

Learning Boolean Functions

Martin Anthony

A chapter for *Boolean Methods and Models* (ed. Yves Crama and Peter L. Hammer)

Contents

1	Introduction	3
2	Probabilistic modelling of learning	4
2.1	A probabilistic model	4
2.2	Definitions	5
2.3	A learnability result for Boolean classes	6
2.4	Learning monomials	8
2.5	Discussion	9
3	The growth function and VC-Dimension	10
3.1	The growth function of a function class	10

3.2	VC-dimension	10
4	Relating growth function and VC-dimension	11
5	VC-dimension and PAC learning	14
5.1	Upper bounds on sample complexity	14
5.2	Lower bounds on sample complexity	14
6	VC-dimensions of Boolean classes	16
6.1	Monomials	16
6.2	Threshold functions	17
6.3	k -DNF	18
7	Efficient PAC learning	19
7.1	Introduction	19
7.2	Graded classes	19
7.3	Definition of efficient learning	20
7.4	Sufficient conditions for efficient learning	21
8	Randomized PAC and SEM algorithms	22
9	Learning and existence of SEM algorithms	23

10 Establishing hardness of learning	25
11 Hardness results	26
11.1 Threshold functions	26
11.2 k -clause CNF	29

1 Introduction

This chapter explores the *learnability* of Boolean functions. Broadly speaking, the problem of interest is how to infer information about an unknown Boolean function given only information about its values on some points, together with the information that it belongs to a particular class of Boolean functions. This broad description can encompass many more precise formulations. Here we focus on probabilistic models of learning, in which the information about the function value on points in its domain is provided through its values on some randomly drawn sample, and in which the criteria for successful ‘learning’ are defined using probability theory. Other approaches, such as ‘exact query learning’ (see [1, 17, 19], for instance) and ‘specification’, ‘testing’ or ‘learning with a helpful teacher’ (see [11, 4, 15, 20, 25]) are possible, and particularly interesting in the context of Boolean functions. We aim to give a fairly thorough account of what can be said in two probabilistic models.

In the probabilistic models discussed, there are two separate, but linked, issues of concern. First, there is the question of how much information is needed about the values of a function on points before a good approximation to the function can be found. Secondly, there is the question of how, algorithmically, we might find a good approximation to the function. These two issues are usually termed the *sample complexity* and *computational complexity* of learning. The chapter breaks fairly naturally into, first, an exploration of sample complexity and then a discussion of computational complexity.

2 Probabilistic modelling of learning

2.1 A probabilistic model

The primary probabilistic model of ‘supervised’ learning we discuss here is a variant of the ‘probably approximately correct’ (or PAC) model introduced by Valiant [30], and further developed by many others; see [31, 12, 2], for example. The probabilistic aspects of the model have their roots in work of Vapnik and Chervonenkis [32, 33], as was pointed out by [5]. Valiant’s model additionally placed considerable emphasis on the computational complexity of learning.

In the model, it is assumed that we are using some class H of Boolean functions on $X = \{0, 1\}^n$ (termed the *hypothesis space*) to find a good fit to a set of data. We assume that the (labeled) data points take the form (x, b) for $x \in \{0, 1\}^n$ and $b \in \{0, 1\}$ (though most of what we discuss will apply also to the more general case in which H maps from \mathbb{R}^n to $\{0, 1\}$ and the data are in $\mathbb{R}^n \times \{0, 1\}$). The learning model is probabilistic: we assume that we are presented with some randomly generated ‘training’ data points and that we choose a hypothesis on this basis.

The simplest assumption to make about the relationship between H and the data is that the data can indeed be exactly matched by some function in H , by which we mean that each data point takes the form $(x, t(x))$ for some fixed $t \in H$ (the *target concept*). In this *realizable* case, we assume that some number m of (labeled) data points (or *labeled examples*) are generated to form a *training sample* $\mathbf{s} = ((x_1, t(x_1)), \dots, (x_m, t(x_m)))$ as follows: each x_i is chosen independently according to some fixed probability distribution μ on X . The learning problem is then, given only \mathbf{s} , and the knowledge that the data are labeled according to *some* target concept in H , to produce some $h \in H$ which is ‘close’ to t (in a sense to be formalized below).

A more general framework can usefully be developed to model the case in which the data cannot necessarily be described completely by a function in H , or, indeed, when there is a stochastic, rather than deterministic, labelling of the data points. In this more general formulation, it is assumed that the data points (x, b) in the training sample are generated according to some probability distribution P on the product $X \times \{0, 1\}$. This formulation includes the realizable case just described, but also permits a given x to appear with the

two different labels 0 and 1, each with certain probability. The aim of learning in this case is to find a function from H that is a good predictor of the data labels (something we will shortly make precise). It is hoped that such a function can be produced given only the training sample.

2.2 Definitions

We now formalize these outline descriptions of what is meant by learning. We place most emphasis on the more general framework, the realizable one being a special case of this.

A training sample is some element of Z^m , for some $m \geq 1$, where $Z = X \times \{0, 1\}$. We may therefore regard a learning algorithm as a function $L : Z^* \rightarrow H$ where $Z^* = \bigcup_{m=1}^{\infty} Z^m$ is the set of all possible training samples. (It is conceivable that we might want to define L only on part of this domain. But we could easily extend its domain to the whole of Z^* by assuming some default output in cases outside the domain of interest.) We denote by $L(\mathbf{s})$ the *output hypothesis* of the learning algorithm after being presented with training sample \mathbf{s} .

Since there is assumed to be some probability distribution, P , on the set $Z = X \times \{0, 1\}$ of all examples, we may define the *error*, $er_P(h)$, of a function h (with respect to P) to be the P -probability that, for a randomly chosen example, the label is not correctly predicted by h . In other words, $er_P(h) = P(\{(x, b) \in Z : h(x) \neq b\})$.

The aim is to ensure that the error of $L(\mathbf{s})$ is ‘usually near-optimal’ provided the training sample is ‘large enough’. Since each of the m examples in the training sample is drawn randomly and independently according to P , the sample \mathbf{s} is drawn randomly from Z^m according to the product probability distribution P^m . Thus, more formally, we want it to be true that with high P^m -probability the sample \mathbf{s} is such that the output function $L(\mathbf{s})$ has near-optimal error with respect to P . The smallest the error could be is $\text{opt}_P(H) = \min\{er_P(h) : h \in H\}$. (For a class of Boolean functions, since H is finite, the minimum is defined, but in general we would use the infimum.)

This leads us to the following formal definition of a version of ‘PAC’, (probably approximately correct) learning.

Definition 2.1 (PAC learning) *The learning algorithm L is a PAC-learning algorithm for the class H of Boolean functions if for any given $\delta, \epsilon > 0$ there is a sample length $m_0(\delta, \epsilon)$ such that for all probability distributions P on $Z = X \times \{0, 1\}$,*

$$m > m_0(\delta, \epsilon) \Rightarrow P^m(\{\mathbf{s} \in Z^m : \text{er}_P(L(\mathbf{s})) \geq \text{opt}_P(H) + \epsilon\}) < \delta.$$

The smallest suitable value of $m_0(\delta, \epsilon)$, denoted $m_L(\delta, \epsilon)$, is called the *sample complexity* of L .

The definition is fairly easy to understand in the realizable case. In this case, $\text{er}_P(h)$ is the probability that a hypothesis h disagrees with the target concept t on a randomly chosen example. So, here, informally speaking, a learning algorithm is PAC if, provided a random sample is long enough (where ‘long enough’ is independent of P), then it is ‘probably’ the case that after training on that sample, the output hypothesis is ‘approximately’ correct. We often refer to ϵ as the *accuracy parameter* and δ as the *confidence parameter*.

Note that the probability distribution P occurs twice in the definition: first in the requirement that the P^m -probability of a sample be small and secondly through the fact that the error of $L(\mathbf{s})$ is measured with reference to P . The crucial feature of the definition is that we require that the sample length $m_0(\delta, \epsilon)$ be independent of P .

2.3 A learnability result for Boolean classes

For $h \in H$ and $\mathbf{s} = ((x_1, b_1), \dots, (x_m, b_m))$, the *sample error* of h on \mathbf{s} is

$$\hat{\text{er}}_{\mathbf{s}}(h) = \frac{1}{m} |\{i : h(x_i) \neq b_i\}|,$$

and we say that L is a SEM (sample-error minimization) algorithm if, for any \mathbf{s} ,

$$\hat{\text{er}}_{\mathbf{s}}(L(\mathbf{s})) = \min\{\hat{\text{er}}_{\mathbf{s}}(h) : h \in H\}.$$

We now show that L is a PAC learning algorithm provided it has this fairly natural property.

Theorem 2.2 *Any SEM learning algorithm L for a set H of Boolean functions is PAC. Moreover, the sample complexity is bounded as follows:*

$$m_L(\delta, \epsilon) \leq \frac{2}{\epsilon^2} \ln \left(\frac{2|H|}{\delta} \right).$$

Proof: By Hoeffding's inequality [13], for any particular $h \in H$,

$$P^m (|\hat{\text{er}}_{\mathbf{s}}(h) - \text{er}_P(h)| \geq \epsilon/2) \leq 2 \exp(-\epsilon^2 m/2).$$

So, for any P and ϵ ,

$$\begin{aligned} P^m \left(\max_{h \in H} |\hat{\text{er}}_{\mathbf{s}}(h) - \text{er}_P(h)| \geq \epsilon/2 \right) &= P^m \left(\bigcup_{h \in H} \{ \mathbf{s} \in Z^m : |\hat{\text{er}}_{\mathbf{s}}(h) - \text{er}_P(h)| \geq \epsilon/2 \} \right) \\ &\leq \sum_{h \in H} P^m (|\hat{\text{er}}_{\mathbf{s}}(h) - \text{er}_P(h)| \geq \epsilon/2) \\ &\leq |H| 2 \exp(-\epsilon^2 m/2). \end{aligned}$$

as required. Now suppose $h^* \in H$ is such that $\text{er}_P(h^*) = \text{opt}_P(H)$. Then

$$P^m \left(\max_{h \in H} |\hat{\text{er}}_{\mathbf{s}}(h) - \text{er}_P(h)| \geq \epsilon/2 \right) \leq 2|H| \exp(-\epsilon^2 m/2),$$

and this is no more than δ if $m \geq (2/\epsilon^2) (2|H|/\delta)$. In this case, with probability at least $1 - \delta$, for every $h \in H$, $\text{er}_P(h) - \epsilon/2 < \hat{\text{er}}_{\mathbf{s}}(h) < \text{er}_P(h) + \epsilon/2$, and so,

$$\begin{aligned} \text{er}_P(L(\mathbf{s})) &\leq \hat{\text{er}}_{\mathbf{s}}(L(\mathbf{s})) + \epsilon/2 = \min_{h \in H} \hat{\text{er}}_{\mathbf{s}}(h) + \epsilon/2 \leq \hat{\text{er}}_{\mathbf{s}}(h^*) + \epsilon/2 \\ &< (\text{er}_P(h^*) + \epsilon/2) + \epsilon/2 = \text{opt}_P(H) + \epsilon. \end{aligned}$$

The result follows. □

We have stated the result for classes of Boolean functions, but it clearly applies also to *finite* classes of $\{0, 1\}$ -valued functions defined on \mathbb{R}^n .

The proof of Theorem 2.2 shows that, for any $m > 0$, with probability at least $1 - \delta$, L returns a function h with

$$\text{er}_P(h) < \text{opt}_P(H) + \sqrt{\frac{2}{m} \ln \left(\frac{2|H|}{\delta} \right)}.$$

Thus, $\epsilon_0(\delta, m) = \sqrt{\frac{2}{m} \ln \left(\frac{2|H|}{\delta} \right)}$ may be thought of as a bound on the *estimation error* of the learning algorithm. The definitions and results can easily be stated in terms of estimation error rather than sample complexity, but here we will mostly use sample complexity.

We state, without its proof (which is, in any case, simpler than the one just given, and may be found in [5]), the following result for the realizable case. Note that, in the realizable case, the optimal error is zero, so a SEM algorithm is what is called a *consistent* algorithm. That is, the output hypothesis h is consistent with the sample, meaning that $h(x_i) = t(x_i)$ for each i , where t is the target concept.

Theorem 2.3 *Suppose that H is a set of Boolean functions. Then, for any m and δ , and any target concept $t \in H$, the following holds with probability at least $1 - \delta$: if $h \in H$ is any hypothesis consistent with a training sample \mathbf{s} of length m , then with probability at least $1 - \delta$,*

$$\text{er}_P(h) < \frac{1}{m} \ln \left(\frac{|H|}{\delta} \right).$$

In particular, for realizable learning problems, any consistent learning algorithm L is PAC and has sample complexity bounded as follows: $m_L(\delta, \epsilon) \leq (1/\epsilon) \ln (|H|/\delta)$.

2.4 Learning monomials

We give a simple example of a PAC algorithm in the realizable case. A *monomial* is a Boolean function which can be represented by a formula that is a simple conjunction of literals. There is a very simple learning algorithm for monomials, due to Valiant [30]. We begin with no information, so we assume that every one of the $2n$ literals $u_1, \bar{u}_1, \dots, u_n, \bar{u}_n$ can occur in the target monomial. On presentation of a positive example $(x, 1)$, the algorithm deletes literals as necessary to ensure that the current hypothesis monomial is true on the example. The algorithm takes no action on negative examples: it will always be the case that the current hypothesis correctly classifies such examples as false points. The formal description is as follows. Suppose we are given a training sample \mathbf{s} containing the labeled examples (x_i, b_i) ($1 \leq i \leq m$), where each example x_i is an n -tuple of bits

$(x_i)_j$. If we let h_U denote the monomial formula containing the literals in the set U , the algorithm can be expressed as follows.

```

set  $U := \{u_1, \bar{u}_1, \dots, u_n, \bar{u}_n\}$ ;
for  $i := 1$  to  $m$  do
  if  $b_i = 1$  then
    for  $j := 1$  to  $n$  do
      if  $(x_i)_j = 1$  then delete  $\bar{u}_j$  if present in  $U$ 
      else delete  $u_j$  if present in  $U$ ;
 $L(\mathbf{s}) := h_U$ 

```

It is easy to check that if \mathbf{s} is a training sample corresponding to a monomial, then the algorithm outputs a monomial consistent with \mathbf{s} . So the algorithm is a PAC algorithm for the realizable case. Furthermore, since the number of monomials is at most $3^n + 1$ (noting that each literal may appear non-negated, negated, or not at all, and that the identically-0 function can also be thought of as a monomial), the sample complexity of L is bounded above by

$$\frac{1}{\epsilon} \ln \left(\frac{3^n + 1}{\delta} \right),$$

which, ignoring constants, is of order $(n + \ln(1/\delta)) / \epsilon$. The algorithm is also computationally efficient, something we shall turn our attention to later.

2.5 Discussion

Theorem 2.2 and Theorem 2.3 show that the sample complexity of learning can be bounded above using the cardinality of H . But it is natural to ask if one can do better: that is, can we obtain tighter upper bounds? Furthermore, we have not yet seen any lower bounds on the sample complexity of learning. To deal with these concerns, we now look at the *VC-dimension*, which turns out to give (often better) upper bounds, and also lower bounds, on sample complexity.

3 The growth function and VC-Dimension

3.1 The growth function of a function class

Suppose that H is a set of Boolean functions defined on $X = \{0, 1\}^n$. Let $\mathbf{x} = (x_1, x_2, \dots, x_m)$ be a sample (unlabeled) of length m of points of X . As in [33, 5], we define $\Pi_H(\mathbf{x})$, the *number of classifications of \mathbf{x} by H* , to be the number of distinct vectors of the form

$$(f(x_1), f(x_2), \dots, f(x_m)),$$

as f runs through all functions of H . (This definition works more generally if H is a set of $\{0, 1\}$ -valued functions defined on some \mathbb{R}^n , for although in this case H may be infinite, $\Pi_H(\mathbf{x})$ will be finite.) Note that for any sample \mathbf{x} of length m , $\Pi_H(\mathbf{x}) \leq 2^m$. An important quantity, and one which turns out to be crucial in PAC learning theory, is the maximum possible number of classifications by H of a sample of a given length. We define the *growth function* Π_H by

$$\Pi_H(m) = \max \{ \Pi_H(\mathbf{x}) : \mathbf{x} \in X^m \}.$$

We have used the notation Π_H for both the number of classifications and the growth function, but this should cause no confusion.

3.2 VC-dimension

We noted that the number of possible classifications by H of a sample of length m is at most 2^m , this being the number of binary vectors of length m . We say that a sample \mathbf{x} of length m is *shattered* by H , or that H *shatters* \mathbf{x} , if this maximum possible value is attained; that is, if H gives all possible classifications of \mathbf{x} . We shall also find it useful to talk of a set of points, rather than a sample, being shattered. The notion is the same: the set is shattered if and only if a sample with those entries is shattered. To be shattered, \mathbf{x} must clearly have m distinct examples. Then, \mathbf{x} is shattered by H if and only if for each subset S of $\{x_1, x_2, \dots, x_m\}$, there is some function f_S in H such that for $1 \leq i \leq m$, $f_S(x_i) = 1 \iff x_i \in S$.

Consistent with the intuitive notion that a set H of functions has high expressive power if it can achieve all possible classifications of a large set of examples, following [33, 5], we

use as a measure of this power the *Vapnik-Chervonenkis dimension*, or *VC-dimension*, of H , which is defined to be the maximum length of a sample shattered by H . Using the notation introduced above, we can say that the VC-dimension of H , denoted $\text{VCdim}(H)$, is given by

$$\text{VCdim}(H) = \max \{m : \Pi_H(m) = 2^m\},$$

We may state this definition formally, and in a slightly different form, as follows.

Definition 3.1 (VC-dimension) *Let H be a set of Boolean functions from a set X to $\{0, 1\}$. The VC-dimension of H is the maximal size of a subset E of X with the property that for each $S \subseteq E$, there is $f_S \in H$ with $f_S(x) = 1$ if $x \in S$ and $f_S(x) = 0$ if $x \in E \setminus S$.*

The VC-dimension of a set of Boolean functions can easily be bounded in terms of its cardinality.

Theorem 3.2 *For any set H of Boolean functions, $\text{VCdim}(H) \leq \log_2 |H|$.*

Proof: If d is the VC-dimension of H and $\mathbf{x} \in X^d$ is shattered by H , then $|H| \geq |H_{\mathbf{x}}| = 2^d$. (Here, $H_{\mathbf{x}}$ denotes the restriction of H to domain $E = \{x_1, x_2, \dots, x_d\}$.) It follows that $d \leq \log_2 |H|$. \square

It should be noted that Theorem 3.2 is sometimes loose, as we shall shortly see. However, it is reasonably tight: to see this, we need to explore further the relationship between growth function and VC-dimension.

Note: All of the definitions in this section can be made more generally for (possibly infinite) sets of functions mapping from $X = \mathbb{R}^n$ to $\{0, 1\}$. The VC-dimension can then be infinite. Theorem 3.2 applies to any finite such class.

4 Relating growth function and VC-dimension

The growth function $\Pi_H(m)$ is a measure of how many different classifications of an m -sample into true and false points can be achieved by the functions of H , while the

VC-dimension of H is the maximum value of m for which $\Pi_H(m) = 2^m$. Thus, the VC-dimension is defined in terms of the growth function. But there is a converse relationship: the growth function $\Pi_H(m)$ can be bounded by a polynomial function of m , and the degree of the polynomial is the VC-dimension d of H . Explicitly, we have the following theorem [23, 26], usually known as Sauer's Lemma (or the Sauer-Shelah Lemma).

Theorem 4.1 (Sauer's Lemma) *Let $d \geq 0$ and $m \geq 1$ be given integers and let H be a set of $\{0, 1\}$ -valued functions with $\text{VCdim}(H) = d \geq 1$. Then*

$$\Pi_H(m) \leq \sum_{i=0}^d \binom{m}{i} < \left(\frac{em}{d}\right)^d,$$

where the second inequality holds for $m \geq d$.

Proof: For $m \leq d$, the first inequality is trivially true since in that case the sum is 2^m . Assume that $m > d$ and fix a set $S = \{x_1, \dots, x_m\} \subseteq X$. We will make use of the correspondence between $\{0, 1\}$ -valued functions on a set and subsets of that set by defining the set system (or family of sets)

$$\mathcal{F} = \{\{x_i \in S : f(x_i) = 1\} : f \in H\}.$$

The proof proceeds, as in [28], by first creating a transformed version \mathcal{F}^* of \mathcal{F} that is a down-set with respect to the partial order induced by set-inclusion, and which has the same cardinality as \mathcal{F} . (To say that \mathcal{F}^* is a down-set means that if $A \in \mathcal{F}^*$ and $B \subseteq A$ then $B \in \mathcal{F}^*$.)

For an element x of S , let T_x denote the operator that, acting on a set system, removes the element x from all sets in the system, unless that would give a set that is already in the system:

$$T_x(\mathcal{F}) = \{A \setminus \{x\} : A \in \mathcal{F}\} \cup \{A \in \mathcal{F} : A \setminus \{x\} \in \mathcal{F}\}.$$

Note that $|T_x(\mathcal{F})| = |\mathcal{F}|$. Consider now $\mathcal{F}^* = T_{x_1}(T_{x_2}(\dots T_{x_m}(\mathcal{F}) \dots))$. Clearly, $|\mathcal{F}^*| = |\mathcal{F}|$. Furthermore, for all x in S , $T_x(\mathcal{F}^*) = \mathcal{F}^*$. Clearly, \mathcal{F}^* is a down-set. For, if it were not, there would be some $C \in \mathcal{F}^*$ and some $x \in C$ such that $C \setminus \{x\} \notin \mathcal{F}^*$. But then applying T_x would cause x to be removed from C , contradicting $T_x(\mathcal{F}^*) = \mathcal{F}^*$.

We can define the notion of shattering for a family of subsets, in the same way as for a family of $\{0, 1\}$ -valued functions. For $R \subseteq S$, we say that \mathcal{F} shatters R if $\mathcal{F} \cap R =$

$\{A \cap R : A \in \mathcal{F}\}$ is the set of all subsets of R . We next show that, whenever \mathcal{F}^* shatters a set, so does \mathcal{F} . It suffices to show that, for any $x \in S$, if $T_x(\mathcal{F})$ shatters a set, so does \mathcal{F} . So suppose that x in S , $R \subseteq S$, and $T_x(\mathcal{F})$ shatters R . If x is not in R , then, trivially, \mathcal{F} shatters R . If x is in R , then for all $A \subseteq R$ with $x \notin A$, since $T_x(\mathcal{F})$ shatters R we have $A \in T_x(\mathcal{F}) \cap R$ and $A \cup \{x\} \in T_x(\mathcal{F}) \cap R$. By the definition of T_x , this implies $A \in \mathcal{F} \cap R$ and $A \cup \{x\} \in \mathcal{F} \cap R$. This argument shows that \mathcal{F} shatters R . It follows that \mathcal{F}^* can only shatter sets of cardinality at most d . Since \mathcal{F}^* is a down-set, this means that the largest set in \mathcal{F}^* has cardinality no more than d . (For, if there were a set of cardinality $d + 1$ in \mathcal{F}^* , all its subsets would be in \mathcal{F}^* too, because \mathcal{F}^* is a down-set, and it would therefore be shattered.) We therefore have $|\mathcal{F}^*| \leq \sum_{i=0}^d \binom{m}{i}$, this expression being the number of subsets of S containing no more than d elements. The result follows, because $|\mathcal{F}| = |\mathcal{F}^*|$, and because S was chosen arbitrarily. For the second inequality, we have, as argued in [6],

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

Now, for all $x > 0$, $(1 + (x/m))^m < e^x$, so this is bounded by $(m/d)^d e^d = (em/d)^d$, giving the bound. \square

The first inequality of this theorem is tight. If H corresponds to the set system \mathcal{F} consisting of all subsets of $\{1, 2, \dots, n\}$ of cardinality at most d , then $\text{VCdim}(H) = d$ and $|\mathcal{F}|$ meets the upper bound.

Now, Theorem 4.1 has the following consequence when we use the fact that $|H| = \Pi_H(2^n)$.

Theorem 4.2 *For any class H of Boolean functions defined on $\{0, 1\}^n$,*

$$\text{VCdim}(H) \geq \frac{\log_2 |H|}{n + \log_2 e}$$

and if $\text{VCdim}(H) \geq 3$, then $\text{VCdim}(H) \geq \log_2 |H|/n$.

Given also the earlier bound, Theorem 3.2, we see that, essentially, for a Boolean class on $\{0, 1\}^n$, $\text{VCdim}(H)$ and $\log_2 |H|$ are within a factor n of each other. This gap can be real. For example, when $H = T_n$ is the class of threshold functions, then $\text{VCdim}(T_n) = n + 1$, whereas $\log_2 |T_n| > n^2/2$. (In fact, as shown by Zuev [34], $\log_2 |T_n| \sim n^2$ as $n \rightarrow \infty$.)

5 VC-dimension and PAC learning

It turns out that the VC-dimension quantifies, in a more precise way than does the cardinality of the hypothesis space, the sample complexity of PAC learning.

5.1 Upper bounds on sample complexity

The following results bound from above the sample complexity of PAC learning (in the general and realizable cases, respectively). It is obtained from a result of Vapnik and Chervonenkis [33]; see [2].

Theorem 5.1 *Suppose that H is a set of Boolean functions with VC-dimension $d \geq 1$ and let L be any SEM algorithm for H . Then L is a PAC learning algorithm for H with sample complexity bounded as follows:*

$$m_L(\delta, \epsilon) \leq m_0(\delta, \epsilon) = \frac{64}{\epsilon^2} \left(2d \ln \left(\frac{12}{\epsilon} \right) + \ln \left(\frac{4}{\delta} \right) \right).$$

In fact, it is possible (using a result of Talagrand [29]; see [2]) to obtain an upper bound of order $(1/\epsilon^2)(d + \ln(1/\delta))$. (However, the constants involved are quite large.) For the realizable case, from a result in [5], we have the following bound.

Theorem 5.2 *Suppose that H is a set of Boolean functions with VC-dimension $d \geq 1$ and let L be any consistent learning algorithm for H . Then L is a PAC learning algorithm for H in the realizable case, with sample complexity bounded as follows:*

$$m_L(\delta, \epsilon) \leq \frac{4}{\epsilon} \left(d \ln \left(\frac{12}{\epsilon} \right) + \ln \left(\frac{2}{\delta} \right) \right).$$

5.2 Lower bounds on sample complexity

The following lower bounds on sample complexity are also obtainable. (These are from [2], and similar bounds can be found in [8, 27].)

Theorem 5.3 *Suppose that H is a class of $\{0, 1\}$ -valued functions with VC-dimension d . For any PAC learning algorithm L for H , the sample complexity $m_L(\delta, \epsilon)$ of L satisfies*

$$m_L(\delta, \epsilon) \geq \frac{d}{320\epsilon^2}$$

for all $0 < \epsilon, \delta < 1/64$. Furthermore, if H contains at least two functions, we have

$$m_L(\delta, \epsilon) \geq 2 \left\lceil \frac{1 - \epsilon^2}{2\epsilon^2} \ln \left(\frac{1}{8\delta(1 - 2\delta)} \right) \right\rceil$$

for all $0 < \epsilon < 1$ and $0 < \delta < 1/4$.

The two bounds taken together imply a sample complexity lower bound of order $(1/\epsilon^2)(d + \ln(1/\delta))$. (This means that there is a constant $k > 0$, such that for ϵ and δ sufficiently small, the sample complexity is at least k times this expression.)

For the realizable case, we have the following [9].

Theorem 5.4 *Suppose that H is a class of $\{0, 1\}$ -valued functions of VC-dimension $d \geq 1$. For any PAC learning algorithm L for H in the realizable model, the sample complexity $m_L(\delta, \epsilon)$ of L satisfies $m_L(\delta, \epsilon) \geq (d - 1)/(32\epsilon)$ for all $0 < \epsilon < 1/8$ and $0 < \delta < 1/100$. Furthermore, if H contains at least three functions, then $m_L(\delta, \epsilon) > (1/2\epsilon) \ln(1/\delta)$, for $0 < \epsilon < 3/4$ and $0 < \delta < 1$.*

Thus, in the realizable case, the sample complexity of a PAC learning algorithm is at least of the order of

$$\frac{1}{\epsilon} \left(d + \ln \left(\frac{1}{\delta} \right) \right).$$

Suppose H_n is a class of Boolean functions on $\{0, 1\}^n$. Given the connections between cardinality and VC-dimension for Boolean classes, we see that any SEM algorithm is PAC and (for fixed δ) has sample complexity at least of order $\frac{\log_2 |H_n|}{n\epsilon^2}$ and at most of order $\frac{\log_2 |H_n|}{\epsilon^2} \ln \left(\frac{1}{\epsilon} \right)$. (In fact, as noted earlier, we can omit the logarithmic factor in the

upper bound at the expense of worse constants.) In the realizable case, we can similarly see that any consistent algorithm is PAC and has sample complexity of order at least $\frac{\log_2 |H_n|}{n\epsilon}$ and at most $\frac{\log_2 |H_n|}{\epsilon} \ln\left(\frac{1}{\epsilon}\right)$.

The cardinality therefore can be used to bound the sample complexity of learning, but the VC-dimension provides tighter bounds. (Moreover, the bounds based on VC-dimension remain valid if we consider not Boolean classes but classes of functions mapping from \mathbb{R}^n to $\{0, 1\}$: as long as such classes have finite VC-dimension—even if infinite cardinality—they are still learnable by SEM algorithms, or consistent algorithms in the realizable model.)

6 VC-dimensions of Boolean classes

6.1 Monomials

As an example of VC-dimension, we consider the set M_n^+ of *positive monomials*, consisting of the simple conjunctions on non-negated literals.

Theorem 6.1 *The class M_n^+ of positive monomials on $\{0, 1\}^n$ has VC-dimension n .*

Proof: Since there are 2^n such functions, we have $\text{VCdim}(M_n^+) \leq \log_2(2^n) = n$. To show that the VC-dimension is in fact exactly n , we show that there is some set $S \subseteq \{0, 1\}^n$ such that $|S| = n$ and S is shattered by M_n^+ . Let S consist of all $\{0, 1\}$ -vectors having exactly $n - 1$ entries equal to 1, and denote by x_i the element of S having a 0 in position i . Let R be any subset of S and let $h_R \in M_n^+$ be the conjunction of the literals u_j for all j such that $x_j \notin R$. Then $h_R(x) = 1$ for $x \in R$ and $h_R(x) = 0$ for $x \in S \setminus R$. This shows S is shattered. \square

6.2 Threshold functions

It is known [7] that if $T = T_n$ is the set of threshold functions on $\{0, 1\}^n$, then

$$\Pi_T(m) \leq \psi(n, m) = 2 \sum_{i=0}^n \binom{m-1}{i}.$$

This result is proved by using the classical fact [7, 24] that N hyperplanes in \mathbb{R}^n , each passing through the origin, divide \mathbb{R}^n into at most $C(N, n) = 2 \sum_{i=0}^{n-1} \binom{N-1}{i}$ regions. It follows directly from this, since $\psi(n, n+1) = 2^{n+1}$ and $\psi(n, n+2) < 2^{n+2}$, that the VC-dimension of T_n is at most $n+1$. In fact, the VC-dimension is exactly $n+1$, as we now show. (In the proof, an alternative, more direct, way of seeing that the VC-dimension is at most $n+1$ is given.)

Theorem 6.2 *The class of threshold functions on $\{0, 1\}^n$ has VC-dimension $n+1$.*

Proof: Recall that any threshold function h is described by a weight-vector $w = (w_1, w_2, \dots, w_n)$ and a threshold θ , so that $h(x) = 1$ if and only if $\sum_{i=1}^n w_i x_i \geq \theta$. Let S be any subset of $\{0, 1\}^n$ with cardinality $n+2$. By Radon's Theorem, there is a non-empty subset R of S such that $\text{conv}(R) \cap \text{conv}(S \setminus R) \neq \emptyset$, where $\text{conv}(X)$ denotes the convex hull of X . Suppose that there is a threshold function h in T_n such that R is the set of true points of h in S . We may assume that none of the points lies on the hyperplane defining h . Let H^+ be the open half-space on which h is true and H^- the open half-space on which it is false. Then $R \subseteq H^+$ and $S \setminus R \subseteq H^-$. But since half-spaces are convex subsets of \mathbb{R}^n , we then have

$$\text{conv}(R) \cap \text{conv}(S \setminus R) \subseteq H^+ \cap H^- = \emptyset,$$

which is a contradiction. It follows that no such t exists and hence S is not shattered. But since S was an arbitrary subset of cardinality $n+2$, it follows that $\text{VCdim}(T_n) \leq n+1$. Now we show that $\text{VCdim}(T_n) \geq n+1$. Let $\mathbf{0}$ denote the all-0 vector and, for $1 \leq i \leq n$, let e_i be the point with a 1 in the i th coordinate and all other coordinates 0. We shall show that T_n shatters the set $S = \{\mathbf{0}, e_1, e_2, \dots, e_n\}$. Suppose that R is any subset of S . For $i = 1, 2, \dots, n$, let

$$w_i = \begin{cases} 1, & \text{if } e_i \in R; \\ -1, & \text{if } e_i \notin R; \end{cases}$$

and let

$$\theta = \begin{cases} -1/2, & \text{if } \mathbf{0} \in R; \\ 1/2, & \text{if } \mathbf{0} \notin R. \end{cases}$$

Then it is straightforward to verify that if h is the threshold function with weight-vector w and threshold θ , then the set of true points of h in S is precisely R . Therefore S is shattered by T_n and, consequently, $\text{VCdim}(T_n) \geq n + 1$. The result now follows. \square

6.3 k -DNF

The class of k -DNF functions on $\{0, 1\}^n$ consists of all those functions representable by a DNF in which the terms are of degree at most k . Let $D_{n,k}$ denote the set of k -DNF functions of n variables. Then, for fixed k , the VC-dimension of $D_{n,k}$ is $\Theta(n^k)$, as shown in [9].

Theorem 6.3 *Let $k \in \mathbb{N}$ be fixed and let $D_{n,k}$ be the set of k -DNF functions on $\{0, 1\}^n$. Then $\text{VCdim}(D_{n,k}) = \Theta(n^k)$.*

Proof: The number of monomials or terms which are non-empty, not identically false, and of degree at most k is $\sum_{i=1}^k \binom{n}{i} 2^i$ which is, for fixed k , $O(n^k)$. Since any k -DNF formula is created by taking the disjunction of a set of such terms, the number of k -DNF formulas (and hence $|D_{n,k}|$) is $2^{O(n^k)}$. Therefore $\text{VCdim}(D_{n,k}) \leq \log_2 |D_{n,k}| = O(n^k)$.

On the other hand, we can show that the VC-dimension is $\Omega(n^k)$ by proving that a sufficiently large subset is shattered. Consider the set S of examples in $\{0, 1\}^n$ which have precisely k entries equal to 1. Then S can be shattered by $D_{n,k}$. Indeed, suppose R is any subset of S . For each $y = (y_1, y_2, \dots, y_n) \in R$, form the term that is the conjunction of those literals u_i such that $y_i = 1$. Since $y \in S$, this term has k literals; further, y is the only true point in S of this term. The disjunction of these terms, one for each member of R , is therefore a function in $D_{n,k}$ whose true points in S are precisely the members of R . Hence S is shattered by $D_{n,k}$. Now, $|S| = \binom{n}{k}$ which, for a fixed k , is $\Omega(n^k)$. \square

7 Efficient PAC learning

7.1 Introduction

Up to now, a learning algorithm has been mainly described as a function which maps training samples into output functions (or hypotheses). We will now be more specific about the computational effectiveness of learning algorithms. If the process of PAC learning by an algorithm L is to be of practical value, it should be possible to implement the algorithm ‘quickly’. We discuss what should be meant by an efficient PAC learning algorithm, and we highlight an important connection between the existence of efficient PAC learning algorithms and the existence of efficient procedures for producing hypotheses with small sample error. As mentioned earlier, computational efficiency was a key aspect of Valiant’s learning model [30], and has been much further explored for the models of this chapter. The papers [5, 22] provided some of the important initial results, and these are further explored in the books [17, 19, 3]. The treatment here follows [2].

Consider the monomial learning algorithm (for the realizable case) described earlier. This is an efficient algorithm: its running time on a training sample of m data points in $\{0, 1\}^n$ is $O(mn)$, which is linear in the size of the training sample. Furthermore, noting either that $\text{VCdim}(M_n) = O(n)$ or that $\log_2 |M_n| = O(n)$, we can see that, for given ϵ and δ , we can produce a hypothesis that, with probability at least $1 - \delta$, has accuracy ϵ , in time of order $n^2 p(1/\epsilon, \ln(1/\delta)) = q(n, 1/\epsilon, \ln(1/\delta))$, where p and q are (small degree) polynomials. This is an example of what we mean by an efficient learning algorithm: as n scales, the time taken to produce a PAC output hypothesis scales polynomially with n ; additionally, the running time is polynomial in $1/\epsilon$ and $\ln(1/\delta)$.

7.2 Graded classes

In order to enable a more general discussion of efficient learning, we introduce the idea of a *graded* function class. Suppose H_n is a set of Boolean functions defined on $\{0, 1\}^n$. Then we say that $H = \bigcup_{n=1}^{\infty} H_n$ is a *graded* hypothesis space. The reason for introducing this idea is that we want to analyse the running time (with respect to n) of what might be termed a ‘general’ learning algorithm for a graded class of Boolean functions. This is

an algorithm that works in essentially the same manner on each of the classes H_n . For example, the monomial learning algorithm works on M_n for any n in essentially the same way: there is no fundamental difference between its actions on, say, the monomials with 5 variables and those with 50 variables.

Denoting $\{0, 1\}^n \times \{0, 1\}$ by Z_n , a *learning algorithm* L for the graded space $H = \bigcup_{n=1}^{\infty} H_n$ is a mapping from $\bigcup_{n=1}^{\infty} Z_n^*$ to H with the property that if $\mathbf{s} \in Z_n^*$ then $L(\mathbf{s}) \in H_n$. The only difference between this definition and the basic notion of a learning algorithm for an ungraded class is that we have now encapsulated some sense of the ‘generality’ of the algorithm in its action over all the H_n . With this, we now state formally what is meant by a PAC learning algorithm for a graded class.

Definition 7.1 *If L is a learning algorithm for $H = \bigcup H_n$, then we say that L is PAC if for all $n \in \mathbb{N}$ and $\delta, \epsilon \in (0, 1)$, there is $m_0(n, \delta, \epsilon)$ such that if $m \geq m_0(n, \delta, \epsilon)$ then, for any probability distribution P on Z_n , if $\mathbf{s} \in Z_n^m$ is drawn randomly according to the product probability distribution P^m on Z_n^m , then with probability at least $1 - \delta$, the hypothesis $L(\mathbf{s})$ output by L satisfies $\text{er}_P(L(\mathbf{s})) < \text{opt}_P(H_n) + \epsilon$.*

7.3 Definition of efficient learning

We now assume that learning algorithms are algorithms in the proper sense (that is, that they are computable functions). Suppose that L is a learning algorithm for a graded function class $H = \bigcup H_n$. An input to L is a training sample, which consists of m labeled binary vectors of length n . It would be possible to use $m(n + 1)$ as the measure of input size, but we will find it useful to consider dependence on m and n separately. We use the notation $R_L(m, n)$ to denote the worst-case running time of L on a training sample of m points of Z_n . Clearly, n is not the only parameter with which the running time of the learning procedure as a whole should be allowed to vary, since decreasing either the confidence parameter δ or the accuracy parameter ϵ makes the learning task more difficult, requiring a larger size of sample. We shall ask that the running time of a learning algorithm L be polynomial in m , and that the sample complexity $m_L(n, \delta, \epsilon)$ depend polynomially on $1/\epsilon$, $\ln(1/\delta)$ and n . If these conditions hold, then the running time required to produce a ‘good’ output hypothesis will be polynomial in n , $\ln(1/\delta)$ and $1/\epsilon$.

We now formally define what we mean by an *efficient learning algorithm* for a graded function class.

Definition 7.2 *Let $H = \bigcup H_n$ be a graded class of Boolean functions and suppose that L is a learning algorithm for H . We say that L is efficient if:*

- *the worst-case running time $R_L(m, n)$ of L on samples $\mathbf{s} \in Z_n^m$ is polynomial in m and n , and*
- *the sample complexity $m_L(n, \delta, \epsilon)$ of L on H_n is polynomial in $n, 1/\epsilon$ and $\ln(1/\delta)$.*

We have described the outputs of learning algorithms as hypotheses. But, more precisely, they are representations of hypotheses. When discussing the complexity of learning, it is always assumed that the output lies in a representation class for the hypothesis class. All this often amounts to is that the output must be a formula of a particular type. For example, the monomial learning algorithm outputs a monomial formula (and not some other representation). This is not something we shall explore much further, but it is sometimes important.

7.4 Sufficient conditions for efficient learning

We define a SEM algorithm for a graded Boolean class H to be an algorithm that given any sample $\mathbf{s} \in Z_n^m$, returns a function $h \in H_n$ with minimal sample error $\hat{\epsilon}_{\mathbf{s}}(h)$ on \mathbf{s} . The following result, which may be found in [5], follows directly from earlier results and shows that the rate of growth with n of the VC-dimension determines the sample complexity of learning algorithms.

Theorem 7.3 *Let $H = \bigcup H_n$ be a graded Boolean function class.*

- *If $\text{VCdim}(H_n)$ is polynomial in n , then any SEM algorithm for H is a PAC learning algorithm with sample complexity $m_L(n, \delta, \epsilon)$ polynomial in $n, 1/\epsilon$ and $\ln(1/\delta)$.*

- *If there is an efficient PAC learning algorithm for H , then $\text{VCdim}(H_n)$ is polynomial in n .*

Note that, by Theorem 4.2, the same statement is true with $\text{VCdim}(H_n)$ replaced by $\ln |H_n|$.

We now turn our attention to the running time of SEM algorithms. Having seen that, in many circumstances, such algorithms yield PAC learning algorithms, we now investigate the *efficiency* of these derived learning algorithms. We say that a SEM algorithm for the graded Boolean function class $H = \bigcup H_n$ is efficient if, given as input $\mathbf{s} \in Z_n^m$, it returns its output in time polynomial in m and n . The following result is immediate.

Theorem 7.4 *Suppose that $H = \bigcup H_n$ is a graded Boolean function class and that $\text{VCdim}(H_n)$ is polynomial in n . Then, any efficient SEM algorithm for H is an efficient PAC learning algorithm for H .*

8 Randomized PAC and SEM algorithms

There may be some advantage in allowing learning algorithms to be randomized. Furthermore, as we shall see, there are some fairly succinct characterizations of learnability provided we permit algorithms to be randomized.

For our purposes, a randomized algorithm \mathcal{A} has available to it a random number generator that produces a sequence of independent, uniformly distributed bits. The randomized algorithm \mathcal{A} uses these random bits as part of its input, but it is useful to think of this input as somehow ‘internal’ to the algorithm, and to think of the algorithm as defining a mapping from an ‘external’ input to a probability distribution over outputs. The computation carried out by the algorithm is, of course, determined by its input, so that, in particular, it depends on the particular sequence produced by the random number generator, as well as on the ‘external’ input. We may speak of the ‘probability’ that \mathcal{A} has a given outcome on an (external) input x , by which we mean the probability that the stream of random numbers gives rise to that outcome when the external input to the algorithm is x . It is useful to extend our concept of a PAC learning algorithm to allow

randomization. The definition of a randomized PAC learning for a graded class is as in Definition 7.1, with the additional feature that the algorithm is randomized. (So, L can no longer be regarded as a deterministic function.)

We shall also be interested in randomized SEM algorithms.

Definition 8.1 *A randomized algorithm \mathcal{A} is an efficient randomized SEM algorithm for the graded Boolean function class $H = \bigcup H_n$ if given any $\mathbf{s} \in Z_n^m$, \mathcal{A} halts in time polynomial in n and m and outputs $h \in H_n$ which, with probability at least $1/2$, satisfies $\hat{e}_{\mathbf{s}}(h) = \min_{g \in H_n} \hat{e}_{\mathbf{s}}(g)$.*

Suppose we run a randomized SEM algorithm k times on a fixed sample (\mathbf{s}) , keeping the output hypothesis $f^{(k)}$ with minimal sample error among all the k hypotheses returned. In other words, we take the *best of k iterations* of the algorithm. Then the probability that $f^{(k)}$ has sample error that is *not* minimal is at most $(1/2)^k$. This is the basis of the following result, which shows that, as far as its applications to learning are concerned, an efficient randomized SEM algorithm is as useful as its deterministic counterpart. (The key idea in establishing this result is to take the best of k iterations of \mathcal{A} for a suitable k , absorbing the randomness in the action of \mathcal{A} into the ‘ δ ’ of learning.)

Theorem 8.2 *Suppose that $H = \bigcup H_n$ is a graded Boolean function class and that $\text{VCdim}(H_n)$ is polynomial in n . If there is an efficient randomized SEM algorithm \mathcal{A} for H , then there is an efficient randomized PAC learning algorithm for H that uses \mathcal{A} as a subroutine.*

9 Learning and existence of SEM algorithms

We have seen that efficient SEM algorithms (both deterministic and randomized) can in many cases be used to construct efficient PAC learning algorithms. The next result proves, as a converse, that if there is an efficient PAC learning algorithm for a graded class then *necessarily* there is an efficient randomized SEM algorithm. (For the realizable case, this may be found in [22, 5, 21].)

Theorem 9.1 *If there is an efficient PAC learning algorithm for the graded Boolean class $H = \bigcup H_n$, then there is an efficient randomized SEM algorithm.*

Proof: Suppose L is an efficient PAC learning algorithm for the graded class $H = \bigcup H_n$. We construct a randomized algorithm \mathcal{A} , which will turn out to be an efficient randomized SEM algorithm. Suppose the sample $\mathbf{s} \in Z_n^m$ is given as input to \mathcal{A} . Let P be the probability distribution that is uniform on the labeled examples in \mathbf{s} and zero elsewhere on Z_n . (This probability is defined with multiplicity; that is, for instance, if there are two labeled examples in \mathbf{s} each equal to z , we assign the labeled example z probability $2/m$ rather than $1/m$.) We use the randomization allowed in \mathcal{A} to form a sample of length $m^* = m_L(n, 1/2, 1/m)$, in which each labeled example is drawn according to P . Let \mathbf{s}^* denote the resulting sample. Feeding \mathbf{s}^* into the learning algorithm, we receive as output $h^* = L(\mathbf{s}^*)$ and we take this to be the output of the algorithm \mathcal{A} ; that is, $\mathcal{A}(\mathbf{s}) = h^* = L(\mathbf{s}^*)$. By the fact that L is a PAC learning algorithm, and given that $m^* = m_L(n, 1/2, 1/m)$, with probability at least $1/2$, we have $\text{er}_P(h^*) < \text{opt}_P(H) + 1/m$. But because P is discrete, with no probability mass less than $1/m$, this means $\text{er}_P(h^*) = \text{opt}_P(H)$. For any h , by the definition of P , $\text{er}_P(h) = \hat{\text{er}}_{\mathbf{s}}(h)$. So with probability at least $1/2$,

$$\hat{\text{er}}_{\mathbf{s}}(h^*) = \text{er}_P(h^*) = \text{opt}_P(H) = \min_{g \in H_n} \text{er}_P(g) = \min_{g \in H_n} \hat{\text{er}}_{\mathbf{s}}(g).$$

This means that \mathcal{A} is a randomized SEM algorithm. Because L is efficient, $m^* = m_L(n, 1/2, 1/m)$ is polynomial in n and m . Since the sample \mathbf{s}^* has length m^* , and since L is efficient, the time taken by L to produce h^* is polynomial in m^* and n . Hence \mathcal{A} has running time polynomial in n and m , as required. \square

We arrive at the following succinct characterization of PAC learnability (allowing randomized algorithms).

Theorem 9.2 *Suppose that $H = \bigcup H_n$ is a graded Boolean function class. Then there is an efficient randomized PAC learning algorithm for H if and only if $\text{VCdim}(H_n)$ is polynomial in n and there is an efficient randomized SEM algorithm for H .*

Given the connection of Theorem 4.2 between cardinality and VC-dimension, the same statement with $\ln |H_n|$ replacing $\text{VCdim}(H_n)$ holds. (It should be noted, however, that Theorem 9.2 holds, more generally, in the case where H_n maps from \mathbb{R}^n to $\{0, 1\}$.)

10 Establishing hardness of learning

There are two quite natural decision problems associated with a graded Boolean function class $H = \bigcup H_n$:

H-FIT

Instance: $\mathbf{s} \in Z_n^m = (\{0, 1\}^n \times \{0, 1\})^m$ and an integer k between 1 and m .

Question: Is there $h \in H_n$ such that $\hat{e}_{\mathbf{s}}(h) \leq k/m$?

H-CONSISTENCY

Instance: $\mathbf{s} \in Z_n^m = (\{0, 1\}^n \times \{0, 1\})^m$.

Question: Is there $h \in H_n$ such that $\hat{e}_{\mathbf{s}}(h) = 0$?

Clearly *H-CONSISTENCY* is a sub-problem of *H-FIT*, obtained by setting $k = 0$. Thus, any algorithm for *H-FIT* can be used also to solve *H-CONSISTENCY*. Note that *H-consistency* is the decision problem associated with finding an extension in H of the partially defined Boolean function described by the sample \mathbf{s} .

We say that a randomized algorithm \mathcal{A} solves a decision problem Π if the algorithm always halts and produces an output—either ‘yes’ or ‘no’—such that if the answer to Π on the given instance is ‘no’, the output of \mathcal{A} is ‘no’, and if the answer to Π on the given instance is ‘yes’ then, with probability at least $1/2$, the output of \mathcal{A} is ‘yes’. A randomized algorithm is *polynomial-time* if its worst-case running time (over all instances) is polynomial in the size of its input. The class of decision problems Π that can be solved by a polynomial-time randomized algorithm is denoted by RP. One approach to proving that PAC learning is computationally intractable for particular classes (in the general or realizable cases) is through showing that these decision problems are hard. The reason is given in the following results. First, we have the following [18, 14].

Theorem 10.1 *Let $H = \bigcup H_n$ be a graded Boolean function class. If there is an efficient learning algorithm for H then there is a polynomial-time randomized algorithm for *H-FIT*; in other words, *H-FIT* is in RP.*

Proof: If H is efficiently learnable then, by Theorem 9.1, there exists an efficient randomized SEM algorithm \mathcal{A} for H . Using \mathcal{A} , we construct a polynomial-time randomized

algorithm \mathcal{B} for H -FIT as follows. Suppose that $\mathbf{s} \in Z_n^m$ and k together constitute an instance of H -FIT, and hence an input to \mathcal{B} . The first step of the algorithm \mathcal{B} is to compute $h = \mathcal{A}(\mathbf{s})$, the output of \mathcal{A} on \mathbf{s} . This function belongs to H_n and, with probability at least $1/2$, $\hat{e}_{\mathbf{s}}(h)$ is minimal among all functions in H_n . The next step in \mathcal{B} is to check whether $\hat{e}_{\mathbf{s}}(h) \leq k/m$. If so, then the output of \mathcal{B} is ‘yes’ and, if not, the output is ‘no’. It is clear that \mathcal{B} is a randomized algorithm for H -FIT. Furthermore, since \mathcal{A} runs in time polynomial in m and n , and since the time taken for \mathcal{B} to calculate $\hat{e}_{\mathbf{s}}(h)$ is linear in the size of \mathbf{s} , \mathcal{B} is a polynomial-time algorithm. \square

The following result [22] applies to the realizable case.

Theorem 10.2 *Suppose that $H = \bigcup H_n$ is a graded Boolean function class. If H is efficiently learnable in the realizable model, then there is a polynomial-time randomized algorithm for H -CONSISTENCY; that is, H -CONSISTENCY is in RP.*

In particular, therefore, we have the following.

Theorem 10.3 *Suppose $RP \neq NP$. If H -FIT is NP-hard, then there is no efficient PAC learning algorithm for H . Furthermore, if H -CONSISTENCY is NP-hard then there is no efficient PAC learning algorithm for H in the realizable case.*

11 Hardness results

We now use the theory just developed to show that PAC learnability of threshold functions is computationally intractable (although it is tractable in the realizable case). We also show the intractability of PAC learning a particular class of Boolean functions in the realizable case.

11.1 Threshold functions

First, we note that it is well-known that if T_n is the set of threshold Boolean functions on $\{0, 1\}^n$, then the graded class $T = \bigcup T_n$ is efficiently PAC learnable in the realizable case.

Indeed, the VC-dimension of T_n is $n + 1$, which is linear, and there exist SEM algorithms based on linear programming. (See [5, 2], for instance.) However, T is *not* efficiently PAC learnable in the general case, if $\text{RP} \neq \text{NP}$. This arises from the following result [10, 16, 14].

Theorem 11.1 *Let $T = \bigcup T_n$ be the graded class of threshold functions. Then T -FIT is NP-hard.*

We prove this by establishing that the problem it is at least as hard as the well-known NP-hard VERTEX COVER problem in graph theory.

We denote a typical graph by $G = (V, E)$, where V is the set of vertices and E the edges. We shall assume that the vertices are labeled with the numbers $1, 2, \dots, n$. Then, a typical edge $\{i, j\}$ will, for convenience, be denoted by ij . A *vertex cover* of the graph is a set U of vertices such that for each edge ij of the graph, at least one of the vertices i, j belongs to U . The following decision problem is known to be NP-hard [10].

VERTEX COVER

Instance: A graph $G = (V, E)$ and an integer $k \leq |V|$.

Question: Is there a vertex cover $U \subseteq V$ such that $|U| \leq k$?

A typical instance of VERTEX COVER is a graph $G = (V, E)$ together with an integer $k \leq |V|$. We assume, for simplicity, that $V = \{1, 2, \dots, n\}$ and we denote the number of edges, $|E|$, by r . Notice that the size of an instance of VERTEX COVER is $\Omega(r + n)$. We construct $\mathbf{s} = \mathbf{s}(G) \in (\{0, 1\}^{2n} \times \{0, 1\})^{2r+n}$ as follows. For any two distinct integers i, j between 1 and $2n$, let $e_{i,j}$ denote the binary vector of length $2n$ with ones in positions i and j and zeroes elsewhere. The sample $\mathbf{s}(G)$ consists of the labeled examples $(e_{i,n+i}, 1)$ for $i = 1, 2, \dots, n$ and, for each edge $ij \in E$, the labeled examples $(e_{i,j}, 0)$ and $(e_{n+i,n+j}, 0)$. Note that the ‘size’ of \mathbf{s} (that is, its size as measured by the number of bits needed to describe it) is $(2r + n)(2n + 1)$, which is polynomial in the size of the original instance of VERTEX COVER, and that $\mathbf{s}(G)$ can be computed in polynomial time.

For example, if a graph G has vertex set $V = \{1, 2, 3, 4\}$ and edge set $E = \{12, 23, 14, 13\}$, then the sample $\mathbf{s}(G)$ consists of the following 12 labeled examples:

$$\begin{aligned} & (10001000, 1), (01000100, 1), (00100010, 1), (00010001, 1), \\ & (11000000, 0), (00001100, 0), (01100000, 0), (00000110, 0), \\ & (10100000, 0), (00001010, 0), (10010000, 0), (00001001, 0). \end{aligned}$$

Lemma 11.2 *Given any graph $G = (V, E)$ with n vertices, and any integer $k \leq n$, let $\mathbf{s} = \mathbf{s}(G)$ be as defined above. Then, there is $h \in T_{2n}$ such that $\hat{e}_{\mathbf{s}}(h) \leq k/(2r + n)$ if and only if there is a vertex cover of G of cardinality at most k .*

Proof: Recall that any threshold function is represented by some weight vector w and threshold θ . Suppose first that there is such an h and that this is represented by the weight-vector $w = (w_1, w_2, \dots, w_{2n})$ and threshold θ . We construct a subset U of V as follows. If $h(e_{i,n+i}) = 0$, then we include i in U ; if, for $i \neq j$, $h(e_{i,j}) = 1$ or $h(e_{n+i,n+j}) = 1$ then we include *either one* of i, j in U . Because h is ‘wrong’ on at most k of the examples in \mathbf{s} , the set U consists of at most k vertices. We claim that U is a vertex cover. To show this, we need to verify that given any edge $ij \in E$, at least one of i, j belongs to U . It is clear from the manner in which U is constructed that this is true if either $h(e_{i,n+i}) = 0$ or $h(e_{j,n+j}) = 0$, so suppose that neither of these holds; in other words, suppose that $h(e_{i,n+i}) = 1 = h(e_{j,n+j})$. Then we may deduce that

$$w_i + w_{n+i} \geq \theta, \quad w_j + w_{n+j} \geq \theta,$$

and so

$$w_i + w_j + w_{n+i} + w_{n+j} \geq 2\theta;$$

that is,

$$(w_i + w_j) + (w_{n+i} + w_{n+j}) \geq 2\theta.$$

From this, we see that either $w_i + w_j \geq \theta$ or $w_{n+i} + w_{n+j} \geq \theta$ (or both); thus, $h(e_{i,j}) = 1$ or $h(e_{n+i,n+j}) = 1$, or both. Because of the way in which U is constructed, it follows that at least one of the vertices i, j belongs to U . Since ij was an arbitrary edge of the graph, this shows that U is indeed a vertex cover.

We now show, conversely, that if there is a vertex cover of G consisting of at most k vertices, then there is a function in T_{2n} with sample error at most $k/(2r + n)$ on $\mathbf{s}(G)$. Suppose U is a vertex cover and $|U| \leq k$. Define a weight-vector $w = (w_1, w_2, \dots, w_{2n})$ and threshold θ as follows: let $\theta = 1$ and, for $i = 1, 2, \dots, n$,

$$w_i = w_{n+i} = \begin{cases} -1 & \text{if } i \in U \\ 1 & \text{if } i \notin U. \end{cases}$$

We claim that if h is the threshold function represented by w and θ , then $\hat{e}_{\mathbf{s}}(h) \leq k/(2r + n)$. Observe that if $ij \in E$, then, since U is a vertex cover, at least one of i, j belongs to U and hence the inner products $w^T e_{i,j}$ and $w^T e_{n+i,n+j}$ are both either 0 or

-2 , less than θ , so $h(e_{i,j}) = h(e_{n+i,n+j}) = 0$. The function h is therefore correct on all the examples in $\mathbf{s}(G)$ arising from the edges of G . We now consider the other types of labeled example in $\mathbf{s}(G)$: those of the form $(e_{i,n+i}, 1)$. Now, $w^T e_{i,n+i}$ is -2 if $i \in U$ and is 2 otherwise, so $h(e_{i,n+i}) = 0$ if $i \in U$ and $h(e_{i,n+i}) = 1$ otherwise. It follows that h is ‘wrong’ only on the examples $e_{i,n+i}$ for $i \in U$ and hence

$$\hat{\text{er}}_{\mathbf{s}}(h) = \frac{|U|}{2r+n} \leq \frac{k}{2r+n},$$

as claimed. □

This result shows that the answer to T -FIT on the instance $(\mathbf{s}(G), k)$ is the same as the answer to VERTEX COVER on instance (G, k) . Given that $\mathbf{s}(G)$ can be computed from G in time polynomial in the size of G , we have therefore established that T -FIT is NP-hard.

11.2 k -clause CNF

Pitt and Valiant [22] were the first to give an example of a Boolean class H for which the consistency problem H -CONSISTENCY is NP-hard. Let C_n^k , the set of k -clause CNF functions, be the set of Boolean functions on $\{0,1\}^n$ that can be represented as the conjunction of at most k clauses.

We show that, for fixed $k \geq 3$, the consistency problem for $C^k = \bigcup C_n^k$ is NP-hard. Thus, if $\text{NP} \neq \text{RP}$, then can be no efficient PAC learning algorithm for C^k in the realizable case.

The reduction in this case is from GRAPH k -COLORABILITY. Suppose we are given a graph $G = (V, E)$, with $V = \{1, 2, \dots, n\}$. We construct a training sample $\mathbf{s}(G)$, as follows. For each vertex $i \in V$ we take as a negative example the vector v_i which has 1 in the i th coordinate position and 0’s elsewhere. For each edge $ij \in E$ we take as a positive example the vector $v_i + v_j$.

Lemma 11.3 *There is a function in C_n^k which is consistent with the training sample $\mathbf{s}(G)$ if and only if the graph G is k -colorable.*

Proof: Suppose that $h \in C_n^k$ is consistent with the training sample. By definition, h is

a conjunction

$$h = h_1 \wedge h_2 \wedge \dots \wedge h_k$$

of clauses. For each vertex i of G , $h(v_i) = 0$, and so there must be at least one clause h_f ($1 \leq f \leq k$) for which $h_f(v_i) = 0$. Thus we may define a function χ from V to $\{1, 2, \dots, k\}$ as follows:

$$\chi(i) = \min\{f : h_f(v_i) = 0\}.$$

We claim that χ is a coloring of G . Suppose that $\chi(i) = \chi(j) = f$, so that $h_f(v_i) = h_f(v_j) = 0$. Since h_f is a clause, every literal occurring in it must be 0 on v_i and on v_j . Now v_i has a 1 only in the i th position, and so $h_f(v_i) = 0$ implies that the only negated literal which can occur in h_f is \bar{x}_i . Since the same is true for \bar{x}_j , we conclude that h_f contains only some literals x_l , with $l \neq i, j$. Thus $h_f(v_i + v_j) = 0$ and $h(v_i + v_j) = 0$. Now if ij were an edge of G , then we should have $h(v_i + v_j) = 1$, because we assumed that h is consistent with $\mathbf{s}(G)$. Thus ij is not an edge of G , and χ is a coloring, as claimed.

Conversely, suppose we are given a coloring $\chi : V \rightarrow \{1, 2, \dots, k\}$. For $1 \leq f \leq k$, define h_f to be the clause $\bigvee_{\chi(i) \neq f} x_i$, and define $h = h_1 \wedge h_2 \wedge \dots \wedge h_k$. We claim that h is consistent with $\mathbf{s}(G)$.

First, given a vertex i suppose that $\chi(i) = g$. The clause h_g is defined to contain only those (non-negated) literals corresponding to vertices *not* colored g , and so x_i does not occur in h_g . It follows that $h_g(v_i) = 0$ and $h(v_i) = 0$. Secondly, let ij be any edge of G . For each color f , there is at least one of i, j which is not colored f ; denote an appropriate choice by $i(f)$. Then h_f contains the literal $x_{i(f)}$, which is 1 on $v_i + v_j$. Thus every clause h_f is 1 on $v_i + v_j$, and $h(v_i + v_j) = 1$, as required. \square

Note that when $k = 1$, we have $C_n^1 = C_n$, and there is a polynomial time learning algorithm for C_n dual to the monomial learning algorithm. The consistency problem (and hence intractability of learning) remains, however, when $k = 2$: to show this, the consistency problem can be related to the NP-complete SET-SPLITTING problem; see [22].

This hardness result is ‘representation-dependent’: part of the difficulty arises from the need to output a formula in k -clause-CNF form. Now, any k -clause-CNF formula can simply be rewritten as an equivalent k -DNF formula. So any function in C_n^k is also a k -DNF function; that is, using the notation from earlier, it belongs to $D_{n,k}$. But there is a simple efficient PAC learning algorithm for $D_{n,k}$; see [30, 3]. So C_n^k is learnable if the output hypotheses are permitted to be drawn from the larger class $D_{n,k}$ (or, more

precisely, if the output formula is a k -DNF).

Acknowledgements

I am grateful to a referee for several useful comments. This work is supported in part by the IST Programme of the European Community, under the PASCAL Network of Excellence, IST-2002-506778.

References

- [1] D. Angluin. Queries and concept learning. *Machine Learning*, 2(4), 1988: 319–342.
- [2] M. Anthony and P. L. Bartlett. *Neural Network Learning: Theoretical Foundations*. Cambridge University Press, 1999.
- [3] Martin Anthony and Norman L. Biggs. *Computational Learning Theory: An Introduction*. Cambridge Tracts in Theoretical Computer Science, 30, 1992. Cambridge University Press, Cambridge, UK.
- [4] M. Anthony, G. Brightwell and J. Shawe-Taylor. On specifying Boolean functions by labelled examples. *Discrete Applied Mathematics*, 61, 1995: 1–25.
- [5] A. Blumer, A. Ehrenfeucht, D. Haussler, and M. K. Warmuth: Learnability and the Vapnik-Chervonenkis dimension. *Journal of the ACM*, 36(4), 1989: 929–965.
- [6] S. Chari, P. Rohatgi and A. Srinivasan. Improved algorithms via approximations of probability distributions (Extended Abstract). In *Proceedings of the Twenty-Sixth Annual ACM Symposium on the Theory of Computing*, ACM Press, New York, NY, 1994: 584–592.
- [7] T. M. Cover. Geometrical and Statistical Properties of Systems of Linear Inequalities with Applications in Pattern Recognition. *IEEE Trans. on Electronic Computers*, EC-14, 1965: 326–334.

- [8] L. Devroye and G. Lugosi. Lower bounds in pattern recognition and learning. *Pattern Recognition* 28(7), 1995: 1011–1018.
- [9] A. Ehrenfeucht, D. Haussler, M. Kearns, and L. Valiant. A general lower bound on the number of examples needed for learning. *Information and Computation*, 82, 1989: 247–261.
- [10] M. Garey and D. Johnson. *Computers and Intractability: A Guide to the Theory of NP-Completeness*. Freeman, San Francisco, 1979.
- [11] S. A. Goldman and M. J. Kearns. On the complexity of teaching. *Journal of Computer and System Sciences*, 50(1), 1995: 20–31.
- [12] D. Haussler. Decision theoretic generalizations of the PAC model for neural net and other learning applications. *Information and Computation*, 100(1), 1992: 78–150.
- [13] W. Hoeffding: Probability inequalities for sums of bounded random variables. *Journal of the American Statistical Association*, 58(301), 1963: 13–30.
- [14] K.-U. Höffgen, H. U. Simon and K. S. Van Horn. Robust trainability of single neurons. *Journal of Computer and System Sciences*, 50(1), 1995: 114–125.
- [15] J. Jackson and A. Tomkins. A computational model of teaching. Proceedings of 5th Annual Workshop on Comput. Learning Theory, ACM Press, New York, NY, 1992: 319–326.
- [16] D. S. Johnson and F. P. Preparata. The densest hemisphere problem. *Theoretical Computer Science*, 6, 1978: 93–107.
- [17] M. J. Kearns. *The Computational Complexity of Machine Learning*. ACM Distinguished Dissertation Series. The MIT Press, Cambridge, MA., 1989.
- [18] M. J. Kearns, R. E. Schapire and L. M. Sellie. Toward efficient agnostic learning. *Machine Learning* 17(2/3), 1994: 115–142.
- [19] M. J. Kearns and U. Vazirani. *Introduction to Computational Learning Theory*, MIT Press, Cambridge, MA, 1995.
- [20] S. A. Goldman and H. D. Mathias. Teaching a smarter learner. *Journal of Computer and System Sciences*, 52(2), 1996: 255–267.

- [21] B. K. Natarajan. On learning sets and functions. *Machine Learning*, 4(1), 1989: 67–97.
- [22] L. Pitt and L. Valiant. Computational limitations on learning from examples. *Journal of the ACM*, 35, 1988: 965–984.
- [23] N. Sauer. On the density of families of sets. *Journal of Combinatorial Theory (A)*, 13, 1972: 145–147.
- [24] L. Schläfli. *Gesammelte Mathematische Abhandlungen I*, Birkhäuser, Basel, 1950.
- [25] R. Servedio. On the Limits of Efficient Teachability. *Information Processing Letters*, 79(6), 2001: 267–272.
- [26] S. Shelah. A combinatorial problem: Stability and order for models and theories in infinitary languages. *Pacific Journal of Mathematics*, 41, 1972: 247–261.
- [27] H. U. Simon. General bounds on the number of examples needed for learning probabilistic concepts. *Journal of Computer and System Sciences*, 52(2), 1996: 239–254.
- [28] J. M. Steele. Existence of submatrices with all possible columns. *Journal of Combinatorial Theory, Series A*, 24, 1978: 84–88.
- [29] M. Talagrand. Sharper bounds for Gaussian and empirical processes. *Annals of Probability*, 22, 1994: 28–76.
- [30] L. G. Valiant. A theory of the learnable. *Communications of the ACM*, 27(11), 1984: 1134–1142.
- [31] V. N. Vapnik: *Statistical Learning Theory*, Wiley, 1998.
- [32] V. N. Vapnik. *Estimation of Dependences Based on Empirical Data*. Springer-Verlag, New York, 1982.
- [33] V.N. Vapnik and A.Y. Chervonenkis. On the uniform convergence of relative frequencies of events to their probabilities. *Theory of Probability and its Applications*, 16(2), 1971: 264–280.
- [34] Y. A. Zuev. Asymptotics of the logarithm of the number of threshold functions of the algebra of logic. *Soviet Mathematics Doklady*, 39, 1989: 512–513.