1

The Probably Approximately Correct Learning Model

1.1 A Rectangle Learning Game

Consider a simple one-player learning game. The object of the game is to learn an unknown axis-aligned rectangle R — that is, a rectangle in the Euclidean plane $\Re^2$ whose sides are parallel with the coordinate axes. We shall call R the **target** rectangle. The player receives information about R only through the following process: every so often, a random point p is chosen in the plane according to some fixed probability distribution $\mathcal{D}$. The player is given the point p together with a label indicating whether p is contained in R (a positive example) or not contained in R (a negative example). Figure 1.1 shows the unknown rectangular region R along with a sample of positive and negative examples.

The goal of the player is to use as few examples as possible, and as little computation as possible, to pick a **hypothesis** rectangle R' which is a close approximation to R. Informally, the player's knowledge of R is tested by picking a new point at random from the same probability distribution $\mathcal{D}$, and checking whether the player can correctly decide whether the point falls inside or outside of R. Formally, we measure the

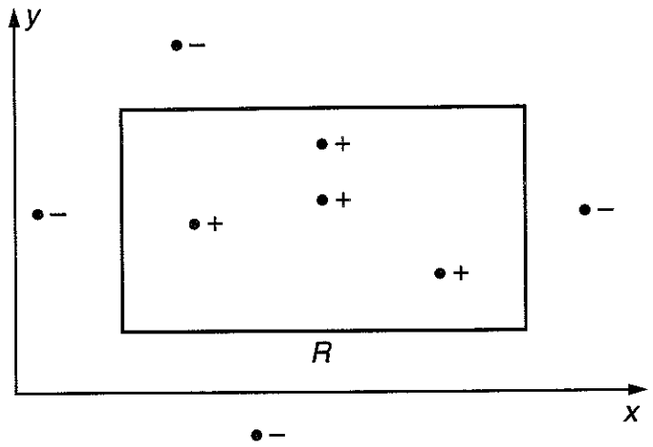

Figure 1.1: *The target rectangle R in the plane along with a sample of positive and negative examples.*

error of R' as the probability that a randomly chosen point from $\mathcal{D}$ falls in the region $R \Delta R'$, where $R \Delta R' = (R - R') \cup (R' - R)$.

To motivate the rectangle learning game, consider a slightly more concrete scenario that can be expressed as an instance of the game. Suppose that we wanted to learn the concept of "men of medium build". Assume that a man is of medium build if his height and weight both lie in some prescribed ranges — for instance, if his height is between five feet six inches and six feet, and his weight is between 150 pounds and 200 pounds. Then each man's build can be represented by a point in the Euclidean plane, and the concept of medium build is represented by an axis-aligned rectangular region of the plane. Thus, during an initial training phase, the learner is told for each new man he meets whether that man is of medium build or not. Over this period, the learner must form some model or hypothesis of the concept of medium build.

Now assume that the learner encounters every man in his city with

equal probability. Even under this assumption, the corresponding points in the plane may not be uniformly distributed (since not all heights and weights are equally likely, and height and weight may be highly dependent quantities), but will instead obey some fixed distribution $\mathcal{D}$ which may be quite difficult to characterize. For this reason, in our learning game, we allow the distribution $\mathcal{D}$ to be arbitrary, but we assume that it is fixed, and that each example is drawn independently from this distribution. (Note that once we allow $\mathcal{D}$ to be arbitrary, we no longer need to assume that the learner encounters every man in his city with equal probability.) To evaluate the hypothesis of the learner, we are simply evaluating its success in classifying the build of men in future encounters, still assuming that men are encountered according to the same probability distribution as during the training phase.

There is a simple and efficient strategy for the player of the rectangle learning game. The strategy is to request a "sufficiently large" number m of random examples, then choose as the hypothesis the axis-aligned rectangle R' which gives the tightest fit to the positive examples (that is, that rectangle with the smallest area that includes all of the positive examples and none of the negative examples). If no positive examples are drawn, then $R' = \emptyset$. Figure 1.2 shows the tightest-fit rectangle defined by the sample shown in Figure 1.1.

We will now show that for any target rectangle R and any distribution $\mathcal{D}$, and for any small values ϵ and δ ($0 < \epsilon, \delta \leq 1/2$), for a suitably chosen value of the sample size m we can assert that with probability at least $1 - \delta$, the tightest-fit rectangle has error at most ϵ with respect to R and $\mathcal{D}$.

First observe that the tightest-fit rectangle R' is always contained in the target rectangle R (that is, $R' \subseteq R$ and so $R \Delta R' = R - R'$). We can express the difference $R - R'$ as the union of four rectangular strips. For instance, the topmost of these strips, which is shaded and denoted T' in Figure 1.3, is the region above the upper boundary of R' extended to the left and right, but below the upper boundary of R. Note that there is some overlap between these four rectangular strips at the corners. Now

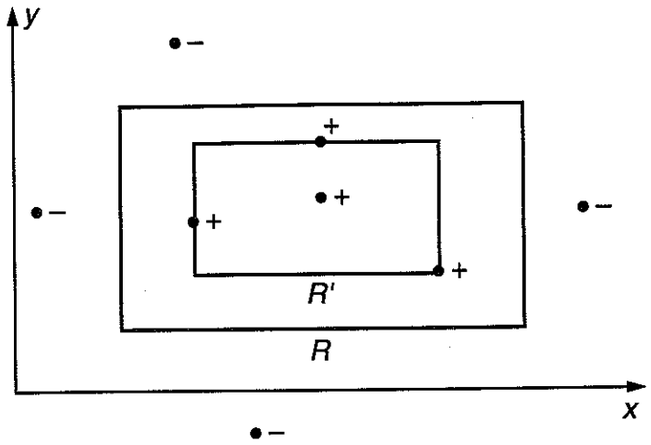

Figure 1.2: *The tightest-fit rectangle R' defined by the sample.*

if we can guarantee that the weight under $\mathcal{D}$ of each strip (that is, the probability with respect to $\mathcal{D}$ of falling in the strip) is at most $\epsilon/4$, then we can conclude that the error of R' is at most $4(\epsilon/4) = \epsilon$. (Here we have erred on the side of pessimism by counting each overlap region twice.)

Let us analyze the weight of the top strip T'. Define T to be the rectangular strip along the inside top of R which encloses *exactly* weight $\epsilon/4$ under $\mathcal{D}$ (thus, we sweep the top edge of R downwards until we have swept out weight $\epsilon/4$; see Figure 1.3). Clearly, T' has weight exceeding $\epsilon/4$ under $\mathcal{D}$ if and only if T' includes T (which it does not in Figure 1.3). Furthermore, T' includes T if and only if no point in T appears in the sample S — since if S does contain a point $p \in T$, this point has a positive label since it is contained in R, and then by definition of the tightest fit, the hypothesis rectangle R' must extend upwards into T to cover p.

By the definition of T, the probability that a single draw from the distribution $\mathcal{D}$ misses the region T is exactly $1 - \epsilon/4$. Therefore the

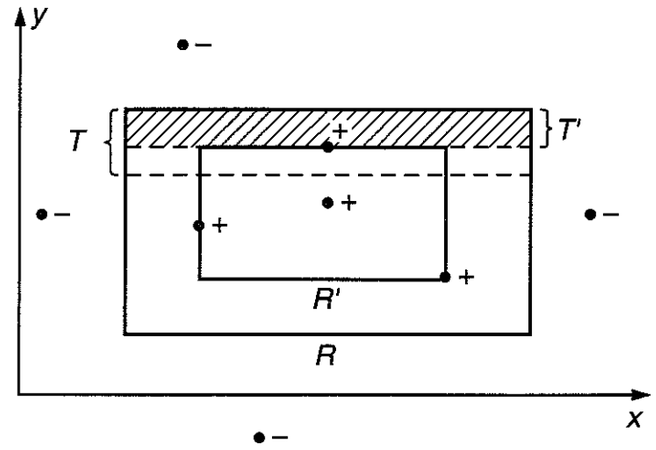

Figure 1.3: *Analysis of the error contributed by the top shaded strip T'. The strip T has weight exactly $\epsilon/4$ under $\mathcal{D}$.*

probability that m independent draws from $\mathcal{D}$ all miss the region T is exactly $(1 - \epsilon/4)^m$. Here we are using the fact that the probability of a conjunction of independent events is simply the product of the probabilities of the individual events. The same analysis holds for the other three rectangular regions of $R - R'$, so by the **union bound**, the probability that any of the four strips of $R - R'$ has weight greater than $\epsilon/4$ is at most $4(1 - \epsilon/4)^m$. By the union bound, we mean the fact that if A and B are any two events (that is, subsets of a probability space), then

$$\mathbf{Pr}[A \cup B] \leq \mathbf{Pr}[A] + \mathbf{Pr}[B].$$

Thus, the probability that one of the four error strips has weight exceeding $\epsilon/4$ is at most four times the probability that a fixed error strip has weight exceeding $\epsilon/4$.

Provided that we choose m to satisfy $4(1 - \epsilon/4)^m \leq \delta$, then with probability $1 - \delta$ over the m random examples, the weight of the error

region $R - R'$ will be bounded by ϵ, as claimed. Using the inequality

$$(1 - x) \leq e^{-x}$$

(which we shall appeal to frequently in our studies) we see that any value of m satisfying $4e^{-\epsilon m/4} \leq \delta$ also satisfies the previous condition. Dividing by 4 and taking natural logarithms of both sides gives $-\epsilon m/4 \leq \ln(\delta/4)$, or equivalently $m \geq (4/\epsilon)\ln(4/\delta)$.

In summary, provided our tightest-fit algorithm takes a sample of at least $(4/\epsilon)\ln(4/\delta)$ examples to form its hypothesis rectangle R', we can assert that with probability at least $1 - \delta$, R' will misclassify a new point (drawn according to the same distribution from which the sample was chosen) with probability at most ϵ.

A few brief comments are appropriate. First, note that the analysis really does hold for any fixed probability distribution. We only needed the independence of successive points to obtain our bound. Second, the sample size bound behaves as we might expect, in that as we increase our demands on the hypothesis rectangle — that is, as we ask for greater **accuracy** by decreasing ϵ or greater **confidence** by decreasing δ — our algorithm requires more examples to meet those demands. Finally, the algorithm we have analyzed is efficient: the required sample size is a slowly growing function of $1/\epsilon$ and $1/\delta$ (linear and logarithmic, respectively), and once the sample is given, the computation of the tightest-fit hypothesis can be carried out rapidly.

1.2 A General Model

In this section, we introduce the model of learning that will be the central object for most of our study: the **Probably Approximately Correct** or **PAC** model of learning. There are a number of features of the rectangle learning game and its solution that are essential to the PAC model, and bear highlighting before we dive into the general definitions.

- The goal of the learning game is to learn an unknown target set, but the target set is not arbitrary. Instead, there is a known and rather strong constraint on the target set — it is a rectangle in the plane whose sides are parallel to the axes.

- Learning occurs in a probabilistic setting. Examples of the target rectangle are drawn randomly in the plane according to a fixed probability distribution which is unknown and unconstrained.

- The hypothesis of the learner is evaluated relative to the *same* probabilistic setting in which the training takes place, and we allow hypotheses that are only approximations to the target. The tightest-fit strategy might not find the target rectangle exactly, but will find one with only a small probability of disagreement with the target.

- We are interested in a solution that is efficient: not many examples are required to obtain small error with high confidence, and we can process those examples rapidly.

We wish to state a general model of learning from examples that shares and formalizes the properties we have listed. We begin by developing and motivating the necessary definitions.

1.2.1 Definition of the PAC Model

Let X be a set called the **instance space**. We think of X as being a set of encodings of instances or objects in the learner's world. In our rectangle game, the instance space X was simply the set of all points in the Euclidean plane $\Re^2$. As another example, in a character recognition application, the instance space might consist of all 2-dimensional arrays of binary pixels of a given width and height.

A **concept** over X is just a subset $c \subseteq X$ of the instance space. In the rectangle game, the concepts were axis-aligned rectangular regions.

Continuing our character recognition example, a natural concept might be the set of all pixel arrays that are representations of the letter "A" (assuming that every pixel array either represents an "A", or fails to represent an "A").

A concept can thus be thought of as the set of all instances that positively exemplify some simple or interesting rule. We can equivalently define a concept to be a boolean mapping $c : X \to \{0,1\}$, with $c(x) = 1$ indicating that x is a positive example of c and $c(x) = 0$ indicating that x is a negative example. For this reason, we also sometimes call X the **input space**.

A **concept class** $\mathcal{C}$ over X is a collection of concepts over X. In the rectangle game, the target rectangle was chosen from the class $\mathcal{C}$ of all axis-aligned rectangles. Ideally, we are interested in concept classes that are sufficiently expressive for fairly general knowledge representation. As an example in a logic-based setting, suppose we have a set $x_1, \ldots, x_n$ of n boolean variables, and let X be the set of all assignments to these variables (that is, $X = \{0,1\}^n$). Suppose we consider concepts c over $\{0,1\}^n$ whose positive examples are exactly the satisfying assignments of some boolean formulae f_c over $x_1, \ldots, x_n$. Then we might define an interesting concept class $\mathcal{C}$ by considering only those boolean formulae f_c that meet some natural syntactic constraints, such as being in disjunctive normal form (DNF) and having a small number of terms.

In our model, a learning algorithm will have access to positive and negative examples of an unknown **target concept** c, chosen from a known concept class $\mathcal{C}$. The learning algorithm will be judged by its ability to identify a hypothesis concept that can accurately classify instances as positive or negative examples of c. Before specifying the learning protocol further, it is important to note that in our model, learning algorithms "know" the target class $\mathcal{C}$, in the sense that the designer of the learning algorithm is guaranteed that the target concept will be chosen from $\mathcal{C}$ (but must design the algorithm to work for any $c \in \mathcal{C}$).

Let $\mathcal{D}$ be any fixed probability distribution over the instance space X.

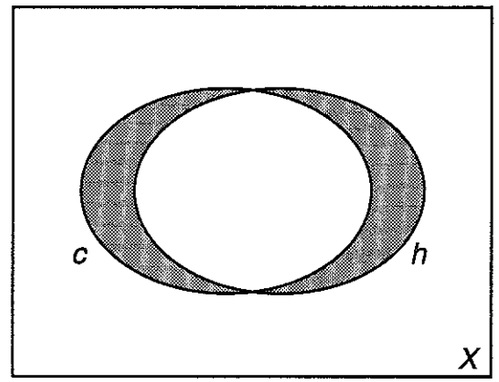

Figure 1.4: *Venn diagram of two concepts, with symmetric difference shaded.*

We will refer to $\mathcal{D}$ as the **target distribution**. If h is any concept over X, then the distribution $\mathcal{D}$ provides a natural measure of **error** between h and the target concept c: namely, we define

$$error(h) = \mathbf{Pr}_{x \in \mathcal{D}}[c(x) \neq h(x)].$$

Here we regard the concepts c and h as boolean functions, and we have introduced a notational convention that we shall use frequently: the subscript $x \in \mathcal{D}$ to $\mathbf{Pr}[\cdot]$ indicates that the probability is taken with respect to the random draw of x according to $\mathcal{D}$. Note that $error(h)$ has an implicit dependence on c and $\mathcal{D}$ that we will usually omit for brevity when no confusion will result.

A useful alternative way to view $error(h)$ is represented in Figure 1.4. Here we view the concepts c and h as sets rather than as functions, and we have drawn an abstract Venn diagram showing the positive examples of c and h, which of course lie within the entire instance space X. Then $error(h)$ is simply the probability with respect to $\mathcal{D}$ that an instance is drawn falling in the shaded region.

Let $EX(c, \mathcal{D})$ be a procedure (we will sometimes call it an *oracle*) that

runs in unit time, and on each call returns a labeled example $\langle x, c(x)\rangle$, where x is drawn randomly and independently according to $\mathcal{D}$. A learning algorithm will have access to this oracle when learning the target concept $c \in \mathcal{C}$. Ideally, the learning algorithm will satisfy three properties:

- The number of calls to $EX(c, \mathcal{D})$ is small, in the sense that it is bounded by a fixed polynomial in some parameters to be specified shortly.
- The amount of computation performed is small.
- The algorithm outputs a **hypothesis concept** h such that $error(h)$ is small.

Note that the number of calls made by a learning algorithm to $EX(c, \mathcal{D})$ is bounded by the running time of the learning algorithm.

We are now ready to give the definition of Probably Approximately Correct learning. We designate it as our preliminary definition, since we shall soon make some important additions to it.

Definition 1 *(The PAC Model, Preliminary Definition) Let $\mathcal{C}$ be a concept class over X. We say that $\mathcal{C}$ is* **PAC learnable** *if there exists an algorithm L with the following property: for every concept $c \in \mathcal{C}$, for every distribution $\mathcal{D}$ on X, and for all $0 < \epsilon < 1/2$ and $0 < \delta < 1/2$, if L is given access to $EX(c, \mathcal{D})$ and inputs ϵ and δ, then with probability at least $1 - \delta$, L outputs a hypothesis concept $h \in \mathcal{C}$ satisfying $error(h) \leq \epsilon$. This probability is taken over the random examples drawn by calls to $EX(c, \mathcal{D})$, and any internal randomization of L.*

If L runs in time polynomial in $1/\epsilon$ and $1/\delta$, we say that $\mathcal{C}$ is **efficiently** *PAC learnable. We will sometimes refer to the input ϵ as the* **error parameter**, *and the input δ as the* **confidence parameter**.

The hypothesis $h \in \mathcal{C}$ of the PAC learning algorithm is thus "approximately correct" with high probability, hence the name Probably Approximately Correct learning.

Two important comments regarding the PAC learning model are now in order. First, the error and confidence parameters ϵ and δ control the two types of failure to which a learning algorithm in the PAC model is inevitably susceptible. The error parameter ϵ is necessary since, for example, there may be only a negligible probability that a small random sample will distinguish between two competing hypotheses that differ on only one improbable point in the instance space. The confidence parameter δ is necessary since the learning algorithm may occasionally be extremely unlucky, and draw a terribly "unrepresentative" sample of the target concept — for instance, a sample consisting only of repeated draws of the same instance despite the fact that the distribution is spread evenly over all instances. The best we can hope for is that the probability of both types of failure can be made arbitrarily small at a modest cost.

Second, notice that we demand that a PAC learning algorithm perform well with respect to any distribution $\mathcal{D}$. This strong requirement is moderated by the fact that we only evaluate the hypothesis of the learning algorithm with respect to the same distribution $\mathcal{D}$. For example, in the rectangle learning game discussed earlier, this means that if the distribution gives negligible weight to some parts of the Euclidean plane, then the learner does not have to be very careful in learning the boundary of the target rectangle in that region.

Definition 1, then, is our tentative definition of PAC learning, which will be the model forming the bulk of our studies. As previously mentioned, we shall make a couple of important refinements to this definition before we begin the serious investigation. Before doing so, however, we pause to note that we have already proven our first result in this model. Recall that our algorithm for the rectangle learning game required the ability to store real numbers and perform basic operations on them, such as comparisons. In the following theorem, and throughout our study, whenever necessary we will assume a model of computation that allows

storage of a single real number in a single memory location, and that charges one unit of computation time for a basic arithmetic operation (addition, multiplication or division) on two real numbers.

Theorem 1.1 *The concept class of axis-aligned rectangles over the Euclidean plane $\Re^2$ is efficiently PAC learnable.*

1.2.2 Representation Size and Instance Dimension

An important issue was swept under the rug in our definition of PAC learning. This is the fundamental distinction between a concept (which is just a set or a boolean function) and its representation (which is a symbolic encoding of that set or function). Consider a class of concepts defined by the satisfying assignments of boolean formulae. A concept from this class — that is, the set of satisfying assignments for some boolean formula f — can be represented by the formula f, by a truth table, or by another boolean formula f' that is logically equivalent to f. Although all of these are representations of the same underlying concept, they may differ radically in representational size.

For instance, it is not hard to prove that for all n, the boolean parity function $f(x_1, \ldots, x_n) = x_1 \oplus \cdots \oplus x_n$ (where $\oplus$ denotes the exclusive-or operation) can be computed by a circuit of $\wedge$, $\vee$ and $\neg$ gates whose size is bounded by a fixed polynomial in n, but to represent this same function as a disjunctive normal form (abbreviated DNF) formula requires size exponential in n. As another example, in high-dimensional Euclidean space $\Re^n$, we may choose to represent a convex polytope either by specifying its vertices, or by specifying linear equations for its faces, and these two representation schemes can differ exponentially in size.

In each of these examples, we are fixing some representation scheme — that is, a precise method for encoding concepts — and then examining

the size of the encoding for various concepts. Other natural representation schemes that the reader may be familiar with include decision trees and neural networks. As with boolean formulae, in these representation schemes there is an obvious mapping from the representation (a decision tree or a neural network) to the set or boolean function that is being represented. There is also a natural measure of the size of a given representation in the scheme (for instance, the number of nodes in the decision tree or the number of weights in a neural network).

Since a PAC learning algorithm only sees examples of the functional (that is, input-output) behavior of the target concept, it has absolutely no information about which, if any, of the many possible representations is actually being used to represent the target concept in reality. However, it matters greatly which representation the algorithm chooses for its hypothesis, since the time to write this representation down is obviously a lower bound on the running time of the algorithm.

Formally speaking, a **representation scheme** for a concept class $\mathcal{C}$ is a function $\mathcal{R} : \Sigma^* \to \mathcal{C}$, where Σ is a finite alphabet of symbols. (In cases where we need to use real numbers to represent concepts, such as axis-aligned rectangles, we allow $\mathcal{R} : (\Sigma \cup \Re)^* \to \mathcal{C}$.) We call any string $\sigma \in \Sigma^*$ such that $\mathcal{R}(\sigma) = c$ a **representation** of c (under $\mathcal{R}$). Note that there may be many representations of a concept c under the representation scheme $\mathcal{R}$.

To capture the notion of representation size, we assume that associated with $\mathcal{R}$ there is a mapping $size : \Sigma^* \to N$ that assigns a natural number $size(h)$ to each representation $h \in \Sigma^*$. Note that we allow $size(\cdot)$ to be any such mapping; results obtained under a particular definition for $size(\cdot)$ will be meaningful only if this definition is natural. Perhaps the most realistic setting, however, is that in which $\Sigma = \{0, 1\}$ (thus, we have a binary encoding of concepts) and we define $size(h)$ to be the length of h in bits. (For representations using real numbers, it is often natural to charge one unit of size for each real number.) Although we will use other definitions of size when binary representations are inconvenient, our definition of $size(\cdot)$ will always be within a polynomial factor

of the binary string length definition. For example, we can define the size of a decision tree to be the number of nodes in the tree, which is always within a polynomial factor of the length of the binary string needed to encode the tree in any reasonable encoding method.

So far our notion of size is applicable only to representations (that is, to strings $h \in \Sigma^*$). We would like to extend this definition to measure the size of a target concept $c \in \mathcal{C}$. Since the learning algorithm has access only to the input-output behavior of c, in the worst case it must assume that the simplest possible mechanism is generating this behavior. Thus, we define $size(c)$ to be $size(c) = \min_{\mathcal{R}(\sigma)=c}\{size(\sigma)\}$. In other words, $size(c)$ is the size of the smallest representation of the concept c in the underlying representation scheme $\mathcal{R}$. Intuitively, the larger $size(c)$ is, the more "complex" the concept c is with respect to the chosen representation scheme. Thus it is natural to modify our notion of learning to allow more computation time for learning more complex concepts, and we shall do this shortly.

For a concept class $\mathcal{C}$, we shall refer to the **representation class** $\mathcal{C}$ to indicate that we have in mind some fixed representation scheme $\mathcal{R}$ for $\mathcal{C}$. In fact, we will usually *define* the concept classes we study by their representation scheme. For instance, we will shortly examine the concept class in which each concept is the set of satisfying assignments of some conjunction of boolean variables. Thus, each concept can be represented by a list of the variables in the associated conjunction.

It is often convenient to also introduce some notion of size or dimension for the elements of the instance space. For example, if the instance space X_n is the n-dimensional Euclidean space $\Re^n$, then each example is specified by n real numbers, and so it is natural to say that the size of the examples is n. The same comments apply to the instance space $X_n = \{0,1\}^n$. It turns out that these are the only two instance spaces that we will ever need to consider in our studies, and in the spirit of asymptotic analysis we will want to regard the instance space dimension n as a parameter of the learning problem (for example, to allow us to study the problem of learning axis-aligned rectangles in $\Re^n$ in time polynomial in n). Now if we let $\mathcal{C}_n$ be the class of concepts over X_n, and write $X = \cup_{n\geq 1} X_n$ and $\mathcal{C} = \cup_{n\geq 1}\mathcal{C}_n$, then X and $\mathcal{C}$ define an infinite family of learning problems of increasing dimension.

To incorporate the notions of target concept size and instance space dimension into our model, we make the following refined definition of PAC learning:

Definition 2 *(The PAC Model, Modified Definition) Let $\mathcal{C}_n$ be a representation class over X_n (where X_n is either $\{0,1\}^n$ or n-dimensional Euclidean space $\Re^n$), and let $X = \cup_{n\geq 1} X_n$ and $\mathcal{C} = \cup_{n\geq 1}\mathcal{C}_n$. The modified definition of PAC learning is the same as the preliminary definition (Definition 1), except that now we allow the learning algorithm time polynomial in n and $size(c)$ (as well as $1/\epsilon$ and $1/\delta$ as before) when learning a target concept $c \in \mathcal{C}_n$.*

Since in our studies X_n will always be either $\{0,1\}^n$ or n-dimensional Euclidean space, the value n is implicit in the instances returned by $EX(c,\mathcal{D})$. We assume that the learner is provided with the value $size(c)$ as an input. (However, see Exercise 1.5.)

We emphasize that while the target concept may have many possible representations in the chosen scheme, we only allow the learning algorithm time polynomial in the size of the smallest such representation. This provides a worst-case guarantee over the possible representations of c, and is consistent with the fact that the learning algorithm has no idea which representation is being used for c, having only functional information about c.

Finally, we note that for several concept classes the natural definition of $size(c)$ is already bounded by a polynomial in n, and thus we really seek an algorithm running in time polynomial in just n. For instance, if we look at the representation class of all DNF formulae with at most 3 terms, any such formula has length at most $3n$, so polynomial dependence

on the size of the target formula is the same as polynomial dependence on n.

1.3 Learning Boolean Conjunctions

We now give our second result in the PAC model, showing that **conjunctions of boolean literals** are efficiently PAC learnable. Here the instance space is $X_n = \{0,1\}^n$. Each $a \in X_n$ is interpreted as an assignment to the n boolean variables $x_1, \ldots, x_n$, and we use the notation a_i to indicate the ith bit of a. Let the representation class $\mathcal{C}_n$ be the class of all conjunctions of literals over $x_1, \ldots, x_n$ (a **literal** is either a variable x_i or its negation $\overline{x}_i$). Thus the conjunction $x_1 \wedge \overline{x}_3 \wedge x_4$ represents the set $\{a \in \{0,1\}^n : a_1 = 1, a_3 = 0, a_4 = 1\}$. It is natural to define the size of a conjunction to be the number of literals in that conjunction. Then clearly $size(c) \leq 2n$ for any conjunction $c \in \mathcal{C}_n$. (We also note that a standard binary encoding of any conjunction $c \in \mathcal{C}_n$ has length $O(n \log n)$.) Thus for this problem, we seek an algorithm that runs in time polynomial in n, $1/\epsilon$ and $1/\delta$.

Theorem 1.2 *The representation class of conjunctions of boolean literals is efficiently PAC learnable.*

Proof: The algorithm we propose begins with the hypothesis conjunction

$$h = x_1 \wedge \overline{x}_1 \wedge \cdots \wedge x_n \wedge \overline{x}_n.$$

Note that initially h has no satisfying assignments. The algorithm simply ignores any negative examples returned by $EX(c, \mathcal{D})$. Let $\langle a, 1 \rangle$ be a positive example returned by $EX(c, \mathcal{D})$. In response to such a positive example, our algorithm updates h as follows: for each i, if $a_i = 0$, we delete x_i from h, and if $a_i = 1$, we delete $\overline{x}_i$ from h. Thus, our algorithm deletes any literal that "contradicts" the positive data.

For the analysis, note that the set of literals appearing in h at any time always contains the set of literals appearing in the target concept c. This is because we begin with h containing all literals, and a literal is only deleted from h when it is set to 0 in a positive example; such a literal clearly cannot appear in c. The fact that the literals of h always include those of c implies that h will never err on a negative example of c (that is, h is more specific than c).

Thus, consider a literal z that occurs in h but not in c. Then z causes h to err only on those positive examples of c in which $z = 0$; also note that it is exactly such positive examples that would have caused our algorithm to delete z from h. Let $p(z)$ denote the total probability of such instances under the distribution $\mathcal{D}$, that is,

$$p(z) = \mathbf{Pr}_{a \in \mathcal{D}}[c(a) = 1 \wedge z \text{ is 0 in } a].$$

Since every error of h can be "blamed" on at least one literal z of h, by the union bound we have $error(h) \leq \sum_{z \in h} p(z)$. We say that a literal is *bad* if $p(z) \geq \epsilon/2n$. If h contains no bad literals, then $error(h) \leq \sum_{z \in h} p(z) \leq 2n(\epsilon/2n) = \epsilon$. We now upper bound the probability that a bad literal will appear in h.

For any *fixed* bad literal z, the probability that this literal is not deleted from h after m calls of our algorithm to $EX(c, \mathcal{D})$ is at most $(1 - \epsilon/2n)^m$, because the probability the literal z is deleted by a single call to $EX(c, \mathcal{D})$ is $p(z)$ (which is at least $\epsilon/2n$ for a bad literal). From this we may conclude that the probability that there is *some* bad literal that is not deleted from h after m calls is at most $2n(1 - \epsilon/2n)^m$, where we have used the union bound over the $2n$ possible literals.

Thus to complete our analysis we simply need to solve for the value of m satisfying $2n(1 - \epsilon/2n)^m \leq \delta$, where $1 - \delta$ is the desired confidence. Using the inequality $1 - x \leq e^{-x}$, it suffices to pick m such that $2ne^{-m\epsilon/2n} \leq \delta$, which yields $m \geq (2n/\epsilon)(\ln(2n) + \ln(1/\delta))$.

Thus, if our algorithm takes at least this number of examples, then with probability at least $1 - \delta$ the resulting conjunction h will have error

at most ϵ with respect to c and $\mathcal{D}$. Since the algorithm takes linear time to process each example, the running time is bounded by mn, and hence is bounded by a polynomial in n, $1/\epsilon$ and $1/\delta$, as required. □(Theorem 1.2)

1.4 Intractability of Learning 3-Term DNF Formulae

We next show that a slight generalization of the representation class of boolean conjunctions results in an intractable PAC learning problem. More precisely, we show that the class of disjunctions of three boolean conjunctions (known as **3-term disjunctive normal form (DNF) formulae**) is not efficiently PAC learnable unless every problem in NP can be efficiently solved in a worst-case sense by a randomized algorithm — that is, unless for every language A in NP there is a randomized algorithm taking as input any string α and a parameter $\delta \in [0,1]$, and that with probability at least $1-\delta$ correctly determines whether $\alpha \in A$ in time polynomial in the length of α and $1/\delta$. The probability here is taken only with respect to the coin flips of the randomized algorithm. In technical language, our hardness result for 3-term DNF is based on the widely believed assumption that $RP \neq NP$.

The representation class $\mathcal{C}_n$ of 3-term DNF formulae is the set of all disjunctions $T_1 \vee T_2 \vee T_3$, where each T_i is a conjunction of literals over the boolean variables $x_1, \ldots, x_n$. We define the size of such a representation to be sum of the number of literals appearing in each term (which is always bounded by a fixed polynomial in the length of the bit string needed to represent the 3-term DNF in a standard encoding). Then $size(c) \leq 6n$ for any concept $c \in \mathcal{C}_n$ because there are at most $2n$ literals in each of the three terms. Thus, an efficient learning algorithm for this problem is required to run in time polynomial in n, $1/\epsilon$ and $1/\delta$.

Theorem 1.3 *If $RP \neq NP$, the representation class of 3-term DNF formulae is not efficiently PAC learnable.*

Proof: The high-level idea of the proof is to reduce an NP-complete language A (to be specified shortly) to the problem of PAC learning 3-term DNF formulae. More precisely, the reduction will efficiently map any string α, for which we wish to determine membership in A, to a set S_α of labeled examples. The cardinality $|S_\alpha|$ will be bounded by a polynomial in the string length $|\alpha|$. We will show that given a PAC learning algorithm L for 3-term DNF formulae, we can run L on S_α, in a manner to be described, to determine (with high probability) if α belongs to A or not.

The key property we desire of the mapping of α to S_α is that $\alpha \in A$ if and only if S_α is **consistent** with some concept $c \in \mathcal{C}$. The notion of a concept being consistent with a sample will recur frequently in our studies.

Definition 3 *Let $S = \{\langle x_1, b_1\rangle, \ldots, \langle x_m, b_m\rangle\}$ be any labeled set of instances, where each $x_i \in X$ and each $b_i \in \{0,1\}$. Let c be a concept over X. Then we say that c is* **consistent** *with S (or equivalently, S is consistent with c) if for all $1 \leq i \leq m$, $c(x_i) = b_i$.*

Before detailing our choice for the NP-complete language A and the mapping of α to S_α, just suppose for now that we have managed to arrange things so that $\alpha \in A$ if and only if S_α is consistent with some concept in $\mathcal{C}$. We now show how a PAC learning algorithm L for $\mathcal{C}$ can be used to *determine* if there exists a concept in $\mathcal{C}$ that is consistent with S_α (and thus whether $\alpha \in A$) with high probability. This is achieved by the following general method: we set the error parameter $\epsilon = 1/(2|S_\alpha|)$ (where $|S_\alpha|$ denotes the number of labeled pairs in S_α), and answer each request of L for a random labeled example by choosing a pair $\langle x_i, b_i\rangle$ uniformly at random from S_α. Note that if there is a concept $c \in \mathcal{C}$ consistent with S_α, then this simulation emulates the oracle $EX(c, \mathcal{D})$, where $\mathcal{D}$ is uniform over the (multiset of) instances appearing in S_α. In

this case, by our choice of ϵ, we have guaranteed that any hypothesis h with error less that ϵ must in fact be consistent with S_α, for if h errs on even a single example in S_α, its error with respect to c and $\mathcal{D}$ is at least $1/|S_\alpha| = 2\epsilon > \epsilon$. On the other hand, if there is no concept in $\mathcal{C}$ consistent with S_α, L cannot possibly find one. Thus we can simply check the output of L for consistency with S_α to determine with confidence $1 - \delta$ if there exists a consistent concept in $\mathcal{C}$.

Combined with the assumed mapping of a string α to a set S_α, we thus can determine (with probability at least $1 - \delta$) the membership of α in A by simulating the PAC learning algorithm on S_α. This general method of using a PAC learning algorithm to determine the existence of a concept that is consistent with a labeled sample is quite common in the computational learning theory literature, and the main effort comes in choosing the right *NP*-complete language A, and finding the desired mapping from instances α of A to sets of labeled examples S_α, which we now undertake.

To demonstrate the intractability of learning 3-term DNF formulae, the *NP*-complete language A that we shall use is Graph 3-Coloring:

The Graph 3-Coloring Problem. Given as input an undirected graph $G = (V, E)$ with vertex set $V = \{1, \ldots, n\}$ and edge set $E \subseteq V \times V$, determine if there is an assignment of a color to each element of V such that at most 3 different colors are used, and for every edge $(i, j) \in E$, vertex i and vertex j are assigned different colors.

We now describe the desired mapping from an instance $G = (V, E)$ of Graph 3-Coloring to a set S_G of labeled examples. S_G will consist of a set S_G^+ of positively labeled examples and a set S_G^- of negatively labeled examples, so $S_G = S_G^+ \cup S_G^-$. For each $1 \leq i \leq n$, S_G^+ will contain the labeled example $\langle v(i), 1\rangle$, where $v(i) \in \{0,1\}^n$ is the vector with a 0 in the ith position and 1's everywhere else. These examples intuitively encode the vertices of G. For each edge $(i, j) \in E$, the set S_G^- will contain the labeled example $\langle e(i, j), 0\rangle$, where $e(i, j) \in \{0,1\}^n$ is the vector with 0's in the ith and jth positions, and 1's everywhere else. Figure 1.5 shows

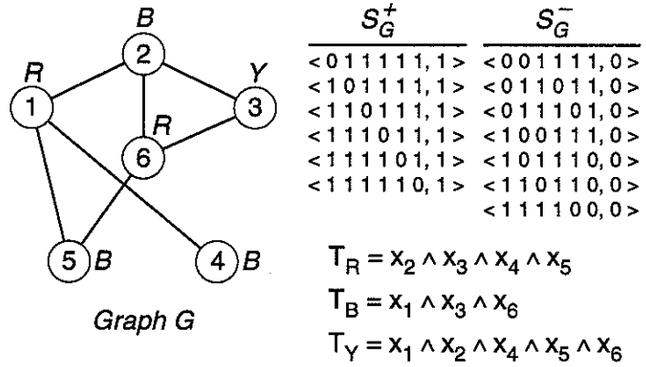

Figure 1.5: *A graph G with a legal 3-coloring, the associated sample, and the terms defined by the coloring.*

an example of a graph G along with the resulting sets S_G^+ and S_G^-. The figure also shows a legal 3-coloring of G, with R, B and Y denoting red, blue and yellow.

We now argue that G is 3-colorable if and only if S_G is consistent with some 3-term DNF formula. First, suppose G is 3-colorable and fix a 3-coloring of G. Let R be the set of all vertices colored red, and let T_R be the conjunction of all variables in $x_1, \ldots, x_n$ whose index does *not* appear in R (see Figure 1.5). Then for each $i \in R$, $v(i)$ must satisfy T_R because the variable x_i does not appear in T_R. Furthermore, no $e(i, j) \in S_G^-$ can satisfy T_R because since both i and j cannot be colored red, one of x_i and x_j must appear in T_R. We can define terms that are satisfied by the non-blue and non-yellow $v(i)$ in a similar fashion, with no negative examples being accepted by any term.

For the other direction, suppose that the formula $T_R \vee T_B \vee T_Y$ is consistent with S_G. Define a coloring of G as follows: the color of vertex i is red if $v(i)$ satisfies T_R, blue if $v(i)$ satisfies T_B, and yellow if $v(i)$ satisfies T_Y (we break ties arbitrarily if $v(i)$ satisfies more than one term). Since

the formula is consistent with S_G, every $v(i)$ must satisfy some term, and so every vertex must be assigned a color by this process. We now argue that it is a legal 3-coloring. To see this, note that if i and j $(i \neq j)$ are assigned the same color (say red), then both $v(i)$ and $v(j)$ satisfy T_R. Since the ith bit of $v(i)$ is 0 and the ith bit of $v(j)$ is 1, it follows that neither x_i nor $\overline{x}_i$ can appear in T_R. Since $v(j)$ and $e(i,j)$ differ only in their ith bits, if $v(j)$ satisfies T_R then so does $e(i,j)$, implying $e(i,j) \notin S_G^-$ and hence $(i,j) \notin E$. □(Theorem 1.3)

Thus, we see that 3-term DNF formulae are not efficiently PAC learnable under the assumption that *NP*-complete problems cannot be solved with high probability by a probabilistic polynomial-time algorithm (technically, under the assumption $RP \neq NP$). With some more elaborate technical gymnastics, the same statement can in fact be made for 2-term DNF formulae, and for k-term DNF formulae for any constant $k \geq 2$.

However, note that our reduction relied critically on our demand in the definition of PAC learning that the learning algorithm output a hypothesis from the same representation class from which the target formula is drawn — we used each term of the hypothesis 3-term formula to define a color class in the graph. In the next section we shall see that this demand is in fact necessary for this intractability result, since its removal permits an efficient learning algorithm for this same class. This will motivate our final modification of the definition of PAC learning.

1.5 Using 3-CNF Formulae to Avoid Intractability

We conclude this chapter by showing that if we allow the learning algorithm to output a more expressive hypothesis representation, then the class of 3-term DNF formulae *is* efficiently PAC learnable. In combination with Theorem 1.3, this motivates our final modification to the definition of PAC learning.

We can use the fact that for boolean algebra, $\vee$ distributes over $\wedge$ (that is, $(u \wedge v) \vee (w \wedge x) = (u \vee w) \wedge (u \vee x) \wedge (v \vee w) \wedge (v \vee x)$ for boolean variables u, v, w, x) to represent any 3-term DNF formula over $x_1, \ldots, x_n$ by an equivalent conjunctive normal form (CNF) formulae over $x_1, \ldots, x_n$ in which each clause contains at most 3 literals (we will call such formulae **3-CNF formulae**):

$$T_1 \vee T_2 \vee T_3 \equiv \bigwedge_{u \in T_1, v \in T_2, w \in T_3} (u \vee v \vee w).$$

Here the conjunction is over all clauses choosing one literal from each term.

We can *reduce* the problem of PAC learning 3-CNF formulae to the problem of PAC learning conjunctions, for which we already have an efficient algorithm. The high-level idea is as follows: given an oracle for random examples of an unknown 3-CNF formula, there is a simple and efficient method by which we can transform each positive or negative example into a corresponding positive or negative example of an unknown conjunction (over a larger set of variables). We then give the transformed examples to the learning algorithm for conjunctions that we have already described in Section 1.3. The hypothesis output by the learning algorithm for conjunctions can then be transformed into a good hypothesis for the unknown 3-CNF formula.

To describe the desired transformation of examples, we regard a 3-CNF formula as a conjunction over a new and larger variable set. For every triple of literals u, v, w over the original variable set $x_1, \ldots, x_n$, the new variable set contains a variable $y_{u,v,w}$ whose value is defined by $y_{u,v,w} = u \vee v \vee w$. Note that when $u = v = w$, then $y_{u,v,w} = u$, so all of the original variables are present in the new set. Also, note that the number of new variables $y_{u,v,w}$ is $(2n)^3 = O(n^3)$.

Thus for any assignment $a \in \{0,1\}^n$ to the original variables $x_1, \ldots, x_n$, we can in time $O(n^3)$ compute the corresponding assignment a' to the new variables $\{y_{u,v,w}\}$. Furthermore, it should be clear that any 3-CNF formula c over $x_1, \ldots, x_n$ is equivalent to a simple conjunction c' over the

new variables (just replace any clause $(u \vee v \vee w)$ by an occurrence of the new variable $y_{u,v,w}$). Thus, we can run our algorithm for conjunctions from Section 1.3, expanding each assignment to $x_1, \ldots, x_n$ that is a positive example of the unknown 3-CNF formula into an assignment for the $y_{u,v,w}$, and giving this expanded assignment to the algorithm as a positive example of an unknown conjunction over the $y_{u,v,w}$. We can then convert the resulting hypothesis conjunction h' over the $y_{u,v,w}$ back to a 3-CNF h in the obvious way, by expanding an occurrence of the variable $y_{u,v,w}$ to the clause $(u \vee v \vee w)$.

Formally, we must argue that if c and $\mathcal{D}$ are the target 3-CNF formula and distribution over $\{0,1\}^n$, and c' and $\mathcal{D}'$ are the corresponding conjunction over the $y_{u,v,w}$ and induced distribution over assignments a' to the $y_{u,v,w}$, then if h' has error less than ϵ with respect to c' and $\mathcal{D}'$, h has error less than ϵ with respect to c and $\mathcal{D}$. This is most easily seen by noting that our transformation of instances is one-to-one: if a_1 is mapped to a_1' and a_2 is mapped to a_2', then $a_1 \neq a_2$ implies $a_1' \neq a_2'$. Thus each vector a' on which h' differs from c' has a unique preimage a on which h differs from c, and the weight of a under $\mathcal{D}$ is exactly that of a' under $\mathcal{D}'$. It is worth noting, however, that our reduction is exploiting the fact that our conjunctions learning algorithm works for any distribution $\mathcal{D}$, as the distribution is "distorted" by the transformation. For example, even if $\mathcal{D}$ was the uniform distribution over $\{0,1\}^n$, $\mathcal{D}'$ would not be uniform over the transformed assignments a'.

We have just given an example of a **reduction** between two learning problems. A general notion of reducibility in PAC learning will be formalized and studied in Chapter 7.

We have proven:

Theorem 1.4 *The representation class of 3-CNF formulae is efficiently PAC learnable.*

Thus, because we have already shown that any 3-term DNF formula

can be written as a 3-CNF formula, we can PAC learn 3-term DNF formulae if we allow the hypothesis to be represented as a 3-CNF formula, but not if we insist that it be represented as a 3-term DNF formula! The same statement holds for any constant $k \geq 2$ for k-term DNF formulae and k-CNF formulae. This demonstrates an important principle that often appears in learning theory: even for a fixed concept class from which target concepts are chosen, the choice of hypothesis representation can sometimes mean the difference between efficient algorithms and intractability. The specific cause of intractability here is worth noting: the problem of just predicting the classification of new examples of a 3-term DNF formula is tractable (we can use a 3-CNF formula for this purpose), but expressing the prediction rule in a particular form (namely, 3-term DNF formulae) is hard.

This state of affairs motivates us to generalize our basic definition one more time, to allow the learning algorithm to use a more expressive hypothesis representation than is strictly required to represent the target concept. After all, we would not have wanted to close the book on the learnability of 3-term DNF formulae after our initial intractability result just because we were constrained by an artificial definition that insisted that learning algorithms use some particular hypothesis representation. Thus our final modification to the definition of PAC learning lets the hypothesis representation used be a parameter of the PAC learning problem.

Definition 4 *(The PAC Model, Final Definition) If $\mathcal{C}$ is a concept class over X and $\mathcal{H}$ is a representation class over X, we will say that $\mathcal{C}$* **is (efficiently) PAC learnable using** *$\mathcal{H}$ if our basic definition of PAC learning (Definition 2) is met by an algorithm that is now allowed to output a hypothesis from $\mathcal{H}$. Here we are implicitly assuming that $\mathcal{H}$ is at least as expressive as $\mathcal{C}$, and so there is a representation in $\mathcal{H}$ of every function in $\mathcal{C}$. We will refer to $\mathcal{H}$ as the* **hypothesis class** *of the PAC learning algorithm.*

While for the reasons already discussed we do not want to place un-

necessary restrictions on $\mathcal{H}$, neither do we want to leave $\mathcal{H}$ entirely unconstrained. In particular, it would be senseless to study a model of learning in which the learning algorithm is constrained to run in polynomial time, but the hypotheses output by this learning algorithm could not even be evaluated in polynomial time. This motivates the following definition.

Definition 5 *We say that the representation class $\mathcal{H}$ is* **polynomially evaluatable** *if there is an algorithm that on input any instance $x \in X_n$ and any representation $h \in \mathcal{H}_n$, outputs the value $h(x)$ in time polynomial in n and size(h).*

Throughout our study, we will always be implicitly assuming that PAC learning algorithms use polynomially evaluatable hypothesis classes. Using our new language, our original definition was for PAC learning $\mathcal{C}$ using $\mathcal{C}$, and now we shall simply say that $\mathcal{C}$ is **efficiently PAC learnable** to mean that $\mathcal{C}$ is efficiently PAC learnable using $\mathcal{H}$ for some polynomially evaluatable hypothesis class $\mathcal{H}$.

The main results of this chapter are summarized in our new language by the following theorem.

Theorem 1.5 *The representation class of 1-term DNF formulae (conjunctions) is efficiently PAC learnable using 1-term DNF formulae. For any constant $k \geq 2$, the representation class of k-term DNF formulae is not efficiently PAC learnable using k-term DNF formulae (unless $RP = NP$), but is efficiently PAC learnable using k-CNF formulae.*

1.6 Exercises

1.1. Generalize the algorithm for the rectangle learning game to prove that if $\mathcal{C}_n$ is the class of all axis-aligned hyperrectangles in n-dimensional Euclidean space $\Re^n$, then $\mathcal{C}$ is efficiently PAC learnable.

1.2. Let $f(\cdot)$ be an integer-valued function, and assume that there does not exist a randomized algorithm taking as input a graph G and a parameter $0 < \delta \leq 1$ that runs in time polynomial in $1/\delta$ and the size of G, and that with probability at least $1 - \delta$ outputs "no" if G is not k-colorable and outputs an $f(k)$-coloring of G otherwise. Then show that for some $k \geq 3$, k-term DNF formulae are not efficiently PAC learnable using $f(k)$-term DNF formulae.

1.3. Consider the following **two-oracle** variant of the PAC model: when $c \in \mathcal{C}$ is the target concept, there are separate and arbitrary distributions $\mathcal{D}_c^+$ over only the positive examples of c and $\mathcal{D}_c^-$ over only the negative examples of c. The learning algorithm now has access to two oracles $EX(c, \mathcal{D}_c^+)$ and $EX(c, \mathcal{D}_c^-)$ that return a random positive example or a random negative example in unit time. For error parameter ϵ, the learning algorithm must find a hypothesis satisfying $\mathbf{Pr}_{x \in \mathcal{D}_c^+}[h(x) = 0] \leq \epsilon$ and $\mathbf{Pr}_{x \in \mathcal{D}_c^-}[h(x) = 1] \leq \epsilon$. Thus, the learning algorithm may now explicitly request either a positive or negative example, but must find a hypothesis with small error on both distributions.

Let $\mathcal{C}$ be any concept class and $\mathcal{H}$ be any hypothesis class. Let h_0 and h_1 be representations of the identically 0 and identically 1 functions, respectively. Prove that $\mathcal{C}$ is efficiently PAC learnable using $\mathcal{H}$ in the original one-oracle model if and only if $\mathcal{C}$ is efficiently PAC learnable using $\mathcal{H} \cup \{h_0, h_1\}$ in the two-oracle model.

1.4. Let $\mathcal{C}$ be any concept class and $\mathcal{H}$ be any hypothesis class. Let h_0 and h_1 be representations of the identically 0 and identically 1 functions, respectively. Show that if there is a randomized algorithm for efficiently PAC learning $\mathcal{C}$ using $\mathcal{H}$, then there is a deterministic algorithm for efficiently PAC learning $\mathcal{C}$ using $\mathcal{H} \cup \{h_0, h_1\}$.

1.5. In Definition 2, we modified the PAC model to allow the learning algorithm time polynomial in n and *size*(c), and also provided the value *size*(c) as input. Prove that this input is actually unnecessary: if there is an efficient PAC learning algorithm for $\mathcal{C}$ that is given *size*(c) as input, then there is an efficient PAC learning algorithm for $\mathcal{C}$ that is not given

this input.

1.7 Bibliographic Notes

The PAC model was defined in the seminal paper of L.G. Valiant [92], and was elaborated upon in his two subsequent papers [91, 93]. Much of this book is devoted to results in this probabilistic model. Papers by Haussler [45, 46, 47, 44] and Kearns, Li, Pitt and Valiant [59] describe some results in the PAC model from an artificial intelligence perspective.

In addition to defining the model, Valiant's original paper [92] proposed and analyzed the algorithm for PAC learning boolean conjunctions that we presented in Section 1.3. The informal rectangle game which began our study was formally analyzed in the PAC model in another important paper due to Blumer, Ehrenfeucht, Haussler and Warmuth [22], whose main results are the topic of Chapter 3.

The importance of hypothesis representation was first explored by Pitt and Valiant [71]. They showed that k-term DNF is not efficiently PAC learnable using a hypothesis class of k-term DNF, but is efficiently PAC learnable using k-CNF. The general techniques we outlined in Section 1.4 have been used to obtain representation-dependent hardness theorems for many classes, including various neural network architectures (Blum and Rivest [16, 20], Judd [53]). Intractability results for PAC learning neural networks that do not rely on hypothesis class restrictions will be given in Chapter 6. The earliest intractability results for learning that can be translated into the PAC model are those for deterministic finite automata due to Gold [40], who showed that the problem of finding the smallest finite state machine consistent with a labeled sample is NP-hard. This result was dramatically improved to obtain a hardness result for even approximating the smallest machine by Pitt and Warmuth [72]. In Chapter 6 we shall give even stronger hardness results for PAC learning finite automata.

Since Valiant introduced the PAC model, there have been a dizzying number of extensions and variants proposed in the computational learning theory. Some of these variants leave what is efficiently learnable essentially unchanged, and were introduced primarily for technical convenience. Others are explicitly designed to change the PAC model in a significant way, for example by providing the learner with more power or a weaker learning criterion. Later we shall study some of these variants. The paper of Haussler, Kearns, Littlestone and Warmuth [49] contains many theorems giving equivalences and relationships between some of the different models in the literature. For instance, the solutions to Exercises 1.3, 1.4 and 1.5 are contained in this paper. Exercise 1.1 is from the Blumer et al. paper [22], and Exercise 1.2 is from Pitt and Valiant [71].

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 $\mathcal{C}$ using $\mathcal{H}$. Let $\mathcal{D}$ be the target distribution over the instance space X, let $c \in \mathcal{C}_n$ 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

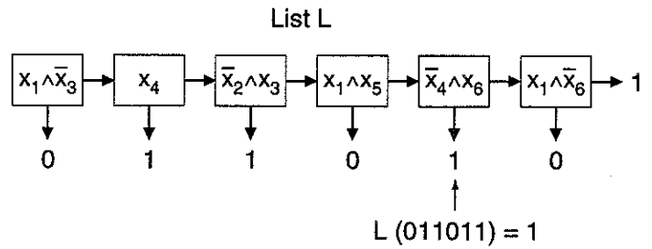

Figure 2.1: *A 2-decision list and the path followed by an input. Evaluation starts at the leftmost item and continues to the right until the first condition is satisfied, at which point the binary value below becomes the final result of the evaluation.*

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.

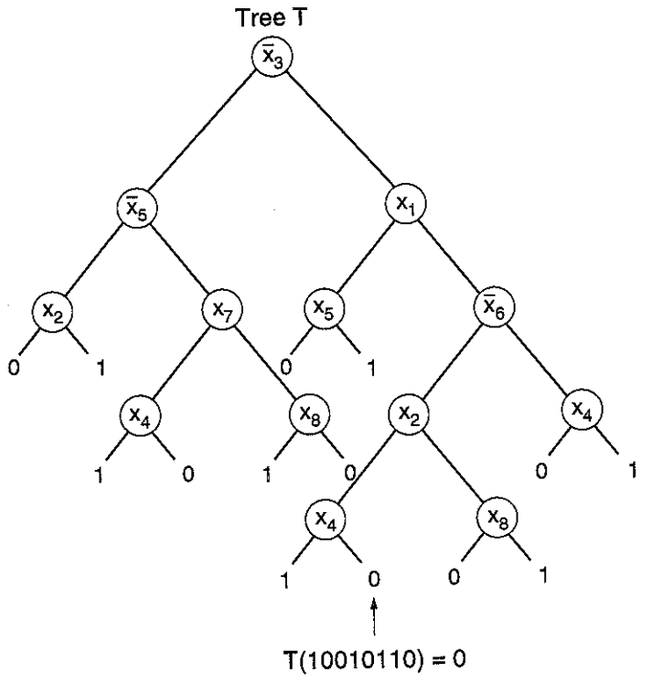

Figure 2.2: *A decision tree and the path followed by an input.*

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 $\mathcal{C}$ is PAC learnable if and only if there is an Occam algorithm for $\mathcal{C}$. 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.

3

The Vapnik-Chervonenkis Dimension

3.1 When Can Infinite Classes Be Learned with a Finite Sample?

In this chapter, we consider the following question: How many random examples does a learning algorithm need to draw before it has sufficient information to learn an unknown target concept chosen from the concept class $\mathcal{C}$? We should emphasize that we will temporarily ignore issues of computational efficiency while studying this question (or equivalently, we assume that the learning algorithm has infinite computing power to process the finite random sample it has drawn). We first note that the results of the previous chapter can be used to give such a bound in the case that $\mathcal{C}$ is a concept class of finite cardinality. If the learning algorithm simply draws a random sample of $O((1/\epsilon)\log(|\mathcal{C}|/\delta))$ examples, and finds any $h \in \mathcal{C}$ consistent with these examples (say, by exhaustive search), then Theorem 2.2 guarantees that h will meet the PAC model criteria. Notice that this bound is not meaningful if $\mathcal{C}$ has infinite cardinality. Are there any non-trivial infinite concept classes that are learnable from a finite sample?

Actually, our PAC learning algorithm for axis-aligned rectangles in the Euclidean plane given in Section 1.1 is an example of such a class. In the analysis of that PAC learning algorithm, we made critical use of the fact that axis-aligned rectangles have simple boundaries: the target rectangle is always completely specified by four real numbers that indicate the locations of the four bounding edges, and this allowed us to partition the error of the tightest-fit hypothesis into four simple rectilinear regions. It is tempting to say that the "complexity" of this concept class is four, because the boundary of any concept in the class can be described by four real numbers.

In this chapter, we are interested in a general measure of complexity for concept classes of infinite cardinality. We would like this measure to play the same role in the sample complexity of PAC learning infinite classes that the quantity $\log |\mathcal{C}|$ (which we saw in Chapter 2 was closely related to the size of representations) plays in the sample complexity of PAC learning finite classes. We will define a purely combinatorial measure of concept class complexity known as the *Vapnik-Chervonenkis dimension*, a measure that assigns to each concept class $\mathcal{C}$ a single number that characterizes the sample size needed to PAC learn $\mathcal{C}$.

3.2 The Vapnik-Chervonenkis Dimension

For the remainder of this chapter, $\mathcal{C}$ will be a concept class over instance space X, and both $\mathcal{C}$ and X may be infinite. The first thing we will need is a way to discuss the behavior of $\mathcal{C}$ when attention is restricted to a finite set of points $S \subseteq X$.

Definition 7 *For any concept class $\mathcal{C}$ over X, and any $S \subseteq X$,*

$$\Pi_{\mathcal{C}}(S) = \{c \cap S : c \in \mathcal{C}\}.$$

Equivalently, if $S = \{x_1, \ldots, x_m\}$ then we can think of $\Pi_{\mathcal{C}}(S)$ as the set of vectors $\Pi_{\mathcal{C}}(S) \subseteq \{0,1\}^m$ defined by

$$\Pi_{\mathcal{C}}(S) = \{(c(x_1), \ldots, c(x_m)) : c \in \mathcal{C}\}.$$

Thus, $\Pi_{\mathcal{C}}(S)$ is the set of all the **behaviors** or **dichotomies** on S that are induced or **realized** by $\mathcal{C}$. We will use the descriptions of $\Pi_{\mathcal{C}}(S)$ as a collection of subsets of S and as a set of vectors interchangeably.

Definition 8 *If $\Pi_{\mathcal{C}}(S) = \{0,1\}^m$ (where $m = |S|$), then we say that S is* **shattered** *by $\mathcal{C}$. Thus, S is shattered by $\mathcal{C}$ if $\mathcal{C}$ realizes all possible dichotomies of S.*

Now we are ready for our key definition.

Definition 9 *The* **Vapnik-Chervonenkis (VC) dimension** *of $\mathcal{C}$, denoted as $VCD(\mathcal{C})$, is the cardinality d of the largest set S shattered by $\mathcal{C}$. If arbitrarily large finite sets can be shattered by $\mathcal{C}$, then $VCD(\mathcal{C}) = \infty$.*

3.3 Examples of the VC Dimension

Let us consider a few natural geometric concept classes, and informally calculate their VC dimension. It is important to emphasize the nature of the existential and universal quantifiers in the definition of VC dimension: in order to show that the VC dimension of a class is at least d, we must simply find some shattered set of size d. In order to show that the VC dimension is at most d, we must show that no set of size $d+1$ is shattered. For this reason, proving upper bounds on the VC dimension is usually considerably more difficult than proving lower bounds. The following examples are not meant to be precise proofs of the stated bounds on

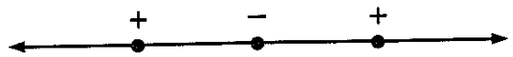

Figure 3.1: *A dichotomy unrealizable by intervals.*

the VC dimension, but are simply illustrative exercises to provide some practice thinking about the VC dimension.

Intervals of the real line. For this concept class, any set of two points can be shattered, so the VC Dimension is at least two, but no set of three points can be shattered: label the three points as shown in Figure 3.1, a labeling which cannot be induced by any interval. Thus the VC dimension for this class is two.

Linear halfspaces in the plane. For this concept class, any three points that are not collinear can be shattered. Figure 3.2(a) shows how one dichotomy out of the possible 8 dichotomies can be realized by a halfspace; the reader can easily verify that the remaining 7 dichotomies can be realized by halfspaces. To see that no set of four points can be shattered, we consider two cases. In the first case (shown in Figure 3.2(b)), all four points lie on the convex hull defined by the four points. In this case, if we label one "diagonal" pair positive and the other "diagonal" pair negative as shown in Figure 3.2(b), no halfspace can induce this labeling. In the second case (shown in Figure 3.2(c)), three of the four points define the convex hull of the four points, and if we label the interior point negative and the hull points positive, again no halfspace can induce the dichotomy. Thus the VC dimension here is three. In general, for halfspaces in $\Re^d$, the VC dimension is $d+1$.

Axis-aligned rectangles in the plane. For this concept class, we can shatter the four points shown in Figure 3.3(a), where we have again indicated how a single dichotomy can be realized and left the remainder to the reader. However, not *all* sets of four points can be shattered, as indicated by the unrealizable dichotomy shown in Figure 3.3(b). Still, the existence of a single shattered set of size four is sufficient to lower bound

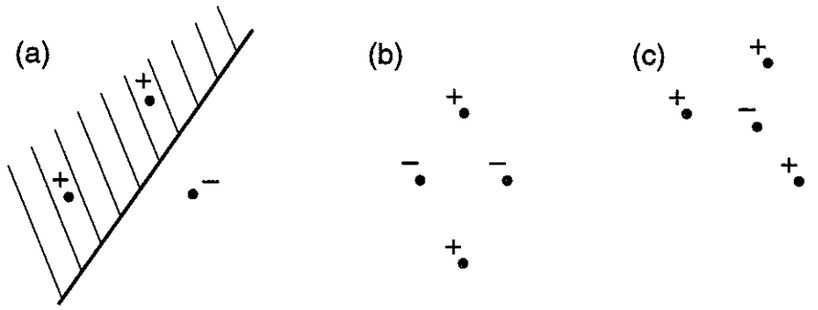

Figure 3.2: (a) *A dichotomy and its realization by a halfspace, with the shaded region indicating the positive side.* (b) and (c) *Dichotomies unrealizable by halfspaces.*

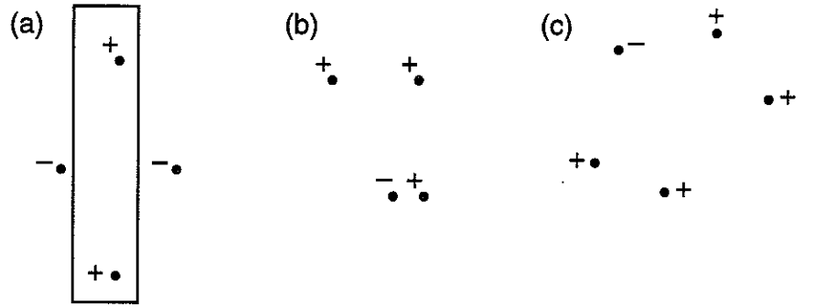

Figure 3.3: (a) *A dichotomy and its realization by an axis-aligned rectangle.* (b) and (c) *Dichotomies unrealizable by axis-aligned rectangles.*

the VC dimension. Now for any set of five points in the plane, there must be some point that is neither the extreme left, right, top or bottom point of the five (see Figure 3.3(c)). If we label this non-extremal point negative and the remaining four extremal point positive, no rectangle can realize the dichotomy. Thus the VC dimension is four.

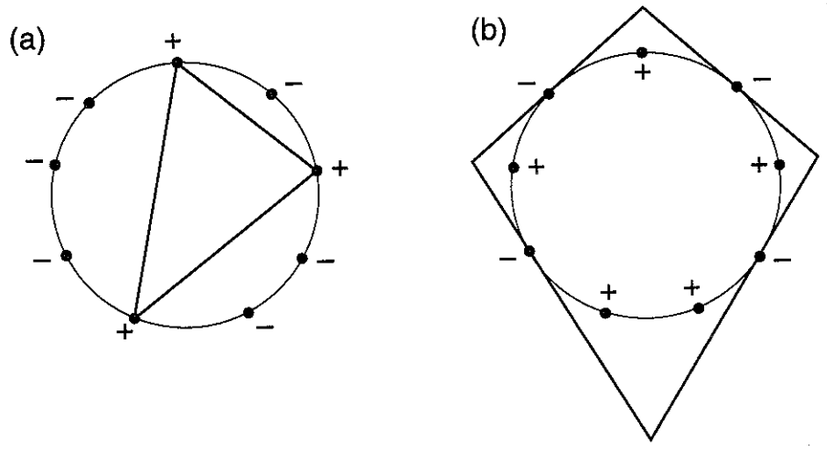

Figure 3.4: (a) *Realizing a dichotomy with a polygon when there are fewer positive labels.* (b) *When there are fewer negative labels.*

Convex polygons in the plane. For convex d-gons in the plane, the VC dimension is $2d + 1$. For the lower bound, we can induce any labeling of any $2d+1$ points on a circle using a d-gon as follows: if there are more negative labels than positive labels, use the positive points as the vertices as shown in Figure 3.4(a). Otherwise, use tangents to the negative points as edges as shown in Figure 3.4(b). For the upper bound, it can be shown that choosing the points to lie on a circle does in fact maximize the number of points that can be shattered, and we can force $d + 1$ sides using $2d + 2$ points on a circle by alternating positive and negative labels.

3.4 A Polynomial Bound on $|\Pi_C(S)|$

Definition 10 *For any natural number m we define*

$$\Pi_C(m) = \max\{|\Pi_C(S)| : |S| = m\}.$$

The function $\Pi_C(m)$ can be thought of as a measure of the complexity of C: the faster this function grows, the more behaviors on sets of m points that can be realized by C as m increases. Now clearly, if C does not have finite VC dimension, then $\Pi_C(m) = 2^m$ for all m since we can shatter arbitrarily large finite sets. In this section, we prove a surprising and beautiful result, namely that despite the fact that we might naively expect $\Pi_C(m)$ to grow as rapidly as an exponential function of m, it is actually bounded by a polynomial in m of degree d, where d is the VC dimension of C. In other words, depending on whether the VC dimension is finite or infinite, the function $\Pi_C(m)$ is either eventually polynomial or forever exponential. For the more interesting and typical case of finite VC dimension, we shall eventually translate the polynomial upper bound on $\Pi_C(m)$ into an upper bound on the sample complexity of PAC learning that is linear in d.

We begin by proving that $\Pi_C(m)$ is bounded by the function $\Phi_d(m)$ defined below. We then show a polynomial bound on $\Phi_d(m)$.

Definition 11 *For any natural numbers m and d, the function $\Phi_d(m)$ is defined inductively by*

$$\Phi_d(m) = \Phi_d(m-1) + \Phi_{d-1}(m-1)$$

with initial conditions $\Phi_d(0) = \Phi_0(m) = 1$.

Lemma 3.1 *If $VCD(C) = d$, then for any m, $\Pi_C(m) \leq \Phi_d(m)$.*

Proof: By induction on both d and m. For the base cases, the lemma is easily established when $d = 0$ and m is arbitrary, and when $m = 0$ and d is arbitrary. We assume for induction that for all m', d' such that $m' \leq m$ and $d' \leq d$ and at least one of the two inequalities is strict, we have $\Pi_C(m') \leq \Phi_{d'}(m')$. We now show that this inductive assumption establishes the desired statement for d and m.

Given any set S of size m, let $x \in S$ be a distinguished point. Let us first compute $|\Pi_{\mathcal{C}}(S - \{x\})|$. This is easy since by induction (note that $S - \{x\}$ is a set of size $m-1$) we have $|\Pi_{\mathcal{C}}(S - \{x\})| \leq \Phi_d(m-1)$.

The difference between $\Pi_{\mathcal{C}}(S)$ and $\Pi_{\mathcal{C}}(S-\{x\})$ is that pairs of distinct sets in $\Pi_{\mathcal{C}}(S)$ that differ only on their labeling of x are identified (that is, merged) in $\Pi_{\mathcal{C}}(S - \{x\})$. Thus let us define

$$\mathcal{C}' = \{c \in \Pi_{\mathcal{C}}(S) : x \notin c, c \cup \{x\} \in \Pi_{\mathcal{C}}(S)\}.$$

Then $|\mathcal{C}'|$ counts the number of pairs of sets in $\Pi_{\mathcal{C}}(S)$ that are collapsed to a single representative in $\Pi_{\mathcal{C}}(S - \{x\})$. Note that $\mathcal{C}' = \Pi_{\mathcal{C}'}(S - \{x\})$ because $\mathcal{C}'$ consists only of subsets of $S - \{x\}$. This yields the simple equality

$$|\Pi_{\mathcal{C}}(S)| = |\Pi_{\mathcal{C}}(S - \{x\})| + |\Pi_{\mathcal{C}'}(S - \{x\})|.$$

We now show that $VCD(\mathcal{C}') \leq d-1$. To see this, let $S' \subseteq S - \{x\}$ be shattered by $\mathcal{C}'$. Then $S' \cup \{x\}$ is shattered by $\mathcal{C}$. Thus we must have $|S'| \leq d-1$. Now by induction we have $|\mathcal{C}'| = |\Pi_{\mathcal{C}'}(S - \{x\})| \leq \Phi_{d-1}(m-1)$.

Our total count is thus bounded by $\Phi_d(m-1)+\Phi_{d-1}(m-1) = \Phi_d(m)$, as desired. □(Lemma 3.1)

Lemma 3.2 $\Phi_d(m) = \sum_{i=0}^{d} \binom{m}{i}$.

Proof: By induction; the base cases are easy to check. For the induction step, we have:

$$\begin{aligned}
\Phi_d(m) &= \Phi_d(m-1) + \Phi_{d-1}(m-1) \\
&= \sum_{i=0}^{d} \binom{m-1}{i} + \sum_{i=0}^{d-1} \binom{m-1}{i} \\
&= \sum_{i=0}^{d} \left[\binom{m-1}{i} + \binom{m-1}{i-1} \right] \\
&= \sum_{i=0}^{d} \binom{m}{i}
\end{aligned}$$

where the second equality is by induction and we define $\binom{m-1}{-1} = 0$ for the third equality. □(Lemma 3.2)

Now for $m \leq d$, $\Phi_d(m) = 2^m$. For $m > d$, since $0 \leq d/m \leq 1$, we may write:

$$\left(\tfrac{d}{m}\right)^d \sum_{i=0}^{d} \binom{m}{i} \leq \sum_{i=0}^{d} \left(\tfrac{d}{m}\right)^i \binom{m}{i} \leq \sum_{i=0}^{m} \left(\tfrac{d}{m}\right)^i \binom{m}{i} = \left(1 + \tfrac{d}{m}\right)^m \leq e^d.$$

Dividing both sides by $\left(\frac{d}{m}\right)^d$ yields

$$\Phi_d(m) = \sum_{i=0}^{d} \binom{m}{i} \leq \left(\frac{em}{d}\right)^d = O(m^d)$$

which is polynomial in m for fixed d, giving us the promised polynomial bound for the case $m > d$.

3.5 A Polynomial Bound on the Sample Size for PAC Learning

3.5.1 The Importance of ϵ-Nets

Let us now fix the target concept $c \in \mathcal{C}$, and define the class of **error regions** with respect to c and $\mathcal{C}$ by $\Delta(c) = \{c \Delta c' : c' \in \mathcal{C}\}$. It is easy to show that $VCD(\mathcal{C}) = VCD(\Delta(c))$. To see this, for any set S we can map each element $c' \in \Pi_{\mathcal{C}}(S)$ to $c'\Delta(c \cap S) \in \Pi_{\Delta(c)}(S)$. Since this is a bijective mapping of $\Pi_{\mathcal{C}}(S)$ to $\Pi_{\Delta(c)}(S)$, $|\Pi_{\Delta(c)}(S)| = |\Pi_{\mathcal{C}}(S)|$. Since this holds for any set S, $VCD(\mathcal{C}) = VCD(\Delta(c))$ follows.

We may further refine the definition of $\Delta(c)$ to consider only those error regions with weight at least ϵ under the fixed target distribution $\mathcal{D}$. Thus, let $\Delta_\epsilon(c) = \{r \in \Delta(c) : \mathbf{Pr}_{x \in \mathcal{D}}[x \in r] \geq \epsilon\}$. We can now make the following important definition:

Definition 12 *For any $\epsilon > 0$, we say that a set S is an* **ϵ-net** *for $\Delta(c)$ if every region in $\Delta_\epsilon(c)$ is "hit" by a point in S, that is, if for every $r \in \Delta_\epsilon(c)$ we have $S \cap r \neq \emptyset$.*

An ϵ-net for $\Delta(c)$ is thus a set that hits all of the ϵ-heavy regions of $\Delta(c)$. As an example, suppose X is the closed interval $[0,1]$ and let $\mathcal{D}$ be the uniform density on X. Suppose that $\mathcal{C}$ consists of all closed intervals on $[0,1]$ as well as the empty set $\emptyset$, and that the target concept $c = \emptyset$. Then $\Delta(c)$ is again the set of all closed intervals on $[0,1]$. For any interval I under the uniform density, $\mathbf{Pr}_{x\in\mathcal{D}}[x \in I]$ is just the length of I. Any interval whose probability is greater than ϵ will have length greater than ϵ, so the set of all points $k\epsilon$, for natural numbers $1 \leq k \leq \lceil 1/\epsilon \rceil$, is an ϵ-net for $\Delta(c)$.

The notion of ϵ-nets has actually been implicit in some of our earlier analyses, in particular those of Occam's Razor in Chapter 2. The important property of ϵ-nets is that if the sample S drawn by a learning algorithm forms an ϵ-net for $\Delta(c)$, and the learning algorithm outputs a hypothesis $h \in \mathcal{C}$ that is consistent with S, then this hypothesis must have error less than ϵ: since $c\Delta h \in \Delta(c)$ was not hit by S (otherwise h would not be consistent with S), and S is an ϵ-net for $\Delta(c)$, we must have $c\Delta h \notin \Delta_\epsilon(c)$ and therefore $error(h) \leq \epsilon$.

Thus if we can bound the probability that the random sample S fails to form an ϵ-net for $\Delta(c)$, then we have bounded the probability that a hypothesis consistent with S has error greater than ϵ. For the case of finite $\mathcal{C}$, the analysis of Occam's Razor obtained such a bound by a simple counting argument that we sketch again here in our new notation: for any fixed error region $c\Delta h \in \Delta_\epsilon(c)$, the probability that we fail to hit $c\Delta h$ in m random examples is at most $(1-\epsilon)^m$. Thus the probability that we fail to hit some $c\Delta h \in \Delta_\epsilon(c)$ is bounded above by $|\Delta(c)|(1-\epsilon)^m$, which in turn is bounded by $|\mathcal{C}|(1-\epsilon)^m$.

Alternatively, we can carry out the above analysis replacing $|\mathcal{C}|$ by $\Phi_d(|X|)$. This follows immediately from the fact that $\mathcal{C} = \Pi_\mathcal{C}(X)$ and

Lemma 3.1. This gives us a bound of $\Phi_d(|X|)(1-\epsilon)^m$ on the probability of failing to draw an ϵ-net for $\Delta(c)$. However, this does not represent any progress over the state of affairs in which we began this chapter, since if X is infinite then $\Phi_d(|X|)$ is infinite as well. Ideally, we would like to carry out a similar analysis that instead of considering the entire domain X considers only the small random subset S observed by the learning algorithm.

3.5.2 A Small ϵ-Net from Random Sampling

We now show that if we draw a small set of examples from the oracle $EX(c,\mathcal{D})$, then they form an ϵ-net with high probability. The important property is that the size of the required sample depends on the VC dimension d and ϵ and δ, but is independent of $|\mathcal{C}|$ and $|X|$. From the preceding discussion, this will immediately lead to an upper bound on the number of examples required for PAC learning that depends only on these same quantities.

Suppose that we draw a multiset S_1 of m random examples from $\mathcal{D}$, and let A denote the event that the elements of S_1 fail to form an ϵ-net for $\Delta(c)$. Clearly, our goal is to upper bound the probability of event A. If event A occurs, then by the definition of ϵ-nets, S_1 misses some region $r \in \Delta_\epsilon(c)$. Let us fix this missed region r, and suppose we now draw an additional multiset S_2 of m random examples from $\mathcal{D}$. Since each element of S_2 has probability at least ϵ of hitting r, if $m = O(1/\epsilon)$ the probability S_2 hits r at least $\epsilon m/2$ times is at least $1/2$ by Markov's inequality (see the Appendix in Chapter 9).

If we let B be the combined event over the random draws of S_1 and S_2 that A occurs on the draw of S_1 (so S_1 is not an ϵ-net) *and* S_2 has at least $\epsilon m/2$ hits in a region of $\Delta_\epsilon(c)$ that is missed by S_1, then we have argued that $\mathbf{Pr}[B|A] \geq 1/2$. Since the definition of event B already requires that event A occurs on S_1, we also have $\mathbf{Pr}[B] = \mathbf{Pr}[B|A]\mathbf{Pr}[A]$, so $2\mathbf{Pr}[B] \geq \mathbf{Pr}[A]$.

Thus, we can upper bound the probability of event A by upper bounding the probability of event B. The principal advantage of event B over event A for the purposes of our analysis can be described as follows. To directly analyze the probability of event A, we must consider all regions of the uncountably infinite class $\Delta_\epsilon(c)$ that S_1 might miss. To analyze the probability of event B, we need only consider the regions of $\Pi_{\Delta_\epsilon(c)}(S_1 \cup S_2)$. This is because the occurrence of event B is equivalent to saying that there is some $r \in \Pi_{\Delta_\epsilon(c)}(S_1 \cup S_2)$ such that $|r| \geq \epsilon m/2$ and $r \cap S_1 = \emptyset$.

To bound the probability that such an r exists, rather than drawing S_1 at random and then drawing S_2 at random, we can instead first draw a multiset S of $2m$ instances at random, and then randomly divide S into S_1 and S_2. The resulting distribution of S_1 and S_2 is the same in both experiments, since each draw from $\mathcal{D}$ is independent and identically distributed. Now once S is drawn and fixed (but before it is divided randomly into S_1 and S_2), we may also fix a region $r \in \Pi_{\Delta_\epsilon(c)}(S)$ satisfying $|r| \geq \epsilon m/2$. For this fixed S and fixed r, we now analyze the probability (with respect only to the random partitioning of S into S_1 and S_2) that $r \cap S_1 = \emptyset$. We will then obtain a bound on the probability of event B by summing over all possible fixed $r \in \Pi_{\Delta_\epsilon(c)}(S)$ and applying the union bound.

Our problem is now reduced to the following simple combinatorial experiment: we have $2m$ balls (the multiset S), each colored red or blue, with exactly $\ell \geq \epsilon m/2$ red balls (these are the instances of S that fall in r). We divide these balls randomly into two groups of equal size S_1 and S_2, and we are interested in bounding the probability that all ℓ of the red balls fall in S_2 (that is, the probability that $r \cap S_1 = \emptyset$).

Equivalently, we can first divide $2m$ uncolored balls into S_1 and S_2, and then randomly choose ℓ of the balls to be marked red, the rest being marked blue. Then the probability that all ℓ of the red marks fall on balls S_2 is exactly $\binom{m}{\ell}/\binom{2m}{\ell}$ — this is simply the number of ways we can choose the ℓ red marks in S_2 divided by the number of ways the ℓ red marks can be chosen without constraints. But $\binom{m}{\ell}/\binom{2m}{\ell} \leq 1/2^\ell$. This is because

$$\frac{\binom{m}{\ell}}{\binom{2m}{\ell}} = \Pi_{i=0}^{\ell-1} \frac{(m-i)}{(2m-i)} \leq \Pi_{i=0}^{\ell-1} \left(\frac{1}{2}\right) = \frac{1}{2^\ell}.$$

Thus, for any fixed S and $r \in \Pi_{\Delta_\epsilon(c)}(S)$ satisfying $|r| \geq \epsilon m/2$, the probability that the random partitioning of S results in $r \cap S_1 = \emptyset$ is at most $2^{-\epsilon m/2}$. The probability that this occurs for *some* $r \in \Pi_{\Delta_\epsilon(c)}(S)$ satisfying $|r| \geq \epsilon m/2$ (and thus $\mathbf{Pr}[B]$) is at most

$$|\Pi_{\Delta_\epsilon(c)}(S)| 2^{-\frac{\epsilon m}{2}} \leq |\Pi_{\Delta(c)}(S)| 2^{-\frac{\epsilon m}{2}} \leq |\Pi_{\mathcal{C}}(S)| 2^{-\frac{\epsilon m}{2}}$$

$$\leq \Phi_d(2m) 2^{-\frac{\epsilon m}{2}} \leq \left(\frac{2em}{d}\right)^d 2^{-\frac{\epsilon m}{2}}.$$

Finally, $\mathbf{Pr}[A] \leq 2\mathbf{Pr}[B] \leq 2(2em/d)^d 2^{-\epsilon m/2}$, which is less than δ for

$$m = O\left(\frac{1}{\epsilon}\log\frac{1}{\delta} + \frac{d}{\epsilon}\log\frac{1}{\epsilon}\right).$$

We have proved the main result of this chapter:

Theorem 3.3 *Let $\mathcal{C}$ be any concept class of VC dimension d. Let L be any algorithm that takes as input a set S of m labeled examples of a concept in $\mathcal{C}$, and produces as output a concept $h \in \mathcal{C}$ that is consistent with S. Then L is a PAC learning algorithm for $\mathcal{C}$ provided it is given a random sample of m examples from $EX(c, \mathcal{D})$, where m obeys*

$$m \geq c_0 \left(\frac{1}{\epsilon}\log\frac{1}{\delta} + \frac{d}{\epsilon}\log\frac{1}{\epsilon}\right)$$

for some constant $c_0 > 0$.

Recall that in Chapter 1, we saw that for computational reasons there may sometimes be a great advantage in using a hypothesis class $\mathcal{H}$ that is more powerful than the class $\mathcal{C}$ from which the target is chosen. The reader can verify that the same proof used to establish Theorem 3.3 can be used to prove the following analogue:

Theorem 3.4 *Let $\mathcal{C}$ be any concept class. Let $\mathcal{H}$ be any representation class of VC dimension d. Let L be any algorithm that takes as input a set S of m labeled examples of a concept in $\mathcal{C}$, and produces as output a concept $h \in \mathcal{H}$ that is consistent with S. Then L is a PAC learning algorithm for $\mathcal{C}$ using $\mathcal{H}$ provided it is given a random sample of m examples from $EX(c, \mathcal{D})$, where m obeys*

$$m \geq c_0 \left(\frac{1}{\epsilon} \log \frac{1}{\delta} + \frac{d}{\epsilon} \log \frac{1}{\epsilon} \right)$$

for some constant $c_0 > 0$.

Thus, to obtain an algorithm for PAC learning $\mathcal{C}$ using $\mathcal{H}$, we take a number of examples on the order of the VC dimension of $\mathcal{H}$ (which is at least as large as the VC dimension of $\mathcal{C}$ if $\mathcal{H} \supset \mathcal{C}$). This shows that while we may reduce our computation time by choosing a more powerful hypothesis representation, we may also increase the number of examples required.

3.6 Sample Size Lower Bounds

We now show that the upper bound on the sample complexity of PAC learning given by Theorem 3.3 is tight within a factor of $O(\log 1/\epsilon)$ (ignoring the dependence on δ). First we show a lower bound of $\Omega(d)$ on the number of examples required for PAC learning using a fairly simple argument, then we present a refined argument that improves the bound to $\Omega(d/\epsilon)$.

Theorem 3.5 *Any algorithm for PAC learning a concept class of Vapnik-Chervonenkis dimension d must use $\Omega(d/\epsilon)$ examples in the worst case.*

Proof: Consider a concept class $\mathcal{C}$ such that $VCD(\mathcal{C}) = d$. Let $S = \{x_1, \ldots, x_d\}$ be shattered by $\mathcal{C}$. To show a lower bound, we construct a particular distribution that forces any PAC learning algorithm to take many examples. Thus, let $\mathcal{D}$ give probability $1/d$ to each point in S, and probability 0 to points not in S. For this distribution, we can assume without loss of generality that $\mathcal{C} = \Pi_{\mathcal{C}}(S)$ (that is, $X = S$), so $\mathcal{C}$ is a finite class and $|\mathcal{C}| = 2^d$.

Note that we have arranged things so that for all of the 2^d possible binary labelings of the points in S, there is exactly one concept in $\mathcal{C}$ that induces this labeling. Thus, choosing the target concept c randomly from $\mathcal{C}$ is equivalent to flipping a fair coin d times to determine the labeling induced by c on S.

Now let L be any PAC learning algorithm for $\mathcal{C}$. Set the error parameter $\epsilon \leq 1/8$, and consider running L when the target concept $c \in \mathcal{C}$ is chosen randomly and the input distribution is $\mathcal{D}$. Suppose that after drawing $m < d$ examples from $EX(c, \mathcal{D})$, L has drawn $m' \leq m$ different instances; without loss of generality, let these be $x_1, \ldots, x_{m'}$. Then from the above observations, it is clear that the problem of predicting the correct label of any unseen instance x_j for $j > m'$ is equivalent to predicting the outcome of a fair coin, since each label of c on S is determined by an independent coin flip. Thus the expected error (over the random choice of c and the sample of points) of L's hypothesis is $(d - m')/2d$, and by Markov's inequality (see the Appendix in Chapter 9) is at least $(d - m')/4d$ with probability at least $1/2$. For $m = d/2$ we obtain that the error of L's hypothesis is at least $1/8$ with probability at least $1/2$ (over the random choice of c and the sample). Since this shows that L must fail when c is chosen randomly, there must certainly be some fixed target concept on which L fails, thus giving the $\Omega(d)$ sample complexity lower bound.

To refine this argument to get a lower bound that incorporates ϵ, we simply scale the above coin flipping construction to a region of the distribution that is small but still too large to be "ignored" by the algorithm. Thus, we modify $\mathcal{D}$ to let the distinguished instance x_1 have probability $1 - 8\epsilon$ under $\mathcal{D}$ (we are essentially "giving" this instance along with its correct label to L), and let $x_2, \ldots, x_d$ each have probability $8\epsilon/(d-1)$

under $\mathcal{D}$ (this is the coin flipping region). Now by simply scaling our previous calculation to the coin flipping region, the expected error of L after seeing at most $d/2$ different instances is at least $(1/8)8\epsilon = \epsilon$ with probability at least $1/2$. But it is not difficult to show that now drawing $d/2$ different points requires $\Omega(d/\epsilon)$ examples, because our problem is reduced to obtaining $d/2$ "successes" in independent trials, each with probability of success only 4ϵ. □(Theorem 3.5)

3.7 An Application to Neural Networks

We conclude this chapter by giving a useful general lemma that bounds $VCD(\mathcal{C})$ when each concept in the class $\mathcal{C}$ is actually a **composition** of simpler concepts. Such classes arise frequently — for instance, a DNF formulae is simply a (very constrained) composition of boolean conjunctions (the constraint being that we can only compute disjunctions of conjunctions). After giving this lemma, we then apply it to obtain upper bounds on the sample size required for PAC learning neural networks.

To formalize a general notion of concept composition, let G be a **layered** directed acyclic graph. By this we mean that the nodes of G can be partitioned into layers, and the directed edges of G go only from a node at layer ℓ to a node at layer $\ell+1$. We let n be the number of nodes at layer 0, and we assume that all of these have indegree 0. We think of these n layer 0 nodes as being the inputs to the graph. We also assume that there is only a single node of outdegree 0 at the highest level of the graph, and we think of this node as being the output node of the graph. All internal (that is, non-input) nodes have the same indegree r, and we let s denote the number of internal nodes. Figure 3.5 shows an example of such a layered graph with $n = 8$, $s = 8$ and $r = 3$.

Now let $\mathcal{C}$ be a concept class over r-dimensional Euclidean space $\Re^r$. Suppose we take such a layered graph G, and we label each internal (that is, non-input) node N_i with a concept $c_i \in \mathcal{C}$. Then such a labeled

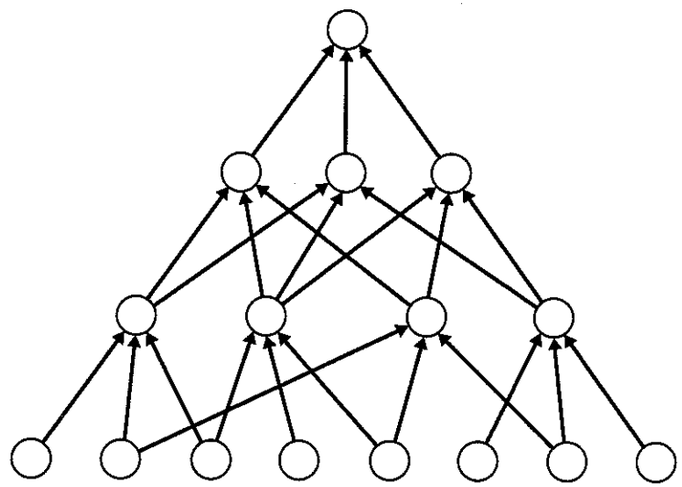

Figure 3.5: *A layered directed acyclic graph.*

graph represents a concept over n-dimensional Euclidean space $\Re^n$ in the obvious way: if we label each of the n input nodes at layer 0 with a real number, then starting with layer 1 we can compute the value at each node N_i by applying the concept c_i labeling node N_i to the values computed at the nodes feeding N_i. (Note that although concepts in $\mathcal{C}$ are defined over $\Re^r$, the input values feeding nodes at level 2 and higher will actually only be from $\{0,1\}^r$.) The output of the entire labeled graph is the binary value computed at the output node. We will call the class of all concepts over $\Re^n$ that can be obtained by labeling G with concepts from $\mathcal{C}$ the **G-composition of $\mathcal{C}$**, which we denote $\mathcal{C}_G$.

Theorem 3.6 *Let G be a layered directed acyclic graph with n input nodes and $s \geq 2$ internal nodes, each of indegree r. Let $\mathcal{C}$ be a concept class over $\Re^r$ of VC dimension d, and let $\mathcal{C}_G$ be the G-composition of $\mathcal{C}$. Then $VCD(\mathcal{C}_G) \leq 2ds\log(es)$.*

Proof: The idea is to first bound the function $\Pi_{\mathcal{C}_G}(m)$. Let us fix any

set S of m input vectors $\vec{x}_1, \ldots, \vec{x}_m \in \Re^n$ to the graph G (thus, each $\vec{x}_i$ determines a complete setting of the n input nodes of G). For this fixed input set S, if we now also label each node in G with a concept from $\mathcal{C}$, then for each $\vec{x}_i$ we have completely determined the binary values that will be computed at every node of G when the input is $\vec{x}_i$. Let us call the collection of all the values computed at each node, for each $\vec{x}_i \in S$, a *computation* of G on S. Thus, a computation can be represented by labeling each internal node with the vector in $\{0,1\}^m$ of the values computed at that node on the m vectors in S. Then the set of all possible computations of G on S is obtained by ranging over all possible choices of labels from $\mathcal{C}$ for the nodes of G. Note that two computations of G on S differ if and only if the value computed at some node on some input from S differs in the two computations. Clearly, $|\Pi_{\mathcal{C}_G}(S)|$ is bounded by the total number of possible computations of G on S, which we shall denote $T_{\mathcal{C}_G}(S)$.

To bound $T_{\mathcal{C}_G}(S)$, let G' be the subgraph obtained by removing the output node N_o from G. Let $T_{\mathcal{C}_{G'}}(S)$ denote the total number of computations of G' on S. Each fixed computation of G' can be extended to at most $\Pi_{\mathcal{C}}(m)$ computations of G, because fixing the computation of G' determines for each $1 \leq i \leq m$ the input $\vec{y}_i \in \{0,1\}^r$ that is fed to N_o when $\vec{x}_i$ is fed to G, and at most $\Pi_{\mathcal{C}}(m)$ labelings of $\vec{y}_1, \ldots, \vec{y}_m$ can be obtained at N_o by varying the choice of concept from $\mathcal{C}$ placed at N_o. Thus we obtain that for any S, $T_{\mathcal{C}_G}(S) \leq T_{\mathcal{C}_{G'}}(S) \times \Pi_{\mathcal{C}}(m)$, and a simple inductive argument establishes

$$|\Pi_{\mathcal{C}_G}(S)| \leq T_{\mathcal{C}_G}(S) \leq (\Pi_{\mathcal{C}}(m))^s \leq \left(\frac{em}{d}\right)^{ds}$$

where the second inequality comes from the polynomial bound on the $\Pi_{\mathcal{C}}(m)$ given in Section 3.4. Since S was arbitrary, this bound in fact holds for $\Pi_{\mathcal{C}_G}(m)$.

Thus in order for $\mathcal{C}_G$ to shatter m points, the inequality $(em/d)^{ds} \geq 2^m$ must hold. Conversely, if $(em/d)^{ds} < 2^m$ for some m, then m is an upper bound on $VCD(\mathcal{C}_G)$. It is easy to verify that this latter inequality holds for $m = 2ds\log(es)$ provided $s \geq 2$. □(Theorem 3.6)

To apply Theorem 3.6 to the problem of PAC learning neural networks, we simply let the function at each node in the graph G be a **linear threshold function**. If the indegree is r, such a function is defined by real **weights** $w_1, \ldots, w_r \in \Re$ and a **threshold** $\Theta \in \Re$. On inputs $x_1, \ldots, x_r \in \Re$ the function outputs 1 if $\sum_{i=1}^r w_i x_i \geq \Theta$, and outputs 0 otherwise. We call G the underlying **architecture** of the neural network.

Now as we mentioned in Section 3.3, it is known that the VC dimension of the class of linear threshold on r inputs is $r+1$. By Theorem 3.6 we find that the Vapnik-Chervonenkis of the class of neural networks with architecture G is at most $2(rs+s)\log(es)$, and combined with Theorem 3.3, we obtain:

Theorem 3.7 *Let G be any directed acyclic graph, and let $\mathcal{C}_G$ be the class of neural networks on an architecture G with indegree r and s internal nodes. Then the number of examples required to learn $\mathcal{C}_G$ is*

$$O\left(\frac{1}{\epsilon}\log\frac{1}{\delta} + \frac{(rs+s)\log s}{\epsilon}\log\frac{1}{\epsilon}\right).$$

3.8 Exercises

3.1. Compute the VC dimension of the class of boolean conjunctions of literals over $\{0,1\}^n$.

3.2. Consider the concept class over the Euclidean plane $\Re^2$ consisting of the interior regions of circles; thus, the positive examples of each concept form a disk in the plane. Compute the VC dimension of this class. Compute the VC dimension of the class of interiors of triangles in the plane.

3.3. Show that there is no 1-decision list over $\{0,1\}^n$ computing the exclusive-or function $x_1 \oplus x_2$. Then show that the VC dimension of

1-decision lists over $\{0,1\}^n$ is $\Theta(n)$, and that the VC dimension of k-decision lists is $\Theta(n^k)$. Hint: show that 1-decision lists over $\{0,1\}^n$ compute linearly separable functions (halfspaces). You may use the fact that the VC dimension of halfspaces over $\Re^n$ is linear in n.

3.4. Let $P_{d,k}$ be the class of concepts over $\Re^d$ defined by convex polytopes with k sides; thus, each the positive examples of each concept in $P_{d,k}$ are defined by the convex intersection of k halfspaces in $\Re^d$. Give the best upper and lower bounds that you can on $VCD(P_{d,k})$. You may use the fact that the VC dimension of halfspaces over $\Re^d$ is linear in d.

3.5. Let $\mathcal{C}$ be any concept class of VC dimension d over X, and let $\mathcal{D}$ be any distribution over X. Suppose we are given access to a source of random (unlabeled) instances drawn according to $\mathcal{D}$, and also access to an oracle that for any labeled sample of points will return "Yes" if there is a concept in $\mathcal{C}$ that is consistent with the labeled sample, and will return "No" otherwise. Describe an algorithm that on input any finite set of instances $S \subseteq X$ and any $\epsilon, \delta > 0$ will output either the answer "Yes, S in an ϵ-net for $\mathcal{C}$ with respect to $\mathcal{D}$", or the answer "No, S is not an $\epsilon/4$-net for $\mathcal{C}$ with respect to $\mathcal{D}$". Moreover, the algorithm must give a correct answer with probability at least $1-\delta$. The algorithm need not be efficient. (The quantity $\epsilon/4$ in the "No" condition can in fact be replaced by $\alpha\epsilon$ for any fixed constant $\alpha < 1$, giving an arbitrarily refined test.)

3.6. Prove that the bound of $\Phi_d(m)$ on $\Pi_{\mathcal{C}}(m)$ is tight: that is, for any concept class $\mathcal{C}$ of VC dimension d and any m, there exists a set S of m points such that $|\Pi_{\mathcal{C}}(S)| = \Phi_d(m)$.

3.7. In this exercise we consider the two-oracle model of PAC learning defined in Exercise 1.3 of Chapter 1. We say that a concept class $\mathcal{C}$ is **PAC learnable from positive examples alone** if it is PAC learnable by an algorithm that only draws from the oracle $EX(c, \mathcal{D}_c^+)$ when learning target concept $c \in \mathcal{C}$ (the hypothesis must still meet the two-sided error criterion). We have already seen in Chapter 1 that boolean conjunctions are efficiently PAC learnable from positive examples alone. This exercise

ignores computational considerations, and concentrates on the number of examples required for learning from positive examples alone.

We say that a subclass $\mathcal{C}' \subseteq \mathcal{C}$ has **unique negative examples** if for every $c \in \mathcal{C}'$, there is an instance $x_c \in X$ such that $x_c \notin c$ but $x_c \in c'$ for every other $c' \in \mathcal{C}'$. We define the **unique negative dimension** of the class $\mathcal{C}$, $UND(\mathcal{C})$, to be the cardinality of the largest subclass $\mathcal{C}'$ that has unique negative examples.

Prove that any algorithm learning $\mathcal{C}$ from positive examples alone (regardless of computation time or the hypothesis class used) requires $\Omega(UND(\mathcal{C})/\epsilon)$ positive examples.

Then prove that $O(UND(\mathcal{C})/\epsilon)$ positive examples are sufficient for learning from positive examples alone by the following steps. Consider the algorithm that takes a sample S of positive examples of the target concept and returns the hypothesis

$$h = \min_{\mathcal{C}}(S) = \bigcap_{c \in \mathcal{C}: S \subseteq c} c.$$

Note that h may not be contained in $\mathcal{C}$, and also that this algorithm will never err on a negative example of the target concept.

First show that if on random samples S of size d/ϵ (where $d = UND(\mathcal{C})$) from $EX(c, \mathcal{D}_c^+)$, the expected error of $\min_{\mathcal{C}}(S)$ with respect to $\mathcal{D}_c^+$ exceeds ϵ, then there must exist a set $S^* \subseteq c$ of size $d/\epsilon + 1$ with the property that for a fraction at least ϵ of the $x \in S^*$, $x \notin \min_{\mathcal{C}}(S^* - \{x\})$. Then show that this implies that $UND(\mathcal{C}) > d$, a contradiction.

Thus, $\Theta(UND(\mathcal{C})/\epsilon)$ positive examples are necessary and sufficient for learning from positive examples alone, and the unique negative dimension plays a role analogous to the Vapnik-Chervonenkis dimension for this model of learning.

3.9 Bibliographic Notes

The classic paper on the VC dimension, and the one in which the main elements of the proof of Theorem 3.3 are first introduced, is by Vapnik and Chervonenkis [95]. These ideas were introduced into the computational learning theory literature and elaborated upon in the influential work of Blumer, Ehrenfeucht, Haussler and Warmuth [22]. Vapnik has also written an excellent book [94] that greatly extends the original ideas into a theory known as structural risk minimization.

The VC dimension and its attendant theorems have been influential in the neural network and artificial intelligence machine learning communities. The calculation of the VC dimension of neural networks is due to Baum and Haussler [13], and Abu-Mostafa [1] and Tesauro and Cohn [89] examine VC dimension issues from a neural network perspective. Haussler [45] examines the VC dimension as a form of inductive bias from an artificial intelligence viewpoint.

The value of the VC dimension as a measure of the sample complexity of learning transcends the PAC model; many authors have shown that the VC dimension provides upper or lower bounds on the resources required for learning in many models. These include on-line models of learning (Haussler, Littlestone and Warmuth [51]; Maass and Turán [69]; Littlestone [66]), models of query learning (Maass and Turán [69]); and many others.

The VC dimension has also been generalized to give combinatorial complexity measures that characterize the sample complexity of learning in various extensions of the PAC model. Perhaps the most general work along these lines in the computational learning theory literature has been undertaken by Haussler [48], who draws on work in statistics, notably the work of Pollard [74] and of Dudley [31]. Haussler's general framework is examined carefully in the context of learning probabilistic concept by Kearns and Schapire [61], who prove that a certain generalization of the VC dimension provides a lower bound on sample size for learning in this

model, and by Alon et al. [4], who give an upper bound.

The VC dimension and its generalizations are only one of the many ways that computational learning theory and statistics attempt to quantify the behavior of *learning curves*, that is, the error of the hypothesis as a function of the number of examples seen. For instance, among the many alternative methods of analysis are theories based on tools from information theory and statistical physics [50, 86].

The $\Omega(d/\epsilon)$ sample size lower bound is due to Ehrenfeucht et al. [33], who also give the solution to Exercise 3.3. Exercise 3.7 is due to Gereb-Graus [39].