2.1)

2.2 Improving the Sample Size for Learning Conjunctions

As an easy warm-up to some more interesting applications of Occam's Razor, we first return to the problem of PAC learning conjunctions of boolean literals, and apply Theorem 2.2 to slightly improve the sample size bound (and therefore the running time bound) of the learning algorithm we presented for this problem in Section 1.3.

Thus as in Section 1.3, we let $X_n = \{0,1\}^n$. Each $a \in \{0,1\}^n$ is interpreted as an assignment to the n boolean variables $x_1, \ldots, x_n$. Let $\mathcal{C}_n$ be the class of conjunctions of literals over $x_1, \ldots, x_n$. Recall that our learning algorithm started with a hypothesis that is the conjunction of all the $2n$ literals. Given as input a set of m labeled examples, the algorithm ignored negative examples, and on each positive example $\langle a, 1\rangle$, the algorithm deleted any literal z such that $z = 0$ in a. Note that this ensures that upon receiving the positive example a, the hypothesis is updated to be consistent with this example. Furthermore, any future deletions will not alter this consistency, since deletions can only increase the set of positive examples of the hypothesis. Finally, recall that we already argued in Section 1.3 that this algorithm never misclassifies any negative example of the target conjunction c. Thus, if we run the algorithm on an arbitrary sample S of labeled examples of some target conjunction, it always outputs a hypothesis conjunction that is consistent with S, and thus it is an Occam algorithm. Note that in this simple example, $size(h)$ (or equivalently, $\log|\mathcal{H}_{n,m}|$) depends only on n and not on m or $size(c)$.

Now the number of conjunctions over $x_1, \ldots, x_n$ is bounded by 3^n (each variable occurs positively or negatively or is absent entirely), so

applying Theorem 2.2, we see that $O((1/\epsilon)\log(1/\delta) + n/\epsilon)$ examples are sufficient to guarantee that the hypothesis output by the learning algorithm has error less than ϵ with confidence at least $1 - \delta$. This is an improvement by a logarithmic factor over the bound given in Chapter 1.

2.3 Learning Conjunctions with Few Relevant Variables

Despite the efficiency of our algorithm for PAC learning boolean conjunctions, we can still imagine improvements. Let us define $size(c)$ be the number of literals appearing in the target conjunction c. Notice that $size(c) \leq n$, but the size of the sample drawn by our learning algorithm for conjunctions is proportional to n independent of how small $size(c)$ might be. In this section, we give a new algorithm that reduces the number of examples to nearly $size(c)$. It can be argued that it is often realistic to assume that $size(c) << n$, since we typically describe an object by describing only a few attributes out of a large list of potential attributes.

Even though we greatly improve the sample size for the case of small $size(c)$, we should point out that the running time of the new learning algorithm still grows with n, since the instances are of length n, and the algorithm must take enough time to read each instance. An interesting feature of the new algorithm is that it makes use of the negative examples, unlike our previous algorithm for learning conjunctions.

In order to describe the new algorithm, we need to introduce a combinatorial problem and a well-known algorithm for its approximate solution. This approximation algorithm has many applications in computational learning theory.

The Set Cover Problem. Given as input a collection $\mathcal{S}$ of subsets of $U = \{1, \ldots, m\}$, find a subcollection $\mathcal{T} \subseteq \mathcal{S}$ such that $|\mathcal{T}|$ is minimized,

and the sets in $\mathcal{T}$ form a cover of U:

$$\bigcup_{t \in \mathcal{T}} t = U.$$

We assume, of course, that the entire collection $\mathcal{S}$ is itself a cover. For any instance $\mathcal{S}$ of the Set Cover Problem, we let $opt(\mathcal{S})$ denote the number of sets in a minimum cardinality cover.

Finding an optimal cover is a well-known *NP*-hard problem. However, there is an efficient greedy heuristic that is guaranteed to find a cover $\mathcal{R}$ of cardinality at most $O(opt(\mathcal{S}) \log m)$.

The greedy heuristic initializes $\mathcal{R}$ to be the empty collection. It first adds to $\mathcal{R}$ the set s^* from $\mathcal{S}$ with the largest cardinality, and then updates $\mathcal{S}$ by replacing each set s in $\mathcal{S}$ by $s - s^*$. It then repeats the process of choosing the remaining set of largest cardinality and updating $\mathcal{S}$ until all the elements of $\{1, \ldots, m\}$ are covered by $\mathcal{R}$.

The greedy heuristic is based on the following fact: let $U^* \subseteq U$. Then there is always a set t in $\mathcal{S}$ such that $|t \cap U^*| \geq |U^*|/opt(\mathcal{S})$. To see why this is true, just observe that U^* has a cover of size at most $opt(\mathcal{S})$ (since U does), and at least one of the sets in the optimal cover must cover a $1/opt(\mathcal{S})$ fraction of U^*.

Let $U_i \subseteq U$ denote the set of elements still not covered after i steps of the greedy heuristic. Then

$$|U_{i+1}| \leq |U_i| - \frac{|U_i|}{opt(\mathcal{S})} = |U_i| \left(1 - \frac{1}{opt(\mathcal{S})}\right).$$

So by induction on i:

$$|U_i| \leq \left(1 - \frac{1}{opt(\mathcal{S})}\right)^i m.$$

Choosing $i \geq opt(\mathcal{S}) \log m$ suffices to drive this upper bound below 1. Thus all the elements of U are covered after the algorithm has chosen $opt(\mathcal{S}) \log m$ sets.

We now return to the problem of PAC learning conjunctions with few relevant variables. We shall describe our new algorithm as an Occam algorithm and apply Theorem 2.2 to obtain the required sample size for PAC learning. Thus, given a sample S of m examples of a target conjunction, the new Occam algorithm starts by applying our original conjunctions algorithm — which uses only the positive examples — to S in order to produce a hypothesis conjunction h. This conjunction will have the property that it is consistent with S, since the old algorithm was indeed an Occam algorithm. The new algorithm will then use the negative examples in S to exclude several additional literals from h in a manner described below, to compute a new hypothesis conjunction h' containing at most $size(c) \log m$ of the literals appearing in h. This new smaller hypothesis will still be consistent with S, and so the sample size bound for PAC learning can be derived from Theorem 2.2.

Recall that excluding literals from h does not affect consistency with the positive examples in S, since the set of positive examples of h only grows as we delete literals. However, the new algorithm has to carefully choose which literals of h it excludes in order to ensure that the hypothesis is still consistent with all the negative examples in S. To do this, we cast the problem as an instance of the Set Cover Problem and apply the greedy algorithm.

For each literal z appearing in h, we can identify a subset $N_z \subseteq S$ of the negative examples in S with the property that inclusion of z in the hypothesis conjunction is sufficient to guarantee consistency with N_z. The set N_z is just those negative examples in $\langle a, 0 \rangle \in S$ for which the value of z is 0 in a. Thus, we can think of the inclusion of z in our hypothesis conjunction as "covering" the set N_z of negative examples. If we have a collection of N_z that covers all the negative examples of S, and each z appears in h, then the conjunction h' of this collection will still form a hypothesis consistent with S.

Our goal is thus reduced to covering the set of all negative examples in S with the minimum number of the sets N_z. Applying the greedy heuristic to this problem, and noting that among the literals of h, a cover

of $size(c)$ sets exists (since the literals that occur in the target conjunction must form a cover), we get a cover of size $size(c) \log m$; in other words, our hypothesis class $\mathcal{H}_{n,m}$ is the set of all conjunctions of at most $size(c) \log m$ literals. Using the fact that a conjunction of ℓ literals over n variables can be encoded using $\ell \log n$ bits, and setting $\ell = size(c) \log m$, we get a bound of $size(c) \log m \log n$ on the number of bits needed to represent our hypothesis, and thus $|\mathcal{H}_{n,m}| \leq 2^{size(c) \log m \log n}$. Applying the condition $m = \Omega((1/\epsilon) \log |\mathcal{H}_{n,m}|)$ required by Theorem 2.2, we obtain the constraint $m \geq c_1((1/\epsilon) size(c) \log m \log n)$ for some constant $c_1 > 0$. It is easily verified that this is satisfied provided $m \geq c_1((1/\epsilon) size(c) \log n \log(size(c) \log n))$. Thus, the overall sample size required by the new algorithm is

$$m \geq c_1 \left(\frac{1}{\epsilon} \log \frac{1}{\delta} + \frac{size(c) \log n (\log size(c) + \log \log n)}{\epsilon} \right).$$

Note that this bound has a slightly superlinear dependence on $size(c)$, but only an approximately logarithmic dependence on the total number of variables n.

In fact, a slight modification of this algorithm that we shall now sketch quite briefly gives a better bound. The basic idea behind the modification is that rather than running the greedy cover heuristic until the hypothesis covers all of the negative examples, we shall run it only until the hypothesis misclassifies fewer than $\epsilon m/2$ negative examples. Thus, our resulting hypothesis will be almost but not quite consistent with its input sample, where the degree of consistency is controlled by the desired error bound ϵ.

For the analysis, observe that now the halting condition for the greedy heuristic is $(1 - 1/size(c))^i m < (\epsilon/2)m$ instead of $(1 - 1/size(c))^i m < 1$ as before; here we are using the correspondence between $opt(\mathcal{S})$ in the covering problem and $size(c)$ in the PAC learning problem. Thus, we halt with a hypothesis of $i = size(c) \log(2/\epsilon)$ literals instead of $i = size(c) \log m$ literals. This gives a smaller hypothesis class cardinality of $2^{size(c) \log(2/\epsilon) \log n}$.

Now we just need a lemma stating that the probability that a fixed conjunction h such that $error(h) \geq \epsilon$ is consistent with at least a fraction $1 - \epsilon/2$ of m random examples is bounded by some exponentially decreasing function of m (that is, we need the analogue of the bound $(1-\epsilon)^m$ on the probability that a hypothesis of error greater than ϵ is completely consistent with the sample). It turns out that we can state a bound of $e^{-\epsilon m/16}$ on this probability, and this is discussed in the section on Chernoff Bounds in the Appendix of Chapter 9. For our immediate problem, given this bound we can now apply the same arguments as those in the proof of Theorem 2.2, and by solving $2^{size(c)\log(2/\epsilon)\log n} e^{-\epsilon m/16} \leq \delta$ we obtain a sample size bound of

$$m \geq c_1 \left(\frac{1}{\epsilon} \log \frac{1}{\delta} + \frac{size(c) \log(2/\epsilon) \log n}{\epsilon} \right).$$

2.4 Learning Decision Lists

Our final application of Occam learning is to an algorithm for PAC learning **decision lists** over the boolean variables $x_1, \ldots, x_n$. A decision list may be thought of as an ordered sequence of if-then-else statements. The sequence of conditions in the decision list are tested in order, and the answer associated with the first satisfied condition is output.

Formally, a **k-decision list** over the boolean variables $x_1, \ldots, x_n$ is an ordered sequence $L = (c_1, b_1), \ldots, (c_l, b_l)$ and a bit b, in which each c_i is a conjunction of at most k literals over $x_1, \ldots, x_n$, and each $b_i \in \{0, 1\}$. For any input $a \in \{0,1\}^n$, the value $L(a)$ is defined to be b_j, where j is the smallest index satisfying $c_j(a) = 1$; if no such index exists, then $L(a) = b$. Thus, b is the "default" value in case a falls off the end of the list. We call b_i the bit *associated* with the condition c_i. Figure 2.1 shows an example of a 2-decision list along with its evaluation on a particular input.

First let us consider the expressive power of k-decision lists. We

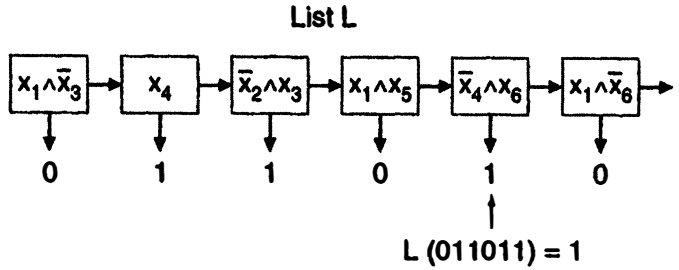

Figure 2.1: *A 2-decision list and the path followed by an input. Evaluation starts at the leftmost item and continues to the right until the first condition is satisfied, at which point the binary value below becomes the final result of the evaluation.*

observe that if a concept c can be represented as a k-decision list, then so can $\neg c$ (simply complement the values of the b_i). Clearly, any k-DNF formula can be represented as a k-decision list of the same length (choose an arbitrary order in which to evaluate the terms of the k-DNF, setting all the b_i to 1 and the default b to 0). Since k-decision lists are closed under complementation, they can also represent k-CNF formulae. Furthermore, in Exercise 2.1 we demonstrate that for each k there exist functions that can be represented by a k-decision list, but not by either a k-DNF or a k-CNF formula. Thus, k-decision lists strictly generalize these classes.

Theorem 2.3 *For any fixed $k \geq 1$, the representation class of k-decision lists is efficiently PAC learnable.*

Proof: We give an Occam algorithm and apply Theorem 2.2. We present the algorithm for 1-decision lists; the problem for general k can easily be reduced to this problem, exactly as the k-CNF PAC learning problem was reduced to the problem of PAC learning conjunctions in Chapter 1.

Given an input sample S of m examples of some 1-decision list, our Occam algorithm starts with the empty decision list as its hypothesis. In each step, it finds some literal z such that the set $S_z \subseteq S$, which we define to be the set of examples (positive or negative) in which z in set to 1, is both non-empty and has the property that it contains either only positive examples of the target concept, or only negative examples. We call such a z a *useful* literal. The algorithm then adds this literal (with the associated bit 1 if S_z contained only positive examples, and the associated bit 0 if S_z contained only negative examples) as the last condition in the current hypothesis decision list, updates S to be $S - S_z$, and iterates the process until $S = \emptyset$ and therefore all examples are correctly classified by the hypothesis decision list.

To prove that the algorithm always succeeds in finding a consistent hypothesis, it suffices to show that it always succeeds in finding a useful literal z at each step as long as $S \neq \emptyset$. But this is true because the target decision correctly classifies every element of S, and so the first condition z in the target decision list such that S_z is non-empty is a useful literal.

Since any decision list on n variables can be encoded in $O(n \log n)$ bits, we can apply Theorem 2.2 to obtain a sample size bound of $m \geq c_1((1/\epsilon)(\log(1/\delta)+n \log n))$ for PAC learning. Since the Occam algorithm clearly runs in time polynomial in m, we have efficient PAC learning. □(Theorem 2.3)

2.5 Exercises

2.1. Show that for each k, there exists a function that can be represented as a k-decision list, but not by a k-CNF or k-DNF formula.

2.2. A **decision tree** is similar to a 1-decision list, except now we allow the (single-literal) decision conditions to be placed in a binary tree, with the decision bits placed only at the leaves. To evaluate such a tree T on input $a \in \{0,1\}^n$, we simply follow the path through T defined by

starting at the root of T and evaluating the literal at each node on input a, going left if the evaluation yields 0 and right if it yields 1. The value $T(a)$ is the bit value stored at the leaf reached by this path. Figure 2.2 shows an example of a decision tree along with its evaluation on an input.

We define the **rank** of a decision tree T recursively as follows: the rank of a tree consisting of a single node is 0. If the ranks of T's left subtrees and right subtrees are r_L and r_R respectively, then if $r_L = r_R$ the rank of T is r_L+1; otherwise, it is $\max(r_L, r_R)$. The rank is a measure of how "unbalanced" the tree is.

Compute the rank of the decision tree given in Figure 2.2, and show that the class of functions computed by rank r decision trees is included in the class of functions computed by r-decision lists. Thus, for any fixed r we can efficiently PAC learn rank r decision trees.

2.3. Let $\mathcal{C}$ be any concept class. Show that if $\mathcal{C}$ is efficiently PAC learnable, then for some constants $\alpha \geq 1$ and $\beta < 1$ there is an (α, β)-Occam algorithm for $\mathcal{C}$. Hint: construct an appropriate simulation of the PAC learning algorithm L in which the accuracy parameter depends on the degree of the polynomial running time of L.

2.4. Recall that following our final definition of PAC learning (Definition 4), we emphasized the importance of restricting our attention to PAC learning algorithms that use polynomially evaluatable hypothesis classes $\mathcal{H}$ (see Definition 5). Suppose that we consider relaxing this restriction, and let $\mathcal{H}$ be the class of all Turing machines (not necessarily polynomial time) — thus, the output of the learning algorithm can be any program. Show that if $\mathcal{C}_n$ is the class of all boolean circuits of size at most $p(n)$ for some fixed polynomial $p(\cdot)$, then $\mathcal{C}$ is efficiently PAC learnable using $\mathcal{H}$. Argue that your solution shows that this relaxation trivializes the model of learning.

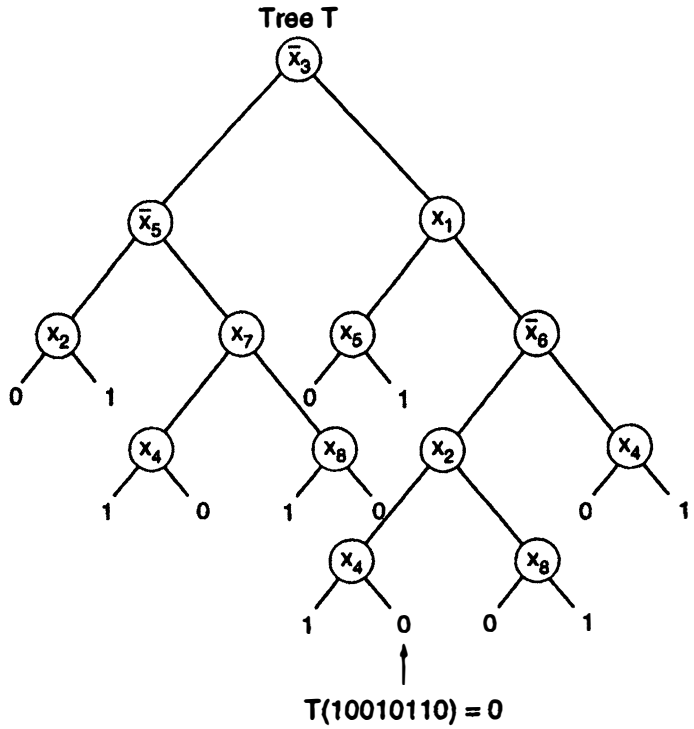

Figure 2.2: *A decision tree and the path followed by an input.*

2.6 Bibliographic Notes

The notion of Occam learning as we have formalized it and our main theorems stating that Occam learning implies PAC learning are due to Blumer, Ehrenfeucht, Haussler and Warmuth [21]. There is a converse to Theorem 2.1 which establishes that C is PAC learnable if and only if there is an Occam algorithm for C. This was the topic of Exercise 2.3, whose intended solution is due to Board and Pitt [23]. A considerably stronger converse is a consequence of the equivalence between weak and strong PAC learning due to Schapire [84, 85] (see also the work of Freund [35, 36]

and Helmbold and Warmuth [52]). We shall study this equivalence in Chapter 4.

The predictive power of Occam algorithms continues to hold for several variants of the PAC model and for more general notions of hypothesis complexity. These include models for PAC learning in the presence of various types of errors (Angluin and Laird [10], Kearns and Li [57, 55]), learning probabilistic concepts (Kearns and Schapire [61, 85]), and function learning (Natarajan [70]). In Chapter 3 we will consider a very general notion of hypothesis complexity, the Vapnik-Chervonenkis dimension (Vapnik [94]; Blumer, Ehrenfeucht, Haussler and Warmuth [22]), and we again prove the predictive power of algorithms finding a consistent hypothesis with limited complexity. The predictive power of Occam algorithms in a setting where the examples are not independent but obey a Markovian constraint is examined by Aldous and Vazirani [3].

The algorithm for learning conjunctions with few relevant literals is due to Haussler [45], who also provides a lucid discussion of Occam learning and inductive bias from the artificial intelligence perspective. The analysis of the greedy set cover approximation algorithm is due to Chvatal [26]. The modification of the covering algorithm to only nearly cover the sample is due to M. Warmuth. The problem of learning when there are many irrelevant variables present has also been carefully examined by Littlestone [65, 66] and Blum [17] in on-line models of learning. The decision list learning algorithm is due to Rivest [78], and Exercise 2.2 is due to A. Blum (see also the paper Ehrenfeucht and Haussler [32]).

Relationships between various measures of hypothesis complexity and generalization ability have been proposed and examined in a a large and fascinating literature that predates the PAC model results given here. Two dominant theories along these lines are the structural risk minimization of Vapnik [94] and the minimum description length principle of Rissanen [77]. The papers of Quinlan and Rivest [75] and DeSantis, Markowsky and Wegman [29] examine variants of the minimum description length principle from a computational learning theory viewpoint. It has frequently been observed that the minimum description length

criterion has a Bayesian interpretation in which representational length determines the prior distribution. This viewpoint is further explored in the paper of Evans, Rajagopalan and Vazirani [34], where the notion of an Occam algorithm is generalized to arbitrary stochastic processes.