Computational Learning Theory - Hilary Term 2018
1 : Introduction to the PAC Learning Framework

Lecturer: Varun Kanade

1 What is computational learning theory?

Machine learning techniques lie at the heart of many technological applications that we use daily. When using a digital camera, the boxes that appear around faces are produced using a machine learning algorithm. When websites such as BBC iplayer or Netflix suggest what a user might like to watch next, they are using machine learning algorithms to provide these recommendations.

In the field of computational learning theory, we develop precise mathematical formulations of 'learning from data'. Having a precise mathematical formulation allows us to answer questions such as the following: What types of functions are easy to learn? Are there classes of functions that are hard to learn? How much data is required to learn a function of a particular type? What amount of computational resources are required to learn?

We will be interested in positive as well as negative answers to these questions. For example, we will consider learning algorithms that are guaranteed to learn certain kinds of functions using modest amount of data and reasonable runnning time. For the most part, we will take the view that as long as the resources used can be bounded by a polynomial function of the problem size, the learning algorithm is efficient. Obviously, as in the case of analysis of algorithms, there may be situations where just being polynomial time may not be considered efficient enough. Some of the algorithms we study will not run in polynomial time at all, but they will still be much better than brute force algorithms.

2 A Rectangle Learning Game

Let us consider the following rectangle learning game. We are given some points in the Euclidean plane, some of which are labeled positive and others negative. Furthermore, we are guaranteed that there is an axis-aligned rectangle such that all the points inside it are labelled positive, while those outside are labelled negative. However, this rectangle itself is not revealed to us. Our goal is to produce a rectangle that is "close" to the true hidden rectangle that was used to label the observed data. (See Fig. 1(a).)

To make the problem a bit more concrete, suppose that the two dimensions measure the curvature and length of bananas. The points that are labelled positive have *medium* curvature and *medium* length and represent the bananas that would pass the stringent EU regulation. However, the actual lower and upper limits that "define" *medium* in each dimension are hidden from us. We wish to learn some rectangle that will be good enough to predict whether bananas would pass the regulators' tests or not.

Let R be the unknown rectangle used to label the points. We can express the labelling process using a boolean function $c_R : \mathbb{R}^2 \to \{+, -\}$, where $c_R(x) = +$ if x is inside the rectangle R and $c_R(x) = -$ otherwise. Let us consider the following simple approach. We consider the *tightest* possible rectangle that can fit all the positively labelled data inside it; let us denote this rectangle by R' (Fig. 1(b)). Our prediction function or *hypothesis* $h_{R'} : \mathbb{R}^2 \to \{+, -\}$ is the following: if $x \in R'$, $h_{R'}(x) = +$, else $h_{R'}(x) = -$.[1] Let us consider the following questions:

- Have we learnt the function c_R ?

[1] For the sake of concreteness, let us say that points on the sides are considered to be inside the rectangle.

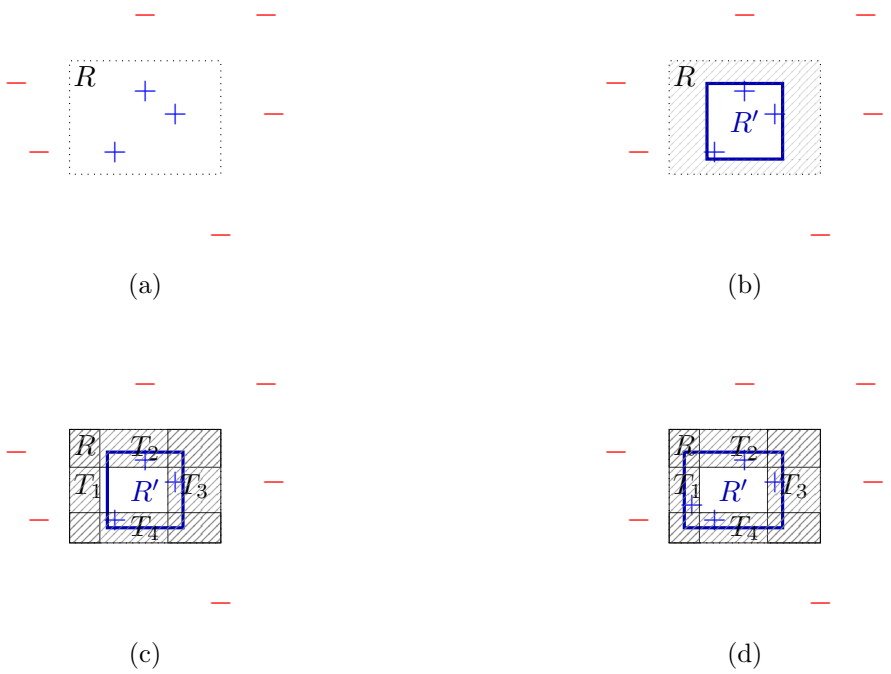


Figure 1: (a) Data received for the rectangle learning game. The rectangle R used to generate labels is not known to the learning algorithm. (b) The *tightest fit* algorithm produces a rectanlge R'. (c) & (d) The regions T_1, T_2, T_3 and T_4 contain $\epsilon/4$ mass each under D.

- How good is our prediction function $h_{R'}$?

Let R denote the *true* rectangle that actually defines the labelling function c_R. Since, we've chosen the tightest possible fit, the rectangle R' must be entirely contained inside R. Consider the shaded region shown in Fig. 1(b). For any point x that is in this shaded region, it must be that $h_{R'}(x) = -$, while $c_R(x) = +$. In other words, our prediction function $h_{R'}$ would make errors on all of these points. This brings us to the important point, if we had to make predictions on points that mostly lie in this region our hypothesis would be quite bad. This brings us to an important point that the data that is used to learn a hypothesis should be similar to the data on which we will be tested. Let us formalise this notion.

Let D be a distribution over $\mathbb{R}^2$. Our training data consists of m points that are drawn independently according to D and then labelled according to the function c_R. We will define the error of a hypothesis $h_{R'}$ with respect to the target function c_R and distribution D as follows:

$$\text{err}(h_{R'}; c_R, D) = \mathbb{P}_{x \sim D}\left[h_{R'}(x) \neq c_R(x)\right]$$

Whenever the target c_R and distribution D are clear from context, we will simply refer to this as $\text{err}(h_{R'})$.

Let us show that in fact our algorithm outputs an $h_{R'}$ that is quite good, in the sense that given any $\epsilon > 0$ as the target error, with high probability it will output $h_{R'}$ such that $\text{err}(h_{R'}; c_R, D) \leq \epsilon$. Consider four rectangular strips T_1, T_2, T_3, T_4 that are chosen along the sides of the rectangle R (and lying inside) such that they each have a probability mass under of exactly $\epsilon/4$ under D.[2] Note that some of these strips overlap, *e.g.,* T_1 and T_2 (See Fig. 1(c)). The total probability mass of the set $T_1 \cup T_2 \cup T_3 \cup T_4$ is at most ϵ. Now, if we can guarantee that the training data of m points contains at least one point from each of T_1, T_2, T_3 and T_4, then the tightest fit rectangle R' will be such that $R \setminus R' \subseteq T_1 \cup T_2 \cup T_3 \cup T_4$ and as a consequence,

[2]Assuming the distribution D is smooth over $\mathbb{R}^2$, *i.e.,* there is no point mass under D, this is always possible. Otherwise, the algorithm is still correct, however, the analysis is slightly more tedious.

$\text{err}(h_{R'}; c_R, D) \leq \epsilon$. This is shown in Fig. 1(d); note that if even one of the T_i do not contain any data point, this may be cause a problem, in the sense that the region of disagreement between R and R' may have probability mass greater than ϵ (See Fig. 1(c)).

Let A_1 be the event that when m points are drawn independently according to D, none of them lies in T_1. Similarly define the events A_2, A_3, A_4 for T_2, T_3, T_4. Let $\mathcal{E} = A_1 \cup A_2 \cup A_3 \cup A_4$. If $\mathcal{E}$ does not occur, then $\text{err}(h_{R'}; c_R, D) \leq \epsilon$. We will use the union bound to bound $\mathbb{P}[\mathcal{E}]$. The union bound states that for any two events A and B,

$$\mathbb{P}[A \cup B] \leq \mathbb{P}[A] + \mathbb{P}[B].$$

Let us compute $\mathbb{P}[A_1]$. The probability that a single point drawn from D does not land in T_1 is $1 - \epsilon/4$, so the probability that after m independent draws from D none of the points are in T_1 is $\left(1 - \frac{\epsilon}{4}\right)^m$. By a similar argument, $\mathbb{P}[A_i] = \left(1 - \frac{\epsilon}{4}\right)^m$ for $i = 1, \ldots, 4$. Thus, we have

$$\begin{aligned}
\mathbb{P}[\mathcal{E}] &\leq \sum_{i=1}^{4} \mathbb{P}[A_i] && \text{By the Union Bound} \\
&= 4\left(1 - \frac{\epsilon}{4}\right)^m \\
&\leq 4\exp\left(-\frac{m\epsilon}{4}\right) && \text{As } 1 - x \leq e^{-x}
\end{aligned}$$

Now let $\delta > 0$ be fixed, then if $m \geq \frac{4}{\epsilon}\log\left(\frac{4}{\delta}\right)$, then we have that $\mathbb{P}[\mathcal{E}] \leq \delta$. In other words, with probability at least $1 - \delta$, $\text{err}(h_{R'}; c_R, D) \leq \epsilon$.

Remarks: A couple of remarks are in order. We can think of ϵ as being an accuracy parameter and δ being the confidence parameter. The bound of $m \geq \frac{4}{\epsilon}\log\left(\frac{4}{\delta}\right)$ shows that as we demand higher accuracy and higher confidence of our learning algorithm, we need to supply more data. This is indeed a reasonable requirement. Furthermore, the cost for getting higher accuracy and higher confidence is relatively modest. For example, if we want to halve the error, say go from $\epsilon = 0.02$ to $\epsilon = 0.01$, the amount of data required (as per the bound) at most doubles.[3]

3 Probably Approximately Correct (PAC) Learning

Let us start defining a precise mathematical framework for learning using the insights gained from the rectangle learning game. First let us make a few observations:

1. The learning algorithm does not know the target concept to be learnt (obviously, otherwise there is nothing to learn!). However, the learning algorithm does know the set of possible target concepts. In this instance, the unknown target is always an axis-aligned rectangle.

2. The learning algorithm has access to data drawn from some distribution D. We do assume that the observations are drawn independently according to D. However, no assumption is made on the distribution D itself.

3. The output hypothesis is evaluated with respect to the same distribution D as was used to obtain the training data.

[3]We are using the world "required" a bit losely here. All we can say is our present analysis of this particular algorithm suggests that the amount of data required scales linearly as $\frac{1}{\epsilon}$. We will see lower bounds of this nature that hold for any algorithm later in the course.

4. We would like our algorithms to be statistically efficient (require relatively small number of examples to guarantee high accuracy and confidence) as well as computationally efficient (reasonable amount of time required for processing examples and producing an output hypothesis).

Let us now formalise a few other concepts related to learning.

Instance Space

Let X denote the set of possible instances or examples that may be seen by the learning algorithm. For instance, in the rectangle learning game the examples were points in $\mathbb{R}^2$. When classifying images the examples may be 3 dimensional arrays, containing the RGB values of each pixel.

Concept Class

A *concept* c over X is a boolean function $c : X \to \{0, 1\}$.[4] A concept class C over X is a collection of concepts c over X. In the rectangle learning game, the concept class under consideration is all axis-aligned rectangles. In order to view the concepts as boolean functions, we may use the convention that $+$ corresponds to 1 and $-$ corresponds to 0. The learning algorithm has knowledge of C, but not of the specific concept $c \in C$ that is used to label the observations.

Data Generation

Let D be a fixed probability distribution over X. The training data is obtained as follows. An example $x \in X$ is drawn according to the distribution D. If c is the target concept, the example x is labelled accordingly as $c(x)$. The learning algorithm observes $(x, c(x))$. We will refer to this process as an example oracle, $\mathsf{EX}(c, D)$. We assume that a learning algorithm can query the oracle $\mathsf{EX}(c, D)$ at unit cost.

3.1 PAC Learning : Take I

Let $h : X \to \{0, 1\}$ be some hypothesis. Once the distribution D over X and $c \in C$ are fixed, the error of h with respect to c and D is defined as:

$$\mathrm{err}(h; c, D) = \mathbb{P}_{x \sim D}\left[h(x) \neq c(x)\right]$$

When c and D are clear from context, we will simply refer to this as $\mathrm{err}(h)$.

Definition 1 (PAC Learning : Take I)**.** *Let C be a concept class over X. We say that C is PAC learnable if there exists a learning algorithm L that satisfies the following: for every concept $c \in C$, for every distribution D over X, for every $0 < \epsilon < 1/2$ and $0 < \delta < 1/2$, if L is given access to $\mathsf{EX}(c, D)$ and inputs ϵ and δ, L outputs a hypothesis $h \in C$ that with probability at least $1 - \delta$ satisfies $\mathrm{err}(h) \leq \epsilon$. The probability is over the random examples drawn from $\mathsf{EX}(c, D)$ as well as any internal randomisation of L. We further say that C is* efficiently *PAC learnable if the running time of L is polynomial in $1/\epsilon$ and $1/\delta$.*

The term PAC stands for probably approximately correct. The approximately correct part captures the notion that the output hypothesis does have some error; demanding higher accuracy (lower ϵ) is possible, but comes at a cost of increased running time and sample complexity. The probably part captures the notion that there is some chance that the algorithm may fail completely. This may happen because the observations are not representative of the distribution,

[4] We will consider concepts that are not boolean functions later in the course.

(a) (b)

Figure 2: Different shape concepts in $\mathbb{R}^2$.

a low probability event, though very much a possible event. Our confidence (lower δ) in the correctness of our algorithm is increased as we allow more sample complexity and running time.

In Section 2, we essentially proved the following theorem.

Theorem 2. *The concept class C of axis aligned rectangles in $\mathbb{R}^2$ is efficiently PAC (Take I) learnable.*

3.2 PAC Learning : Take II

Let us now discuss a couple of questions that we have glossed over so far. The question is that of the complexity of the concepts that we are trying to learn. For example, consider the question of learning rectangles (Fig. 2(a)) vs the shapes as shown in Fig. 2(b). Intuitively, we believe that it should be harder to learn concepts defined by shapes as shown in Fig. 2(b) than rectangles. Thus, an algorithm that learns a more complex class should be allowed more resources (samples, running time, memory, *etc.*). For example, to represent a rectangle we only need to store four real numbers, the lower and upper limits in both the x and y directions. The number of real numbers used to represent more complex shapes is higher.[5]

The question of representation is better elucidated by taking the case of boolean functions. Suppose that the instance space X is $\{0,1\}^n$, the set of length n bit vectors. We can consider boolean functions $f : X \to \{0,1\}$. There are several ways of representing boolean functions. One option is to keep the entire truth table with 2^n entries. Alternatively, we may represent f as a circuit using $\wedge$, $\vee$ and $\neg$ gates. We may ask that f be represented in disjunctive normal form (DNF), *i.e.,* of the form shown below

$$(x_1 \wedge \bar{x}_3 \wedge x_7 \wedge \cdots) \vee (x_2 \wedge x_4 \wedge \bar{x}_8 \wedge \cdots) \vee \cdots \vee (x_1 \wedge x_3)$$

The choice of representation is quite important as shown by the following exercise.

Exercise: Let $f = x_1 \oplus x_2 \oplus \cdots \oplus x_n$ be the parity function on n bits. Show that f can be represented as a circuit of size $O(n \log n)$ using $\wedge$, $\vee$ and $\neg$ gates. Show that f represented in disjunctive normal form consists of $\Omega(2^n)$ terms.

There are other possible representations of boolean functions, such as decision trees or neural networks.

Representation Scheme

Abstractly, a representation scheme for a concept class C is a function $R : \Sigma^* \to C$, where Σ is a finite alphabet.[6] Any σ satisfying $R(\sigma) = c$ is called a representation of c. We assume that there is a function, size : $\Sigma^* \to \mathbb{N}$, that measures the size of a representation. A concept

[5] We assume that our computers can store and manipulate real numbers at unit cost.

[6] If the concept requires using real numbers, such as in the case of rectangles, we may use $R : (\Sigma \cup \mathbb{R})^* \to C$.

$c \in C$ may in general have multiple representations under R. For example, there are several boolean circuits that compute exactly the same boolean function. We extend size to the set C by defining, $\text{size}(c) = \min_{\sigma: R(\sigma)=c} \{\text{size}(\sigma)\}$. When we refer to a concept class, we will assume by default that it is associated with a representation scheme and a size function, so that $\text{size}(c)$ is well defined for $c \in C$.

Instance Size

Typically in a machine learning problems instances have a size; roughly we may think of the size of an instance as the memory required to store an instance. For example, 10×10 black and white images can be defined using 100 bits, whereas 256×256 colour images will require over 2 million real numbers. Thus, when faced with larger instances it is natural to allow learning algorithms more time. In this course, we will only consider settings when the instance space is either $X_n = \{0,1\}^n$ or $X_n = \mathbb{R}^n$. We denote by C_n a concept class over X_n. We consider the instance space $X = \bigcup_{n \geq 1} X_n$ and the concept class $C = \bigcup_{n \geq 1} C_n$.

Definition 3 (PAC Learning : Take II)**.** *Let C_n be a concept class over X_n and let $C = \bigcup_{n \geq 1} C_n$ and $X = \bigcup_{n \geq 1} X_n$. We say that C is PAC learnable (take II) if there exists an algorithm L that satisfies the following: for all $n \in \mathbb{N}$, for every $c \in C_n$, for every D over X_n, for every $0 < \epsilon < 1/2$ and $0 < \delta < 1/2$, if L is given access to $\mathsf{EX}(c, D)$ and inputs ϵ, δ and $\text{size}(c)$, L outputs $h \in C_n$ that with probability at least $1 - \delta$ satisfies $\text{err}(h) \leq \epsilon$. We say that C is efficiently PAC learnable if the running time of L is polynomial in n, $\text{size}(c)$, $1/\epsilon$ and $1/\delta$, when learning $c \in C_n$.*

4 Learning Conjunctions

Let us now consider a second learning problem. Let $X_n = \{0,1\}^n$ and C_n denote the set of conjunctions over X_n. Let $X = \bigcup_{n \geq 1} X_n$ and $\mathsf{CONJUNCTIONS} = \bigcup_{n \geq 1} C_n$. A conjunction can be represented by a set of literals, where each literal corresponds to either a variable or its negation. An example of a conjunction is $x_1 \wedge \bar{x}_3 \wedge x_4$. Sometimes a conjunction is referred to as a *term*. We can represent a conjunction by using a bit string of length at most $2n$ as there are only $2n$ possible literals for n boolean variables. Thus, our goal is to design an algorithm that runs in time polynomial in n, $1/\epsilon$ and $1/\delta$.

The example oracle $\mathsf{EX}(c, D)$ returns examples of the form (a, y) where $y \in \{0,1\}$. The i^{th} bit a_i is the assignment of the variable x_i. The label y is 1 is the target conjunction c evaluates to true under the assignment a and 0 otherwise.

We consider the following simple algorithm:

(i) Start with a candidate hypothesis that includes all $2n$ literals, *i.e.,*

$$h \equiv x_1 \wedge \bar{x}_1 \wedge x_2 \wedge \bar{x}_2 \wedge \cdots \wedge x_n \wedge \bar{x}_n$$

(ii) For each positive example, $(a, 1)$, and for $i = 1, \ldots, n$, do the following:

  (a) If $a_i = 1$, drop the literal $\bar{x}_i$ from h if it still exists,

  (b) Else drop the literal x_i if it still exists.

(iii) Output the resulting h

Let us make a couple of observations about the algorithm.

1. The algorithm only uses positive examples; the negative examples are completely ignored.

2. If c is the target conjunction. Any literal ℓ that is present in c is also present in the hypothesis h. This is because the algorithm only throws out literals that cannot possibly be present in c, as the literal evaluated to negative in the example $(a, 1)$ but the label of the example was positive. This also ensure that if $c(a) = 0$ then $h(a) = 0$. Thus, h if it errs at all only does so on examples a, such that $c(a) = 1$.

Theorem 4. *The class of boolean conjunctions,* CONJUNCTIONS*, is efficiently PAC (take II) learnable.*

Proof. Let c be the target conjunction and D the distribution over $\{0,1\}^n$. For a literal ℓ, let $p(\ell) = \mathbb{P}_{a\sim D}\left[c(a) = 1 \wedge \ell \text{ is 0 in } a\right]$; here ℓ is 0 in a, means that the assignment of the variables according to a causes the literal ℓ to evaluate to 0. Notice that if $p(\ell) > 0$, then the literal ℓ cannot be present in c; if it were, then there can be no a such that $c(a) = 1$ and ℓ is 0 in a.

We define a literal ℓ to be bad if $p(z) \geq \frac{\epsilon}{2n}$. We wish to ensure that all bad literals are eliminated from the hypothesis h. For a bad literal ℓ, let A_ℓ denote the event that after m independent draws from $\mathsf{EX}(c, D)$, ℓ is not eliminiated from h. Note that this can only happen if no a such that $c(a) = 1$ but ℓ is 0 in a is drawn. This can happen with probability at most $\left(1 - \frac{\epsilon}{2n}\right)^m$. Let B denote the set of bad literals and let $\mathcal{E} = \bigcup_{\ell\in B} A_\ell$ be the event that at least one bad literal survives in h. We shall choose m large enough so that $\mathbb{P}[\mathcal{E}] \leq \delta$. Consider the following:

$$\begin{aligned}
\mathbb{P}[\mathcal{E}] &\leq \sum_{\ell\in B} \mathbb{P}[A_\ell] && \text{By the Union Bound} \\
&\leq 2n\left(1 - \frac{\epsilon}{2n}\right)^m && |B| \leq 2n \text{ and for each } \ell \in B, \mathbb{P}[A_\ell] \leq \left(1 - \frac{\epsilon}{2n}\right)^m \\
&\leq 2n \exp\left(-\frac{m\epsilon}{2n}\right) && \text{As } 1 - x \leq e^{-x}
\end{aligned}$$

Thus, whenever $m \geq \frac{2n}{\epsilon}\log\left(\frac{2n}{\delta}\right)$, we know that $\mathbb{P}[\mathcal{E}] \leq \delta$. Now, suppose that $\mathcal{E}$ does not occur, *i.e.,* all bad literals are eliminated from h. Let G be the set of good literals.

$$\begin{aligned}
\text{err}(h) &= \mathbb{P}_{a\sim D}\left[c(a) = 1 \wedge h(a) = 0\right] \\
&\leq \sum_{\ell\in G} \mathbb{P}_{a\sim D}\left[c(a) = 1 \wedge \ell \text{ is 0 in } a\right] \\
&\leq 2n \cdot \frac{\epsilon}{2n} \leq \epsilon
\end{aligned}$$

This finishes the proof. □

5 Hardness of Learning 3-term DNF

Let us now look at a richer class of boolean functions that simply conjunctions. The class we'll consider is the class of 3-term DNF formulae. What is a 3-term DNF formula? It is simply a disjunction of exactly three terms, where each term is simply a boolean conjunction. Define the class,

$$\text{3-TERM-DNF} = \{T_1 \vee T_2 \vee T_3 \mid T_i \text{ conjunction over } x_1, \ldots, x_n\}$$

Note that any function that can be expressed as a 3-term DNF formula has representation size at most $6n$—there are three terms each of which is a boolean conjunction expressible by a boolean string of length $2n$. Thus, an efficient algorithm for learning 3-TERM-DNF would have to run in time polynomial in n, $1/\epsilon$ and $1/\delta$. Our next result shows that such an algorithm in fact is unlikely to exist. Formally, we'll proof the following theorem.

Theorem 5. 3-TERM-DNF *is not efficiently PAC learnable (take II) unless* RP = NP.

Let us first discuss the condition "unless RP = NP". Let us briefly describe what the class RP is. For further details, the reader is referred to a book on complexity theory such as that by Arora and Barak (2009).

The class RP consists of languages for which membership can be determined by a randomised polynomial time algorithm that errs on only one side. More precisely, a language $L \in \mathsf{RP}$, if there exists a randomised polynomial time algorithm A that satisfies the following:

- For $\pi \notin L$, $A(x) = 0$
- For $\pi \in L$, $A(x) = 1$ with probability at least $1/2$.

In order to prove Theorem 5, we shall reduce an NP-complete language to the problem of PAC (take II) learning 3-TERM-DNF. Suppose L is a language that is NP-complete. Given an instance π we wish to decide whether $\pi \in L$. We will construct a sample, a set of positive and negative examples, with the following property: There exists a 3-term DNF formula φ that is consistent with all the examples in the sample if and only if $\pi \in L$.

Let us see how an algorithm that can PAC-learn 3-TERM-DNF helps us test whether or not $\pi \in L$. Let S denote the sample constructed and let D be a distribution that is uniform over the sample, *i.e.,* a distribution that assigns probability mass $\frac{1}{|S|}$ for every example that appears in the sample and 0 mass on all other examples. Let $\epsilon = \frac{1}{2|S|}$ and $\delta = 1/2$. Now, let us suppose that $\pi \in L$, then indeed there does exist 3-term DNF φ that is consistent with the sample S. So we can simulate a valid example oracle $\mathsf{EX}(\varphi, D)$, by simply returning a random example from S along with its label. Now, with probability at least $1/2$, the algorithm returns $h \in$ 3-TERM-DNF, such that $\mathrm{err}(h) \leq \frac{1}{2|S|}$. However, as there are only $|S|$ points in S and the distribution is uniform, it must be that h is consistent with all examples in S, which implies $\pi \in L$.

On the other hand, if $\pi \notin L$, there is no 3-term DNF formula that is consistent with the sample produced. Hence, the learning algorithm cannot output such an h. Thus, assuming a PAC (take II) learning algorithm for 3-TERM-DNF exists, we have a randomised algorithm to solve the decision problem for the NP-complete language L. This in turn implies that NP = RP, something that is widely believed to be untrue.

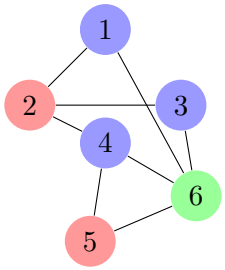
5.1 Reducing Graph 3-Colouring to PAC (Take II) Learning 3-term DNF

Let us now actually choose a particular NP-complete language and show that it can be reduced to the learning problem for 3-TERM-DNF. The language 3-COLOURABLE consists of representations of graphs that can be 3-*coloured*. We say a graph is 3-*colourable* if there is an assignment from the vertices to the set $\{r, g, b\}$ such that no two adjacent vertices are assigned the same colour. As already discussed, given a graph G, we only need to produce a sample of positively and negatively labelled points such that the graph G is 3-colourable if and only if there exists a 3-term DNF consistent with all the labelled points.

Let $S = S_+ \cup S_-$ where S_+ consists of points that have label 1 and S_- of points that have label 0. Suppose G has n vertices. For vertex $i \in G$, we let $v(i) \in \{0,1\}^n$ that has a 1 in every position except i. For an edge (i, j) in G, we let $e(i, j) \in \{0,1\}^n$ that has a 1 in all positions except i and j. Let $S_+ = \{v(i) \mid i \text{ a node of } G\}$ and $S_- = \{e(i,j) \mid (i,j) \text{ an edge of } G\}$. Figure 3 shows an example of a graph that is 3-colourable along with the sample S.

G 3-colourable implies a consistent 3-term DNF exists

Suppose G is three colourable. Let V_r, V_g, V_b be the set of vertices of G that are labelled red, blue and green respectively. Let $T_r = \bigwedge_{i \notin V_r} x_i$. T_g and T_b are defined similarly. Now $\varphi = T_r \vee T_g \vee T_b$



(a) Graph

v(1)	(0, 1, 1, 1, 1, 1)
v(2)	(1, 0, 1, 1, 1, 1)
v(3)	(1, 1, 0, 1, 1, 1)
v(4)	(1, 1, 1, 0, 1, 1)
v(5)	(1, 1, 1, 1, 0, 1)
v(6)	(1, 1, 1, 1, 1, 0)

(b) Positive Examples

e(1, 2)	(0, 0, 1, 1, 1, 1)
e(1, 6)	(0, 1, 1, 1, 1, 0)
e(2, 3)	(1, 0, 0, 1, 1, 1)
e(2, 4)	(1, 0, 1, 0, 1, 1)
e(3, 6)	(1, 1, 0, 1, 1, 0)
e(4, 5)	(1, 1, 1, 0, 0, 1)
e(4, 6)	(1, 1, 1, 0, 1, 0)
e(5, 6)	(1, 1, 1, 1, 0, 0)

(c) Negative Examples

Figure 3: (a) A graph G along with a valid three colouring. (b) Positive examples of the sample generated using G. (c) Negative examples of the sample generated using G.

is a 3-term DNF formula. We will show that φ is consistent with S. To show this we need to show that all examples in S_+ satisfy φ and none of the examples in S_- satisfy φ. First consider $v(i) \in S_+$. Without loss of generality, suppose i is coloured red, *i.e.,* $i \in V_r$. Then, we claim that $v(i)$ is a satisfying assignment of T_r and hence also of φ. Clearly, the literal x_i is not contained in T_r and there are no negative literals in T_r. Since all the bits of $v(i)$ other than the i^{th} position are 1, $v(i)$ is a satisfying assignment of T_r.

Now, consider $e(i, j)$. We will show that $e(i, j)$ is not a satisfying assignment of any of T_r, T_g or T_b and hence it also does not satisfy φ. For a colour $c \in \{r, g, b\}$, either i is not coloured c or j isn't. Suppose i is the one that is not coloured c, then T_c contains the literal x_i, but the i^{th} bit of $e(i, j)$ is 0 and so $e(i, j)$ is not a satisfying assignment of T_c. This argument applies to all colours and hence $e(i, j)$ is not a satisfying assignment of φ. This shows that φ is consistent with S.

Existence of 3-term DNF implies G 3-colourable

Suppose $\varphi = T_r \vee T_g \vee T_b$ is a 3-term DNF that is consistent with S. We use this to assign colours to the nodes of G. For a node i, since $v(i)$ is a satisfying assignment of φ, it is also a satisfying assignment of at least one of T_r, T_g or T_b. We assign a colour to it based on the term for which it is a satisfying assignment, and ties may be broken arbitrarily. Since, for every vertex i, there exists $v(i) \in S_+$, this ensures that every vertex is assigned a colour.

Now, we need to ensure that no two adjacent vertices are assigned the same colour. Suppose there is an edge (i, j) such that i and j are assigned the same colour. Without loss of generality, suppose that this colour is red. Now, we know that since φ is consistent with S, $e(i, j)$ does not satisfy φ and hence also does not satisfy T_r. Also, as i and j were both coloured red, $v(i)$ and $v(j)$ do satisfy T_r. This implies that the literals x_i and x_j are not present in T_r. The fact that $v(i)$ satisfies T_r ensures that $\bar{x}_k$ for any $k \neq i$ cannot be a literal in T_r. However, if T_r does not contain any negated literal other than possibly x_i and if it does not contain the literals x_i and x_j, then $e(i, j)$ satisfies T_r and hence φ, a contradiction. Hence, there cannot be any two adjacent vertices that have been assigned the same colour.

This completes the proof of Theorem 5.

6 Learning 3-CNF

Let us revisit the problem of PAC-learning 3-term DNF formulae. Let us recall the the distributive law of boolean operations.

$$(a \wedge b) \vee (c \wedge d) \equiv (a \vee c) \wedge (a \vee d) \wedge (b \vee c) \wedge (b \vee d)$$

Using the distributive rule, we can re-write a 3-term DNF formula as follows:

$$T_1 \vee T_2 \vee T_3 \equiv \bigwedge_{\substack{\ell_1 \in T_1 \\ \ell_2 \in T_2 \\ \ell_3 \in T_3}} (\ell_1 \vee \ell_2 \vee \ell_3) \tag{1}$$

The form on the right hand side is called a 3-CNF formula. CNF stands for conjunctive normal form, *i.e.,* expression as a conjunction of clauses (disjunctions). The 3 indicates that each clause has length 3 (*i.e.,* it contains 3 literals). Note that any 3-term DNF formula can be expressed as a 3-CNF formula with at most $(2n)^3$ clauses, as each of the terms in a 3-term DNF formula can have at most $2n$ literals. The converse however is not true, *i.e.,* there are 3-CNF formulae that cannot be represented as a 3-term DNF formula. Thus, 3-CNF formulae is a more general class of functions.

Let us now consider the question of learning 3-CNF formulae. In doing so, we'll study an important notion in computational learning theory, as in all of computer science, that of a reduction. Note that a 3-CNF formula can be viewed simply as a conjunction over a set of clauses. Thus, if we can create a new *instance space* where the variables are actually encodings of all the $(2n)^3$ possible clauses, the 3-CNF formula simply becomes a conjunction. For any three literals, ℓ_1, ℓ_2, ℓ_3, we create a variable z_{ℓ_1,ℓ_2,ℓ_3} that takes the value $\ell_1 \vee \ell_2 \vee \ell_3$.

More precisely consider the following maps: (i) $f : \{0,1\}^n \to \{0,1\}^{(2n)^3}$ which is a one-one map that takes an assignment of n boolean variables and maps it to an assignment of all possible clauses of length 3 over the n boolean variables, (ii) $g : \text{3-CNF}[n] \to \text{CONJUNCTIONS}[(2n)^3]$ which maps a 3-CNF formula over n variables to a conjunction over $(2n)^3$ variables. We have already seen that we can PAC learn the class of conjunctions. We'd like to use this algorithm to learn the class 3-CNF.

Let c be the target 3-CNF formula and D a distribution over $\{0,1\}^n$. Let $c' = g(c)$ and let D' be a distribution over $\{0,1\}^{(2n)^3}$, where $D'(S) = D(f^{-1}(S))$. We need to be able to simulate the example oracle $\mathsf{EX}(c', D')$. Doing so is simple, we receive $(x, c(x))$ form $\mathsf{EX}(c, D)$, we output $(f(x), c(x))$; this is a valid simulation of $\mathsf{EX}(c', D')$. The learning algorithm returns a conjunction of (positive or negative) literals over $(2n)^3$ variables. We need to transform this back to a 3-CNF formula over n variables. The first thing to note is that we only need to learn the class of conjunctions with positive literals, so we can modify the conjunction learning algorithm to start with a conjunction that includes all the positive literals (instead of all the positive and negative literals); the rest of the algorithm remains the same. Then, if a literal corresponding to z_{ℓ_1,ℓ_2,ℓ_3} is included in the output conjunction, we include the clause $(x_{\ell_1} \vee x_{\ell_2} \vee x_{\ell_3})$ in the 3-CNF formula.

Theorem 6. *The class of* 3-CNF *formulae is efficiently PAC-learnable.*

7 PAC Learning

Let us consider the results in Sections 5 and 6 and reconsider the goal of learning. On the one hand, we saw that is impossible to learn the class of 3-term DNF formulae unless NP = RP. The hardness result crucially relies on the fact that the output of the algorithm was itself required to be a 3-term DNF formula. On the other hand, we know that any 3-term DNF formula can be expressed as a 3-CNF formula (with a modest, but still polynomial, blow-up in size). The class of 3-CNF formulae turns out to be PAC-learnable. This means that when receiving examples labelled according to a 3-term DNF formula, if we had in fact applied the 3-CNF learning algorithm, we would have produced a hypothesis with low error. The output hypothesis however would be a 3-CNF formula not a 3-term DNF formula. If the aim of machine learning is prediction, it should not matter what form we represent the hypothesis in. In fact, we may not even care that the output is a 3-CNF formula, it could potentially be something

even more complex, as long as it makes correct predictions. Thus, in the final definition of PAC learning, we'll remove the condition that the output hypothesis actually belongs to the concept class being learnt. In general, we'll allow learning algorithms to output hypotheses that lie in some class H, called the *hypothesis* class. While we allow the hypothesis class H to be different from the concept class C and possibly a lot more powerful, for the notion of *efficiency* we do require that any hypothesis $h \in H$ is *polynomial time evaluatable*. As with the case of concept classes, we assume that there is a representation scheme and a suitable *size* function for the hypothesis class H. Formally,

Definition 7. *A hypothesis class H is* polynomially evaluatable *if there exists an algorithm that on input any instance $x \in X_n$ and any representation $h \in H_n$, outputs the value $h(x)$ in time polynomial in n and* $\text{size}(h)$.

We can now give the final defintion of PAC learning.

Definition 8 (PAC Learning)**.** *Let C_n be a concept class over X_n and let $C = \bigcup_{n\geq 1} C_n$ and $X = \bigcup_{n\geq 1} X_n$. We say that C is* PAC learnable using hypothesis class H *if there exists an algorithm L that satisfies the following: for all $n \in \mathbb{N}$, for every $c \in C_n$, for every D over X_n, for every $0 < \epsilon < 1/2$ and $0 < \delta < 1/2$, if L is given access to $\mathsf{EX}(c, D)$ and inputs ϵ, δ and* $\text{size}(c)$*, L outputs $h \in H$ that with probability at least $1 - \delta$ satisfies* $\text{err}(h) \leq \epsilon$*. We say that C is efficiently PAC learnable if the running time of L is polynomial in n,* $\text{size}(c)$*, $1/\epsilon$ and $1/\delta$, when learning $c \in C_n$ and if H is polynomially evaluatable.*

Remark 9. *It is worth noting that the requirement that H is polynomially evaluatable is quite necessary and natural. Essentially what we are asking is that the designer of a learning algorithm, when returning the "code" to make predictions, actually returns code that can be evaluated by the user.*

Bibliographic Notes

Material in this lecture is almost entirely adopted from (Kearns and Vazirani, 1994, Chap. 1). The original PAC learning framework was introduced in a seminal paper by Valiant (1984).

References

Sanjeev Arora and Boaz Barak. *Computational Complexity: A Modern Approach*. Cambridge University Press, 2009.

Michael J. Kearns and Umesh K. Vazirani. *An Introduction to Computational Learning Theory*. The MIT Press, 1994.

Leslie Valiant. A theory of the learnable. *Communications of the ACM*, 27(11):1134–1142, 1984.