Links Between Learning and Optimization: a Brief Tutorial

Martin Anthony
Department of Mathematics
and Centre for Discrete and Applicable Mathematics
The London School of Economics and Political Science
London WC2A 2AE, UK
m.anthony@lse.ac.uk

CDAM Research Report LSE-CDAM-2003-02
January 2003

Abstract

This report is a brief exposition of some of the important links between machine learning and combinatorial optimization. We explain how efficient 'learnability' in standard probabilistic models of learning is linked to the existence of efficient randomized algorithms for certain natural combinatorial optimization problems, and we discuss the complexity of some of these optimization problems.

1 Introduction

In this report, we aim to give a brief exposition of some of the important links between machine learning and combinatorial optimization. We explain how efficient 'learnability' in standard probabilistic models of learning is linked to the existence of efficient randomized algorithms for certain natural optimization problems, and we discuss the complexity of some of these problems. There is no attempt here to give an all-encompassing survey of work on the complexity of learning and its associated optimization problems; rather, the goal is to provide a brief self-contained tutorial for those unfamiliar with computational learning theory, highlighting the aspects that may be of interest to researchers in optimization or operations research.

2 Probabilistic modeling of learning

The main probabilistic model of 'supervised' learning we discuss here is a variant of the 'probably approximately correct' (or PAC) model introduced by Valiant [20], and further developed by a number of many others; see [21, 9, 1], for example. The probabilistic aspects of the model have their roots in work of Vapnik and Chervonenkis [22, 23], as was pointed out in [4]. Computational efficiency was a key aspect of Valiant's learning model [20], and has been much further explored for the models of this chapter. The papers [4, 17] provided some of the important initial results, and these are further explored in the books [12, 15, 2]. The treatment here follows [1].

In the model, it is assumed that, for each n, we use some class H_n of functions defined on $X_n \subseteq \mathbb{R}^n$ to find a good fit to a set of data, where the data points are of the form (x, b) for $x \in X_n$, and $b \in \{0, 1\}$. The union $H = \bigcup H_n$ is called the *hypothesis class*. 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 $t \in H_n$ for some n (where $x \in X_n$). (In this case, t is called 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* $\mathbf{s} =$

$((x_1, t(x_1)), \ldots, (x_m, t(x_m))$ as follows: for some n, each x_i is chosen independently according to some fixed probability distribution μ on X_n. 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 $h \in H_n$ 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, labeling of the data points. In this more general formulation, it is assumed that, for some n, the data points (x, b) in the training sample are generated according to a probability distribution P on the product $X_n \times \{0, 1\}$. 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_n that is a good predictor of the data labels (something we will shortly make precise).

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_n^*, where $Z_n = X_n \times \{0, 1\}$ and $Z_n^* = \bigcup_{m=1}^{\infty} Z_n^m$. We may therefore regard a learning algorithm as a function

$$L : \bigcup_{n=1}^{\infty} Z_n^* \to H = \bigcup_{n=1}^{\infty} H_n$$

with the property that if $\mathbf{s} \in Z_n^*$ then $L(\mathbf{s}) \in H_n$. 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 $Z_n = X_n \times \{0, 1\}$, for some n, we may define the *error*, $\mathrm{er}_P(h)$, of a function $h \in H_n$ (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, $\mathrm{er}_P(h) = P(\{(x, b) \in Z_n : h(x) \neq b\})$.

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_n^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 $\mathrm{opt}_P(H_n) = \min\{\mathrm{er}_P(h) : h \in H_n\}$. We are led to the following definition of a PAC learning algorithm. (See [20, 9, 13], for example.)

Definition 2.1 *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*

$$\mathrm{er}_P(L(\mathbf{s})) < \mathrm{opt}_P(H_n) + \epsilon.$$

The smallest suitable value of $m_0(n, \delta, \epsilon)$, denoted $m_L(n, \delta, \epsilon)$, is called the *sample complexity* of L. The definition is fairly easy to understand in the realizable case. In this case, $\mathrm{er}_P(h)$ is the probability that a hypothesis h disagrees with the target concept t on a randomly chosen example from the appropriate domain X_n. 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 $m_0(n, \delta, \epsilon)$ be independent of P.

Additionally, we want the learning to be not only accurate enough in the sense just indicated, but also *efficient*. An input to L is a training sample, which consists of m labeled vectors of length n. 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. We 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 $1/\epsilon$ and $\ln(1/\delta)$. If these conditions hold, then the running time required to produce a 'good' output hypothesis will be polynomial in n, $\ln(1/\delta)$ and $1/\epsilon$. Thus we have the following definition of an *efficient PAC learning algorithm*.

Definition 2.2 *Let $H = \bigcup H_n$ be a hypothesis class and suppose that L is a PAC 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 (for instance, a Boolean formula, or a set of weights for a neural network). When discussing the complexity of learning, it is always assumed that the output lies in a representation class for the hypothesis class. This is not something we shall explore much further, for the sake of simplicity, but it is sometimes important.

3 Sample complexity of learning

We now describe some results concerning the sample complexity of learning. We will present these without proof, since our main interest, in subsequent sections, lies in what they tell us about the computational efficiency of learning. We first need a few important definitions.

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

$$\hat{\mathrm{er}}_{\mathbf{s}}(h) = \frac{1}{m} \left| \{i : h(x_i) \neq x_i\} \right|.$$

A natural optimization problem in this context is, given $\mathbf{s}$, to seek to find $h \in H$ that has minimum sample error. We say that L is a SEM (sample-error minimization) algorithm for $H = \bigcup H_n$ if, for any n and any $\mathbf{s} \in Z_n^*$,

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

(So, such an algorithm solves the optimization problem.)

3.1 Finite H_n

The following result shows that if each H_n is finite, then any SEM algorithm for H is a PAC learning algorithm.

Theorem 3.1 *Suppose that $H = \bigcup H_n$ where each H_n is finite. Then any SEM learning algorithm L for H is PAC and has sample complexity bounded as follows:*

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

Note that, in the realizable case, the optimal error is zero, so a SEM algorithm is what is called a *consistent* algorithm. This is one which, given a sample labeled by a target concept $t \in H$, returns h that is consistent with the sample, meaning that $h(x_i) = t(x_i)$ for each i, where t is the target concept. We have the following [4].

Theorem 3.2 *Suppose that $H = \bigcup H_n$ where each H_n is finite. Then, for realizable learning problems, any consistent learning algorithm L is PAC and has sample complexity bounded as follows: $m_L(n, \delta, \epsilon) \leq (1/\epsilon) \ln (|H_n|/\delta)$.*

3.2 The VC dimension

An important measure of the *expressive power* of a set of functions H from X to $\{0, 1\}$ is the *Vapnik-Chervonenkis dimension* [23], or *VC-dimension*, of H, which is defined as follows.

Definition 3.3 *Let H be a set of 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 $x \in E \setminus S$.*

Thus, the VC-dimension is the largest cardinality of a set S of points in X on which all possible $2^{|S|}$ classifications can be achieved. The VC-dimension of a finite set of functions can easily be bounded in terms of its cardinality: if H is finite then $\mathrm{VCdim}(H) \leq \log_2 |H|$.

The importance of the VC-dimension in learning theory was highlighted in [4]. The following result bounds from above the sample complexity of PAC learning (in the general and realizable cases). It is obtained from results of Vapnik and Chervonenkis [23] (see [1]) and Blumer *et al.* [4].

Theorem 3.4 *Suppose that $H = \bigcup H_n$ is a hypothesis class, suppose H_n has VC-dimension $d_n \geq 1$, and let L be any SEM algorithm for H. Then L is a PAC learning algorithm for H with sample complexity bounded as follows:*

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

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:

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

In fact, it is possible (using a result of Talagrand [19]; see [1]) to obtain an upper bound of order $(1/\epsilon^2)\left(d + \ln\left(1/\delta\right)\right)$. for the general case. (However, the constants involved are quite large.)

The following lower bounds on sample complexity are also obtainable; see [7, 18, 1] for these and similar results.

Theorem 3.5 *Suppose that H is a hypothesis class with $|H_n| \geq 3$ for all n, and $\mathrm{VCdim}(H_n) = d_n \geq 1$. For any PAC learning algorithm L for H, the sample complexity $m_L(n, \delta, \epsilon)$ of L satisfies*

$$m_L(n, \delta, \epsilon) \geq \frac{1}{\epsilon^2}\left(\frac{d_n}{640} + \frac{1}{4}\ln\left(\frac{1}{8\delta}\right)\right)$$

for all $0 < \epsilon, \delta < 1/64$. For any PAC learning algorithm L for H in the realizable case, the sample complexity $m_L(n, \delta, \epsilon)$ of L satisfies

$$m_L(n, \delta, \epsilon) \geq \frac{1}{\epsilon}\left(\frac{(d_n - 1)}{32} + \frac{1}{2}\ln\left(\frac{1}{\delta}\right)\right)$$

for all $0 < \epsilon < 1/8$ and $0 < \delta < 1/100$.

4 Sufficient conditions for efficient learning

There may be some advantage in allowing SEM algorithms and PAC learning 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. We may therefore speak of the 'probability' that $\mathcal{A}$ has a given outcome on an (external) input x. It is useful to extend our concept of a PAC learning algorithm to allow randomization. The definition of a randomized PAC learning is as in Definition 2.1, with the additional feature that the algorithm is randomized. (So, L should no longer be regarded as a deterministic function, and the $1-\delta$ probability is a probability jointly over randomly chosen samples and over the random bitstream employed in the randomization of the algorithm.) An *efficient* randomized PAC algorithm is then defined in the obvious manner. We shall also be interested in efficient randomized SEM algorithms.

Definition 4.1 *A randomized algorithm $\mathcal{A}$ is an* efficient randomized SEM algorithm *for the hypothesis 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 $\hat{\mathrm{er}}_{\mathbf{s}}(h) = \min_{g \in H_n} \hat{\mathrm{er}}_{\mathbf{s}}(g)$.*

Suppose we run a randomized SEM algorithm k times on a fixed sample, 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 $(1/2)^k$. The following result, which may be found in [4] (for the realizable and deterministic case), follows directly from the observation just made (by taking the best of k iterations of $\mathcal{A}$ for a suitable k) and from the results presented in the previous section. (See [1] for details.) It shows that if $\mathrm{VCdim}(H_n)$ is small enough, then the existence of an efficient randomized SEM algorithm implies the existence of an efficient randomized PAC learning algorithm.

Theorem 4.2 *Suppose that $H = \bigcup H_n$ is a hypothesis class and that $\mathrm{VCdim}(H_n)$ 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.*

5 Necessary conditions for efficient learning

We have seen that efficient SEM algorithms 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 randomized PAC learning algorithm for a hypothesis class then *necessarily* there is an efficient randomized SEM algorithm. (For the realizable case, this may be found in [17, 4, 16].)

Theorem 5.1 *If there is an efficient randomized PAC learning algorithm for the hypothesis class $H = \bigcup H_n$, then there is an efficient randomized SEM algorithm.*

Proof: Suppose L is an efficient PAC learning algorithm for the hypothesis 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 randomized PAC learning algorithm, and given that $m^* = m_L(n, 1/2, 1/m)$, with probability at least $1/2$, we have $\mathrm{er}_P(h^*) < \mathrm{opt}_P(H) + 1/m$. But because P is discrete, with no probability mass less than $1/m$, this means $\mathrm{er}_P(h^*) = \mathrm{opt}_P(H)$. For any h, by the definition of P, $\mathrm{er}_P(h) = \hat{\mathrm{er}}_{\mathbf{s}}(h)$. So with probability at least $1/2$,

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

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. □

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

Theorem 5.2 *Suppose that $H = \bigcup H_n$ is a hypothesis class. Then there is an efficient randomized PAC learning algorithm for H if and only if $\mathrm{VCdim}(H_n)$ is polynomial in n* and *there is an efficient randomized SEM algorithm for H.*

6 Establishing hardness of learning

There are two quite natural decision problems associated with a hypothesis 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 $\hat{\mathrm{er}}_{\mathbf{s}}(h) \leq k/m$?

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 $\hat{\mathrm{er}}_{\mathbf{s}}(h) = 0$?

H-FIT is clearly related to the optimization problem of finding a function in H with minimal error, with this optimization problem being at least as difficult as H-FIT. 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.

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 following results explain why this approach can be taken. First, we have the following [13, 10].

Theorem 6.1 *Let $H = \bigcup H_n$ be a hypothesis class. If there is an efficient randomized 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 5.1, 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 $1/2$, $\hat{\mathrm{er}}_{\mathbf{s}}(h)$ is minimal among all functions in H_n. The next step in $\mathcal{B}$ is to check whether $\hat{\mathrm{er}}_{\mathbf{s}}(h) \leq k/m$. 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 $\hat{\mathrm{er}}_{\mathbf{s}}(h)$ is linear in the size of $\mathbf{s}$, $\mathcal{B}$ is a polynomial-time algorithm. □

The following result [17] applies to the realizable case.

Theorem 6.2 *Suppose that* $H = \bigcup H_n$ *is a hypothesis 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 6.3 *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.*

7 Hardness results

We now use the theory just developed to show that PAC learnability of Boolean 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.

7.1 Threshold functions

A function t defined on $\{0,1\}^n$ is a *(Boolean) threshold function* if there are $w \in \mathbb{R}^n$ and $\theta \in \mathbb{R}$ such that

$$t(x) = \begin{cases} 1 & \text{if } \langle w, x \rangle \geq \theta \\ 0 & \text{if } \langle w, x \rangle < \theta, \end{cases}$$

where $\langle w, x \rangle = w^T x$ is the standard inner product of w and x. The vector w is known as the *weight-vector*, and θ is known as the *threshold*. We denote the class of threshold functions on $\{0,1\}^n$ by T_n.

It is well-known that if T_n is the set of threshold Boolean functions on $\{0,1\}^n$, then the hypothesis 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 [4, 1], for instance.) However, T is *not* efficiently PAC learnable in the general case, if RP $\neq$ NP. This arises from the following result [8, 11, 10].

Theorem 7.1 *Let $T = \bigcup T_n$ be the hypothesis class of threshold functions. Then T-*FIT *is* NP*-hard.*

We present a proof of this which establishes 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 $1, 2, \ldots, n$. Then, a typical edge $\{i, j\}$ will, for convenience, be denotes 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 [8].

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 $V = \{1, 2, \ldots, n\}$ 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 $i = 1, 2, \ldots, n$ 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 $\mathbf{s}$ is $(2r + n)(2n + 1)$, which is polynomial in the size of the original instance of VERTEX COVER, and that $z(G)$ can be computed in polynomial time.

Lemma 7.2 *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 $\hat{\mathrm{er}}_{\mathbf{s}}(h) \leq k/(2n)$ 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 $w = (w_1, w_2, \ldots, w_{2n})$ 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

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

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 $k/(2n)$ on $\mathbf{s}(G)$. Suppose U is a vertex

cover and $|U| \leq k$. Define a weight-vector $w = (w_1, w_2, \ldots, w_{2n})$ and threshold θ as follows: let $\theta = 1$ and, for $i = 1, 2, \ldots, n$,

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

We claim that if h is the threshold function represented by w and θ, then $\hat{\mathrm{er}}_{\mathbf{s}}(h) \leq k/(2n)$. 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

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

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.

7.2 k-clause CNF

Pitt and Valiant [17] were the first to give an example of a hypothesis class H for which the consistency problem H-CONSISTENCY is NP-hard, as we now describe.

Any Boolean function (that is, any function from $\{0,1\}^n$ to $\{0,1\}$ can be expressed by a *conjunctive normal formula* (or CNF), using *literals* $x_1, x_2, \ldots, x_n, \bar{x}_1, \ldots, \bar{x}_n$, where the $\bar{x}_i$ are known as *negated literals*. A conjunctive normal formula is one of the form

$$C_1 \wedge C_2 \wedge \cdots \wedge C_k,$$

where each C_l is a *clause* of the form

$$C_l = \left(\bigvee_{i \in P} x_i \right) \bigvee \left(\bigvee_{j \in N} \bar{x}_j \right),$$

for some disjoint subsets P, N of $\{1, 2, \ldots, n\}$. A Boolean function is said to be a k-clause-CNF if there is such a formula representing the function in which the number of clauses C_i is at most k.

Let C_n^k be the set of k-clause CNF functions. Pitt and Valiant [17] showed that, for fixed $k \geq 2$, the consistency problem for $C^k = \bigcup C_n^k$ is NP-hard. Thus, if NP$\neq$ RP, then can be no efficient PAC learning algorithm for C^k in the realizable case. We prove this here for the case $k \geq 3$.

The reduction in this case is to GRAPH K-COLORABILITY. Suppose we are given a graph $G = (V, E)$, with $V = \{1, 2, \ldots, n\}$. 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 ith 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 7.3 *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

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

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 $\{1, 2, \ldots, k\}$ 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 ith 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 $\chi : V \to \{1, 2, \ldots, k\}$. For $1 \leq f \leq k$, define h_f to be the clause $\bigvee_{\chi(i) \neq f} x_i$, and define $h = h_1 \wedge h_2 \wedge \ldots \wedge h_k$. 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. □

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 [17].

8 Conclusions

This report has only briefly explored some of the connections between machine learning and optimization, but has attempted to show that such connections are natural and central. But of course there has been much more work on the complexity of learning and the associated optimization problems. Another approach to proving the difficulty of learning has been via reducing learning to problems that are assumed to be hard on the basis of standard cryptographic hardness assumptions. (See [14], for example). Additionally, the optimization problems of finding a hypothesis with smallest sample error have been shown, for many hypothesis classes, to be difficult even to approximate to; see, for example, [10, 3, 5] and the references therein.

References

[1] M. Anthony and P. L. Bartlett. *Neural Network Learning: Theoretical Foundations*. Cambridge University Press, 1999.

[2] Martin Anthony and Norman L. Biggs. *Computational Learning Theory: An Introduction*. Cambridge Tracts in Theoretical Computer Science, 30, 1992. Cambridge University Press, Cambridge, UK.

[3] P.L. Bartlett and S. Ben-David. Hardness results for neural network approximation problems. In *Proceedings of the 4th European Conference on Computational Learning Theory* (ed. P. Fischer and H. Simon), Springer, 1999.

[4] 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.

[5] N. H. Bshouty and L. Burroughs. Maximizing agreements and CoAgnostic learning. In *Algorithmic Learning Theory, 13th International Conference, ALT 2002, Lübeck, Germany, November 2002*, (ed. N. Cesa-Bianchi, M. Numao and R. Reischuk). Springer Lecture Notes in Artificial Intelligence LNAI 2533. Springer, 2002.

[6] L. Devroye and G. Lugosi. Lower bounds in pattern recognition and learning. *Pattern Recognition* 28(7), 1995: 1011–1018.

[7] 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.

[8] M. Garey and D. Johnson. *Computers and Intractibility: A Guide to the Theory of NP-Completeness*. Freemans, San Francisco, 1979.

[9] D. Haussler. Decision theoretic generalizations of the PAC model for neural net and other learning applications. *Information and Computation*, 100(1), 1992: 78–150.

[10] 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.

[11] D. S. Johnson and F. P. Preparata. The densest hemisphere problem. *Theoretical Computer Science*, 6, 1978: 93–107.

[12] M. J. Kearns. *The Computational Complexity of Machine Learning*. ACM Distinguished Dissertation Series. The MIT Press, Cambridge, MA., 1989.

[13] M. J. Kearns, R. E. Schapire and L. M. Sellie. Toward efficient agnostic learning. *Machine Learning* 17(2/3), 1994: 115–142.

[14] M. Kearns and L.G. Valiant. Cryptographic limitations on learning Boolean formulae and finite automata. In *Proceedings of the 21st Annual ACM Symposium on the Theory of Computing*. The Association for Computing Machinery, New York.

[15] M. J. Kearns and U. Vazirani. *Introduction to Computational Learning Theory*, MIT Press, Cambridge, MA, 1995.

[16] B. K. Natarajan. On learning sets and functions. *Machine Learning*, 4(1), 1989: 67–97.

[17] L. Pitt and L. Valiant. Computational limitations on learning from examples. *Journal of the ACM*, 35, 1988: 965–984.

[18] 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.

[19] M. Talagrand. Sharper bounds for Gaussian and empirical processes. *Annals of Probability*, 22, 1994: 28–76.

[20] L. G. Valiant. A theory of the learnable. *Communications of the ACM*, 27(11), 1984: 1134–1142.

[21] V. N. Vapnik: *Statistical Learning Theory*, Wiley, 1998.

[22] V. N. Vapnik. *Estimation of Dependences Based on Empirical Data*. Springer-Verlag, New York, 1982.

[23] 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.