Introduction to Complexity Theory*
Lecture 25: Computational Learning Theory

Guest Lecturer: Dana Ron
Notes taken by Oded Lachish and Eli Porat

July 31, 1999

Summary: We define a model of learning known as probably approximately correct (PAC) learning. We define efficient PAC learning, and present several efficient PAC learning algorithms. We prove the Occam's Razor Theorem, which reduces the PAC learning problem to the problem of finding a succinct representation for the values of a large number of given labeled examples.

1 Towards a definition of Computational learning

Learning is a notion we are familiar with from every day life. When embarking on the task of importing this notion into computer science, the first natural step is to open a dictionary and find an exact meaning for it. A natural meaning that can be found is "gaining knowledge through experience". With this meaning in mind, we set on the task of defining a formal computer science model for learning. In order to get a clue to what the model should look like, it is worthwhile to examine a real life setting.

Learning to diagnose a new disease. A medical doctor learns the symptoms for diagnosing a new disease by drawing a number of files, from a file archive. Each file contains a list of the patient's parameters such as weight, body temperature, age, etc., and a *label* indicating the diagnosis; that is, whether the person has this specific disease. The number of examples the doctor has drawn depends on how accurate he wanted his list of symptoms to be. Using these files he concludes a list of symptoms for diagnosing the new disease. In order to check the accuracy of symptoms he concluded, he draws a labeled file, uses the list of symptoms to reach a diagnosis, and finally whether the diagnosis obtained by him matches the true diagnosis provided in the file (i.e., the label of the file).

Using this setting we can phrase the process of learning as follows: In order to learn an unknown subject (disease) from a family of subjects (family of diseases) the learner (doctor) receives a measure of accuracy, with this measure in mind he draws a number of labeled examples (files) from the set of all possible examples (archive of files), labeled with respect to subject. Using the examples he reaches a rule (list of symptoms) for labeling examples. He checks his rule accuracy, by drawing an example, and comparing its label to the label computed by the rule for this example.

But something is still lacking in the above model. We demonstate this by using the above setting (of a doctor learning to diagnose a new disease): Can a doctor learn a new disease that

* Lecture Notes for a course given by Oded Goldreich at the Weizmann Institute of Science, Israel.

occurs in a certain group of people, when he does not see a file of a person from this specific group? Of course we cannot hope for this to happen. Thus, the missing component in the above model is that the doctor's ability to diagnose will be tested against the same distribution of examples (files) based on which he has learned to diagnose. Thus, if in the learning stage the doctor didn't draw any file of a person from a specific group, then chances are that he won't be asked to diagnose such a person later (i.e., the the accuracy of symptoms he concluded will not be checked for such a person). We stress that we do not require the learner (doctor) to have any knowledge regarding the distribution of examples.

The components of the learning process: We call the object that conducts the learning process, the *learner*. In this lecture the learner is an algorithm. The objects on which the learning is done are called *instances*. The specifics of the instances do not interest us. We are only interested in an abstract representation of the parameters characterizing it. We represent an instance by a vector defined as follows: a value is assigned to each parameter of the instance, this vector is the representation of these values.

- $X = \bigcup_{n \geq 1} X_n$ Instance space – the set of all possible values of instance representation.
- $D : X_n \longrightarrow [0,1]$ Underlying distribution – the (unknown) probability distribution over example space.

where n is the number of parameters characterizing an instance. In this lecture the instance space will be either $\{0,1\}^n$ or R^n. The learning is done with respect to the distribution D, from which we obtain independent samples. We denote by $x \sim D$ an example x drawn from X_n according to distribution D.

The subject and aim of our learning is a *labeling* of the instance space, where by labeling we mean a correlation between the instance space and the a set values. In this lecture we use the set $\{0,1\}$ (or any set isomorphic to it) as possible labeling values. Each instance is labeled by such a value, and so the labeling (of all instances) is a function from X to $\{0,1\}$. Such a labeling is called a *concept*. We use the following notations

- $F = \bigcup_{n \geq 1} F_n$ concept class – family of concepts we may need to learn. This family is a subset of the set of all Boolean functions defined over X.
- $f \in F_n$ target concept – the concept which is the subject of the learning.

where n appears in notations since a concept is a function over the instance space X_n.

The rule that is the output of the learner is also a labeling of the instance space, therefore it is also a Boolean function. We call this function the *hypothesis*.

The final component missing is a measure of accuracy of the hypothesis. We actually consider the complementary measure; that is the error of the hypothesis. The latter is merely the probability that for an instance drawn according to D, which is the very distribution used in the learning process, the hypothesis agrees with the target concept. That is,

Definition 1 (hypothesis error): *$err_{f,D}(h)$ is the probability that the hypothesis h disagrees with the target function f on an instance x drawn from X_n according to D. That is,*

$$err_{f,D}(h) \stackrel{\text{def}}{=} \Pr_{x \sim D}[h(x) \neq f(x)]$$

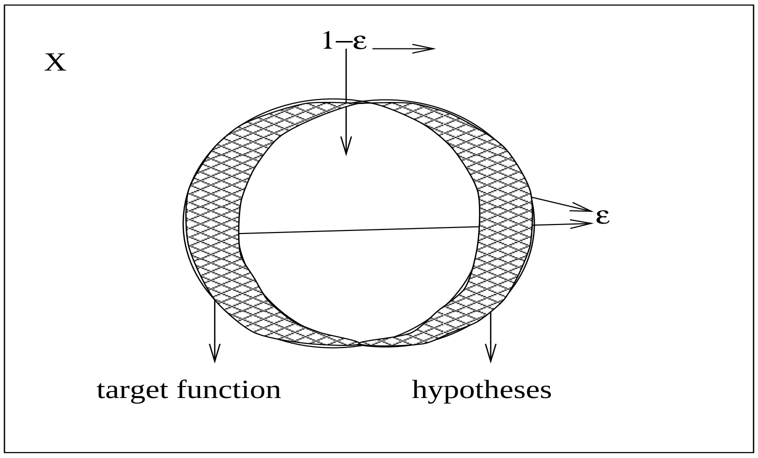

Figure 1: The target function and hypothesis with their difference shaded

2 Probably Approximately Correct (PAC) Learning

In this model the algorithm is only given partial information on the target concept, that is, a "small" number of examples drawn independently by a given distribution. Since the examples are chosen independently by a distribution, it could be the case that the algorithm was given the same example all the time. Therefore it is unconceivable to expect the algorithm to give an hypothesis h which is fully equivalent to the target function f. A realistic view to the problem would be to expect with a given probability that the algorithm will supply an hypothesis h that is an approximation of the target function f with respect to the underlying distribution. The following definitions capture these qualities:

Definition 2 (PAC learning algorithm): *An algorithm A is called a PAC* learning algorithm for a concept class *F if the following holds: For every n, for every function $f \in F_n$, for every distribution $D : X_n \longrightarrow [0,1]$, for every error parameter $0 < \epsilon < 1$ and for every confidence parameter $0 < \delta < 1$, given parameters n, ϵ, δ and a set of f-labeled examples, $\{< x^i, f(x^i) >\}$, where x^i is drawn independently under the distribution D, the algorithm outputs a hypothesis h such that:*

$$\Pr[err_{f,D}(h) \leq \epsilon] \geq 1 - \delta$$

where the probability is taken over the choice of examples, as well as over the internal coin tosses of the algorithm. Note $h = A(n, \epsilon, \delta, \{< x^i, f(x^i) >\})$.

Once we established a formal model, a natural step would be to inquire upon following questions:

1. What is an efficient PAC learning algorithm?

2. How many examples do we need for PAC learning a concept from a concept class?

In order to deal with these questions we define complexity measures for the PAC learning model. A natural complexity measure is the number of examples needed:

Definition 3 (sample complexity): *The* sample complexity *of a PAC algorithm is the number of examples it utilizes, as a function of n, ϵ and δ.*

The classical measure of running time of an algorithm is also applicable for this model. The running time complexity is necessarily larger then the sample complexity. In some cases it appears to be substaintially larger.

> Oded's Note: *That is, under reasonable complexity assumptions (e.g., existence of one-way functions), there are concept classes that can be PAC learned using* $\mathrm{poly}(n, \frac{1}{\epsilon}, \log \frac{1}{\delta})$ *examples, but cannot be PAC learned in* $\mathrm{poly}(n, \frac{1}{\epsilon}, \frac{1}{\delta})$*-time. (In analogy to other contexts, we should expect complexities to be logarithmic in* $1/\delta$*.)*

Using these complexity measures we can answer the first question. We say that a PAC learning algorithm is efficient if it runs in time polynomial in n, $\frac{1}{\epsilon}$, $\frac{1}{\delta}$, and $size(f)$, where the size of $f \in F_n$ is usually polynomial in n. In the rest of this lecture, we focus on the second question (i.e., consider only the sample complexity of PAC algorithms).

Learning axis-aligned rectangles over $[0,1]^2$. We wish to design an efficient PAC learning algorithm for axis-aligned rectangles over $[0,1]^2$.

Let us first explicitly cast the problem according to the PAC learning model.

- An instance is a point in $[0,1]^2$, we represent it by its x,y axis coordinates, that is $< x, y >$.
- The instance space is $X = \{\text{all } < x, y > \text{ representations of points in } [0,1]^2\}$.
- A concept is an axis-aligned rectangle in $[0,1]^2$, represented by four points $< x_{min}, y_{min}, x_{min}, y_{max} >$ such that:

  $$x_{min}, x_{max}, y_{min}, y_{max} \in [0,1]$$

  $$x_{min} \leq x_{max} \text{ and } y_{min} \leq y_{max}$$

  An instance $< x, y >$ is in the axis aligned rectangle f represented by $< x_{min}, y_{min}, x_{min}, y_{max} >$ if:

  $$x_{min} \leq x \leq x_{max} \text{ and } y_{min} \leq y \leq y_{max}$$

  If $< x, y >$ is in this f, we label it by a "+" (i.e., $f(< x, y >) = +$), otherwise we label it by a "-".
- The concept class is $F = \{\text{all axis-aligned rectangle in } [0,1]^2\}$. The target concept is an axis aligned rectangle $f \in F$.
- Finally, as usual, the underlying distribution is denoted D, the error parameter is $0 < \epsilon < 1$ and the confidence parameter is $0 < \delta < 1$.

For the sake of clarity we will refer to the target concept by target rectangle, and to axis-aligned rectangles as rectangles. We use the following notations:

- $W_D(g)$ is the weight assigned by distribution D to the rectangle g. That is,

  $$W_D(g) \stackrel{\text{def}}{=} \Pr_{s \sim D}[g(s) = +]$$

- S is a random variable representing the set of example points. Each element in S is drawn according to D. We let S^+ denote the subset of S labeled "+".

We stress that all probabilities are taken over the choice of examples.

Claim 2.1 *The following algorithm is an efficient PAC learning algorithm for axis-aligned rectangles over* $[0,1]^2$*.*

If S^+ *is empty then output the empty rectangle as hypothesis.*
Otherwise, output the rectangle represented by $< \tilde{x}_{min}, \tilde{y}_{min}, \tilde{x}_{max}, \tilde{y}_{max} >$*, where*

$\tilde{x}_{min} =$ *the minimal x-axis coordinate in* S^+*.*
$\tilde{y}_{min} =$ *the minimal y-axis coordinate in* S^+*.*
$\tilde{x}_{max} =$ *the maximal x-axis coordinate in* S^+*.*
$\tilde{y}_{max} =$ *the maximal y-axis coordinate in* S^+*.*

Proof: Let f be the target rectangle and g be the hypothesis output by the algorith. Note that the hypothesis rectangle is always contained in the target rectangle (since the borders of the former are determined by positive examples).

We partition the proof into two cases:

1. $W_D(f) > \epsilon$
2. $W_D(f) \leq \epsilon$

We start with the proof of the first case. Given a target rectangle let us draw the following auxiliary construction: We cover the upper side of the target rectangle with a line, we push this line towards the opposite side, until we get a rectangle A_1, such that $W_D(A_1) = \frac{\epsilon}{4}$.

> Oded's Note: *We assume that an adequate stopping point exists; that is, that the distribution is such that some rectangle has weight smaller than* $\epsilon/4$ *whereas a slightly bigger rectangle has weight* $\epsilon/4$*. Clearly, an approximation of this assumption is good enough, but even such an approximation is not guaranteed to exist. This issue is dealt with in an appendex.*

We repeat this process for all other sides of the residual target rectangle (see Fig. 2). We get rectangle A_2,A_3,A_4 such that $W_D(A_2) = W_D(A_3) = W_D(A_4) = \frac{\epsilon}{4}$. (It should be stressed, that we assumed that this process can be done, which is not necessarily the case. We will deal with this problem in appendix.) Let us look at the part of the f, that is not covered by A_1,A_2,A_3 and A_4, it is a rectangle we call it B. According to auxiliary construction:

$$\begin{aligned} W_D(f) - W_D(B) &= W_D(A_1) + W_D(A_2) + W_D(A_3) + W_D(A_4) \\ &= 4 \cdot \frac{\epsilon}{4} = \epsilon \end{aligned}$$

If hypothesis rectangle h contains B (i.e., $h \supseteq B$) then (using $h \subseteq f$)

$$\begin{aligned} err_{f,D}(h) &= \Pr_{s\sim D}[s \in f \text{ and } s \notin h] \\ &\leq \Pr_{s\sim D}[s \in f \text{ and } s \notin B] \\ &= W_D(f) - W_D(B) = \epsilon \end{aligned}$$

Furthermore, according to algorithm, if there is an example in each of the rectangle A_1, A_2, A_3 and A_4, then $B \subseteq h$. Thus, we merely bound the number of examples, denoted m, such that the probability for this event not to occur, is less then δ. For any $i \in \{1,2,3,4\}$, the probability that no example resides in A_i is $(1 - W_D(A_i))^m = (1 - \frac{\epsilon}{4})^m$. Thus,

$$\begin{aligned} \Pr[err_{f,D}(h) > \epsilon] &\leq \Pr[\exists i \text{ no sample in } A_i] \\ &\leq 4 \cdot \left(1 - \frac{\epsilon}{4}\right)^m \end{aligned}$$

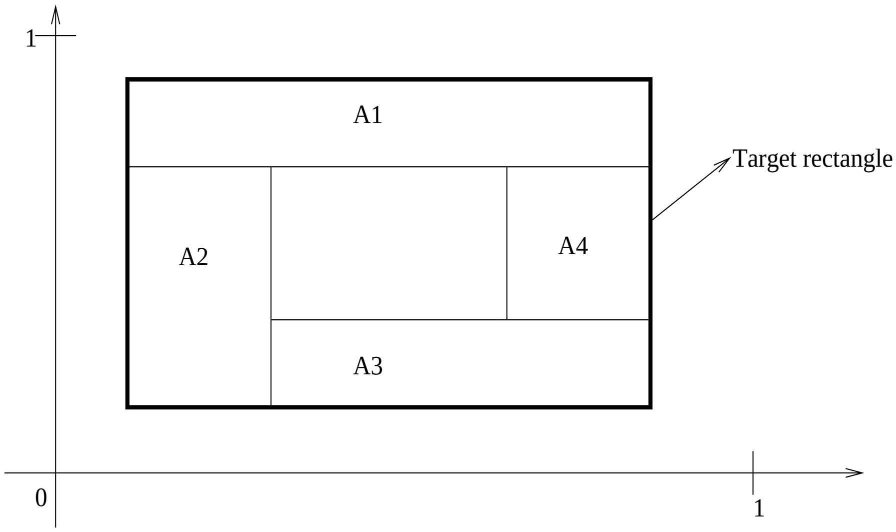

Figure 2: Target rectangle in bold, and rectangle A_1, A_2, A_3, A_4 each of weight $\frac{\epsilon}{4}$

So we need to set m so that $4 \cdot (1-(\epsilon/4))^m \leq \delta$. Using the inequality $1-\frac{\epsilon}{4} \leq e^{-\frac{\epsilon}{4}}$, the condition simplifies to $4(e^{-\frac{\epsilon}{4}})^m \leq \delta$, which solves to $m \geq (4/\epsilon) \cdot \ln(4/\delta)$.

We now turn to the second case, where $W_D(f) \leq \epsilon$: Using $h \subseteq f$, we have (for every sequence of examples used by the algorithm):

$$\begin{aligned} err_{f,D}(h) &= \Pr_{s\sim D}[s \in f \text{ and } s \notin h] \\ &\leq \Pr_{s\sim D}[s \in f] \;=\; W_D(f) \;\leq\; \epsilon \end{aligned}$$

Thus, in this case, $\Pr[err_{f,D}(h) \leq \epsilon] = 1$.

Note all operations in algorithm depend linearly on the size of the sample, which in turn depends linearly on $\frac{1}{\epsilon}$ and $ln\frac{4}{\delta}$. Therefore the algorithm is an efficient PAC learning algorithm. ■

3 Occam's Razor

Occam's Razor is based on a principle stated by William of Occam. We interpret Occam's principle as follows: learning can be achieved by finding a succinct representation for the labels of large number of examples. By a succinct representation, we mean that the size of the representation is sublinear in the number of examples. This reduces the problem of learning to the problem of finding a hypothesis, consistent with every given example. We say an algorithm is an *Occam's Algorithm*, if it outputs a succinct hypothesis consistent with every given example.

Theorem 4 (Occam's Razor – basic version): *Let F_n be a finite concept class. Let A be an algorithm such that for every n, for every $f \in F_n$ and for every number of examples labeled by f, the algorithm outputs an hypothesis $h \in F_n$, that is consistent with all the given examples. Then for any distribution D, for any error parameter $0 < \epsilon < 1$ and for any confidence parameter $0 < \delta < 1$ if the number of examples drawn independently by D is larger or equal $\frac{1}{\epsilon}(\log\frac{1}{\delta} + \log|F_n|)$, then*

$$\Pr[err_{f,D}(h) \leq \epsilon] \geq 1-\delta$$

where h is the hypothesis output by the algorithm, and the probability is taken over the choice of examples as well as over the internal coin tosses of the algorithm.

Proof: We use the following notations:

- $S =< a_j, f(a_j) >_{j=1...m}$ is the set of labeled examples.
- $C_F(S)$ is the set of hypotheses in F that are consistent on all examples in S. That is,

$$C_F(S) \stackrel{\text{def}}{=} \{h \in F : h(a_i) = f(a_i) \text{for } i = 1, ..., m \ \}$$

- $B_{D,\epsilon}(f)$ is the set of hypotheses in F that have a probability of error larger than ϵ. That is,

$$B_{D,\epsilon}(f) \stackrel{\text{def}}{=} \{h \in F : err_{f,D}(h) > \epsilon\}$$

Note that the algorithm always outputs an hypothesis $h \in C_F(S)$. We will upper bound the probability that this hypothesis is in $B_{D,\epsilon}(f)$. Actually, we will upper bound the probability that any hypothesis in $C_F(S)$ remains in $B_{D,\epsilon}(f)$.

$$\Pr_S[\text{there exists } h \in B_{D,\epsilon}(f) \text{ such that } h \in C_F(S)] \leq \sum_{h \in B_{D,\epsilon}(f)} \Pr_S[h \in C_F(S)] \tag{1}$$

Proposition 3.1 *For any $h \in B_{D,\epsilon}(f)$,* $\Pr_S[h \in C_F(S)] \leq \frac{\delta}{|F|}$

Proof: For any hypothesis in $B_{D,\epsilon}$ the probability that the hypothesis will be in $C_F(S)$ is the probability that all given example points landed in the agreement area. By using definition of $B_{D,\epsilon}(f)$ the probability that a single example point landed in the agreement area is at most $1 - \epsilon$. Therefore for any fixed $h \in B_{D,\epsilon}(f)$:

$$\Pr_S[h \in C_F(S)] \leq (1 - \epsilon)^{|S|} \leq e^{-\epsilon|S|}$$

Using the assumption on $|S|$:

$$\epsilon|S| \geq \log(\tfrac{1}{\delta}|F|)$$

Thus,

$$e^{-\epsilon|S|} \leq \frac{\delta}{|F|}$$

and the proposition follows. ■

Combining Eq. (1) and Proposition 3.1, the theorem follows. ■

It is easy to observe, that an algorithm satisfying this theorem, is a PAC learning algorithm. Also note that since the hypothesis is taken from the concept class it is necessarily succinct.

Learning monomials. The concept class F_n of monomials is a family of boolean expressions, over literals corresponding to n variables $x_1, x_2, ..., x_n$, defined as follows:

$$F_n = \{f : f = l_1 \wedge l_2 \wedge ... \wedge l_t\}$$

Where for each $1 \leq i \leq t$

$$l_i \in \{x_j, \overline{x_j}\}_{j=1}^{n}$$

We seek an efficient PAC learning algorithm for the concept class of monomials. A instance $a \in X_n$ is interpreted as an assignment to the variables $x_1, x_2, ..., x_n$. We us the notation a_i for the value of the i'th bit of a.

We denote the set of given examples by:

$$S = \{< a^j, f(a^j) >\}_{j=1,...,m}$$

where each $a^j = a_1^j, ..., a_n^j$.

We call an example *negative* if it is labeled by 0, otherwise we call it *positive*. Before stating the algorithm, let us see what information can we conclude about the target concept, from a given example. A positive example consists of an assignment to the target concept, such that every the literal in it evaluates to 1. A negative example consists of an assignment to the target concept, such that there exists a literal in it that evaluates to 0. Thus negative examples convey much less information then positive examples, and the information they convey is not trivial to use. Our algorithm uses only positive examples.

Claim 3.2 *Given at least $\frac{1}{\epsilon}(2n + \log \frac{1}{\delta})$ examples, the following algorithm is a PAC learning algorithm.*

1. *initialize $h = x_1 \wedge \overline{x_1} \wedge x_2 \wedge \overline{x_2} \wedge ... \wedge x_n \wedge \overline{x_n}$*
2. *for $j = 1...m$, if $f(a^j) =$ '$+$' then do*
   - *for each $i \in \{1...n\}$ if $a_i^j = 1$ remove $\overline{x_i}$ from h*
   - *for each $i \in \{1...n\}$ if $a_i^j = 0$ remove x_i from h*

Proof: We use the following notations: By $S = \{< a^j, f(a^j) >\}_{j=1,...,m}$ we denote the set of all samples. By h^j we denote the expression h after j iterations; that is, after processing the examples $a^1, ..., a^j$.

We first show that the final hypothesis h is consistent with all examples. Using induction on j, we show that h^j is consistent with the first j examples and that h^j includes all literals in f (i.e., $h^j \supseteq f$). The induction basis ($j = 0$) holds trivially (as h^0 is as initialized in Step 1). In the induction step, suppose that $h^j \supseteq f$ is consistent with the first j examples, and consider what happens in the $j + 1^{\text{st}}$ iteration. We consider two cases

Case 1: the $j + 1^{\text{st}}$ example is positive. In this case $h^{j+1} \subseteq h^j$ and $h^{j+1} \supseteq f$ (since the only literals omitted from h^j are those that cannot be in f).

- Using $h^{j+1} \subseteq h^j$ (i.e., $h^j(a) = 1$ implies $h^{j+1}(a) = 1$), we have $h^{j+1}(a^i) = h^j(a^i) = f(a^i)$ for every $i = 1, ..., j$ satisfying $f(a^i) = 1$.
- Using $h^{j+1} \supseteq f$ (i.e., $f(a) = 0$ implies $h^{j+1}(a) = 0$), we have $h^{j+1}(a^i) = f(a^i)$ for every $i = 1, ..., j$ satisfying $f(a^i) = 0$.
- Finally, by the operation of the current interaction, $h^{j+1} \subseteq \wedge_{i=1}^{n} l_i$, where $l_i = x_i$ if $a_i^{j+1} = 1$, and $l_i = \overline{x}_i$ otherwise. Thus, $h^{j+1}(a^{j+1}) = 1$.

It follows that $h^{j+1}(a^i) = f(a^i)$ holds for $i = 1, ..., j, j + 1$.

Case 2: the $j + 1^{\text{st}}$ example is negative. In this case $h^{j+1} = h^j$ and so $h^{j+1} \supseteq f$. Also, since $f(a^{j+1}) = 0$, it follows that $h^{j+1}(a^{j+1}) = 0$. Thus, $h^{j+1}(a^i) = f(a^i)$ holds for $i = 1, ..., j, j+1$.

Let us compute the cardinality of F_n. The cardinality of F_n is bounded by 2^{2n}, since each literal can either not appear in the monomial or appear in monomial, and the number of literals is $2n$. Thus, $\log_2 |F_n| \leq 2n$, and the claim follows by applying Occam's Razor. ■

Is this version of Occam's Razor powerful enough? The following example shows that this current version is somewhat limited.

Learning a 3-term DNF**.** The concept class F_n of 3-term DNF's is a family of boolean expressions, over variables $x_1, x_2, ..., x_n$, defined as follows:

$$F_n = \{f : f = M1 \wedge M2 \wedge M3 \text{ where } M1, M2, M3 \text{ are monomials over } x_1, x_2, ..., x_n\}$$

The learning task seems similar to the previous task of learning monomials. However, the problem of finding a consistent 3-term DNF seems intractable. Specifically:

Claim 3.3 *The problem of finding a 3-term DNF that is consistent with a given set of examples is $\mathcal{NP}$-complete.*

The proof is via a reduction to 3-colorability and is omitted. Actually, the difficulty is not due to Occam algorithms only. It rather holds with respect to any PAC learning algorithm that always outputs hypotheses in the concept class (in our case a 3-term DNF). Recall that in our definition of PAC algorithms we did not insist that when learning a target concept from F_n the algorithm must output a hypothesis in F_n. However, we did make this condition when defining an Occam algorithm.

Claim 3.4 *If $\mathcal{NP}$ is not contained in $\mathcal{BPP}$ then no probabilistic polynomial-time that outputs a hypothesis in 3-term DNF can learn the class of 3-term DNF formulae.*

Proof: We show a randomized polynomial-time reduction of the problem of finding a 3-term DNF that is consistent with a given set of examples to the problem of PAC learning 3-term DNF's via such hypotheses.

Let L be a PAC learning algorithm of the latter form, and suppose we are given a set of m instances, denoted S, labeled by some 3-term DNF, denoted f. We invoke algorithm L on input parameters $\epsilon = 1/2m$ and $\delta = 1/3$, and feed it with a sequence of (labeled) examples uniformly distributed in S. (This sequence may contain repetirtions.) Thus, L is running with distribution D which is uniform on S, and outputs a 3-term DNF hypothesis h satisfying

$$\Pr\left[err_{f,D}(h) \geq \frac{1}{2m}\right] \leq \frac{1}{3}$$

Since each $s \in S$ has probability mass $1/m$, it follows that

$$\Pr[\exists s \in S \text{ s.t. } h(s) \neq f(s)] \leq \frac{1}{3}$$

Thus, with probability $2/3$ the hypothesis h is consistent with f on S. Invoking Claim 3.3, the current claim follows. ■

Discussion: What the last claim says is that if we insist that the learning algorithm outputs a hypothesis in the concept class being learned (as we do in case of Occam's Razor) then we cannot learn 3-term DNF formulae. In the next section, we shall see that the latter class can be learned if we allow the hypothesis class to be different.

4 Generalized definition of PAC learning algorithm

Oded's Note: *This section was drastically revised by me.*

In accordance with the above discussion we define explicitly the notion of learning one concept class with a possibly different class of hypotheses, which typically is a superset of the concept class.

Definition 5 (PAC learning, revisited): *Let $F = \cup_n F_n$ and $H = \cup_n H_n$ so that $F_n \subseteq H_n$ are classes of functions mapping X_n to $\{0,1\}$. We say that algorithm A PAC* learns the concept class F using the hypothesis class H *if the following holds: For every n, for every function $f \in F_n$, for every distribution $D : X_n \longrightarrow [0,1]$, for every error parameter $0 < \epsilon < 1$ and for every confidence parameter $0 < \delta < 1$, given parameters n, ϵ, δ and a set of f-labeled examples, $\{< x^i, f(x^i) >\}$, where x^i is drawn independently under the distribution D, the algorithm outputs a hypothesis $h \in H_n$ such that:*

$$\Pr[err_{f,D}(h) \leq \epsilon] \geq 1 - \delta$$

where the probability is taken over the choice of examples, as well as over the internal coin tosses of the algorithm.

That is, the only change relative to Definition 2 is the condition that the output hypothesis h belongs to H_n. In case $H = F$ we say that the algorithm is a proper learning algorithm for F.

In contrast to the negative results in the previous section, we show that the class 3-term DNF can be efficiently learned using the hypothesis class 3CNF. This statement is proven via a reduction of this learning task to the task of learning monomials (already solved efficiently above). To present this reduction, we first define what we mean in general by reduction among learning tasks.

4.1 Reductions among learning tasks

Reductions are a powerful tool common in computer science models. It is only natural to define such notion for the model of PAC learning.

Definition 6 *We say that the concept class F over the instance space X, is PAC-reducible to the concept class F' over the instance X' if for some polynomial p*

- *there exists a polynomial-time computable mapping G from X to X' such that for every n and for every $x \in X_n$, $\quad G(x) \in X_{p(n)}$.*
- *there exists a polynomial $q()$ such that for every concept $f \in F_n$, there exists a concept $f' \in F'_{p(n)}$ such that $size(c') \leq q(size(c))$ and for every $x \in X_n$, $\quad f(x) = f'(G(x))$.*

Note that the second item does not require an efficient transformation between f and f'. In fact, for proper learning (by efficient algorithms), an efficient transformation from F' to F must be required.

Theorem 7 *Let F and F' be concept classes. If F is PAC-reducible to F', and F' is efficiently PAC learnable, then F is efficiently PAC learnable.*

Proof: Given an efficient PAC learning algorithm L'. We use L' to learn a an unknown target concept $f \in F_n$ as follows: Given an example labeled $< x, f(x) >$, where $x \in X_n$, we compute an example labeled $< G(x), f(x) >$, where $G(x) \in X'_n$, and supply it to L'. The original examples are chosen by a distribution D on X, and so the reduced examples $G(x)$ (computed by us) are drawn by distribution D' on X' induced by ditribution D. Also, let f' be the function associated by Item 2 to f (i.e., $f'(G(x)) = f(x)$). Algorithm L' will output an hypothesis h' such that:

$$\Pr[err_{f',D'}(h') \leq \epsilon] \geq 1 - \delta$$

where the probability depends on a sample of D'. We take the composition of h' and G to be our hypothesis h for f (i.e., $h(x) = h'(G(x))$).

We need to evaluate $\Pr[err_{f,D}(h) \leq \epsilon]$, where $h = h' \circ G$ and h' is the output of L'. We first observe that

$$\begin{aligned} err_{f,D}(h) &= \Pr_{x \sim D}[h(x) \neq f(x)] \\ &= \Pr_{x \sim D}[h'(G(x)) \neq f'(G(x))] \\ &= \Pr_{x' \sim D'}[h'(x') \neq f'(x')] \\ &= err_{f',D'}(h') \end{aligned}$$

and so $\Pr[err_{f,D}(h) \leq \epsilon] \geq 1 - \delta$, as required. ■

Learning a 3-term *DNF*, revisited. Recall that, assuming $\mathcal{NP}$ is not contained in $\mathcal{BPP}$, it is infeasible to *properly* learn 3-term DNF's (i.e., learn this class by hypotheses in it). In contrast, we now show that it is feasible to learn 3-term DNF's by 3CNF hypotheses. Actually, we show that it is feasible to (properly) learn the class of 3CNF (which contain via a easy reduction all 3-term DNF's).[1] This is shown by reducting the learning of 3-CNF's to the learning of monomials.

Claim 4.1 *Learning 3-CNF's is reducible to learning monomials. Furthermore, there exists an efficient algorithm for* properly *learning 3-CNF.*

Proof: We define the following transformation G from 3-CNF instance space X_n, to the monomial instance space X'_k, where $k = 8 \cdot \binom{n^3}{3}$. For each of the possible k clauses, we associate a distinct variable, and the transformation from 3CNF to monomials just maps the set of clauses in the 3CNF into the set of variables. (Indeed, we reduce to learning monotone monomials.) The transformation G does the analogous thing; that is, it maps truth assignments to the n 3NCF variables onto truth assignments to the k monomial varaibles in the natural way (i.e., a monomial variable representing a possible clause is assigned the value to which this clause evaluates under the assignment to the 3CNF variables). This transformation satisfies the conditions of Definition 6.

To show the furthermore-part, observe that the hypothesis constructed by the reduction-algorithm (given in the proof of Theorem 7) can be readily put in 3CNF with respect to the space X_n. ■

4.2 Generalized forms of Occam's Razor

We have seen that it is worthwhile to use a hypothesis from a wider class of functions (than merely from the concept class). A staightfoward generalization of Occam's Razor to a case where the algorithm outputs a hypothesis from a predetermined hypothesis class follows.

Theorem 8 (Occam's Razor – generalization to predetermined hypothesis class): *Let $F_n \subseteq H_n$ be finite concept classes. Let A be an algorithm such that for every n, for every $f \in F_n$ and for every*

[1] Applying the distributional law to a 3-term DNF over n variables, we obtain a 3CNF with at most $(2n)^3$ clauses.

number of examples labeled by f, the algorithm outputs an hypothesis $h \in H_n$, that is consistent with all the given examples. Then for any distribution D, for any error parameter $0 < \epsilon < 1$ and for any confidence parameter $0 < \delta < 1$ if the number of examples drawn independently by D is larger or equal $\frac{1}{\epsilon}(\log \frac{1}{\delta} + \log |H_n|)$, then

$$\Pr[err_{f,D}(h) \leq \epsilon] \geq 1 - \delta$$

The proof is by a straightforward adaptation of the proof given to Theorem 4 (which is indeed a special case obtained by setting $H_n = F_n$).

A wider generalization is obtained by not determining a priori the hypothesis class. In such case, we need some other way to bound the hypothesis from merely recording all examples. This is done by requiring that the hypothesis's length is strictly shorter than the number of examples seen. This leads to the following formulation.

Definition 9 *Let α and $\beta < 1$ be constants. We say that L is an (α, β)-*Occam's algorithm *for F if given m examples, L outputs a hypothesis h such that h is consistent with every given example and*

$$size(h) \leq (n \cdot size(f))^{\alpha} m^{\beta}$$

Theorem 10 *For every α and $\beta < 1$, an (α, β)-Occam's algorithm can be turned into a PAC learning algorithm by running it on $O(\epsilon^{-1} \cdot (\mathrm{poly}(n \cdot size(f)) + \log(1/\delta))$ examples.*

Proof: Firstly, we generalize Theorem 8 to a setting in which the Occam algorithm may uses different hypothesis classes for different number of examples. Specifically, suppose that when seeing m examples labelled by $f \in F_n$, the algorithm outputs a hypothesis in $H_{n,m}$. Then such an algorithm is a PAC learner provided

$$m \geq \epsilon^{-1} \cdot (\log(1/\delta) + \log |H_{n,m}|),$$

where ϵ and δ are the error and confidence parameters (given to the PAC version).

Now, the conditions of the current theorem provide such a "dynamic" hypothesis class, $H_{n,m}$, and the upper bound on the length of hypothesis guarantees that

$$\log |H_{n,m}| \leq (n \cdot size(f))^{\alpha} m^{\beta}$$

So, since $\beta < 1$, for sufficiently large $m = \mathrm{poly}(n \cdot size(f))/\epsilon$, we have $\epsilon m \ll (n \cdot size(f))^{\alpha} m^{\beta}$. The theorem follows. ∎

5 The (VC) Vapnik-Chervonenkis Dimension

In all versions of Occam's Razor discussed above, we assumed that the hypothesis class is finite. In general this is not necessarily the case. For example, the natural hypothesis class for learning axis-aligned rectangles over $[0, 1]^2$ is the infinite set of all possible axis-aligned rectangles. Therefore we can not apply Occam's Razor to this problem. We would like a tool with similar flavor for the case of infinite hypothesis classes. The first step for achieving this goal, is finding a parameter of finite value that characterizing also infinite classes.

Definition 11 (shattering a set): *A finite subset S of instance space X_n is* shattered by a family of functions *F_n if for every $S' \subseteq S$ there exists a function $f \in F_n$ such that*

$$f(x) = \begin{cases} 1 & x \in S' \\ 0 & x \in S \backslash S' \end{cases}$$

That is for $S = \{x^1, ..., x^m\}$ and for every $\alpha \in \{0,1\}^m$ there exists a function $f \in F$, such that for any $i = \{1...m\}$, $\quad f(x^i) = \alpha^i$, where α^i is the i^{th} bit of α.

Definition 12 (VC dimention): *The* VC dimention of a set of functions *F, denoted VC-dim(F), is the maximal integer d such that there exists a set S of cardinality d that is shattered by F.*

5.1 An example: VC dimension of axis aligned rectangles

The following example demonstrates the computation of VC dimension of a family of functions.

Proposition 5.1 *VC-dim(axis-aligned rectangles)* $= 4$

Proof: We start by exhibiting a set of four points in $[0,1]^2$ that can be shattered. For every labeling of the set of four points exhibited in Fig. 3, we can find a rectangle that induces such labeling.

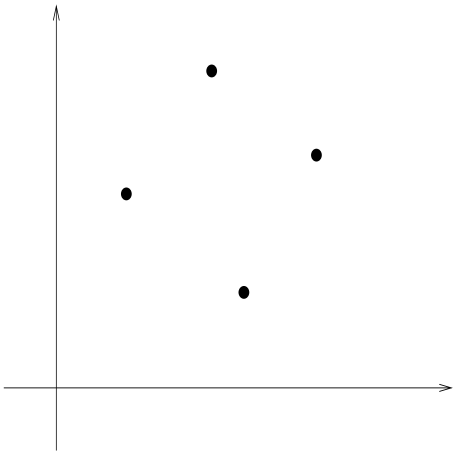

Figure 3: Four point that in $[0,1]^2$, that can be shattered

1. In case all points are labeled "+", we take the rectangle $[0,1]^2$ itself.
2. In case all points are labeled "-", we take the empty rectangle.
3. For the case three points are labeled "-" and one point "+", take a "small" rectangle that covers only the point labeled "+".
4. For the case three points are labeled "+" and one point"-", we take rectangles as can be seen in Fig. 4.
5. For the case two points are labeled "-" and two points "+", we take rectangles as can be seen in Fig. 5. (The figure shows only 4 out of the 6 subcases; the other two are easier.)

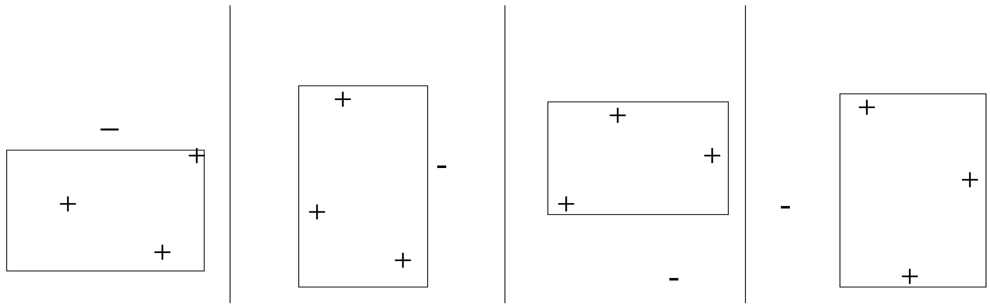

Figure 4: rectangle covering the three points labeled "+"

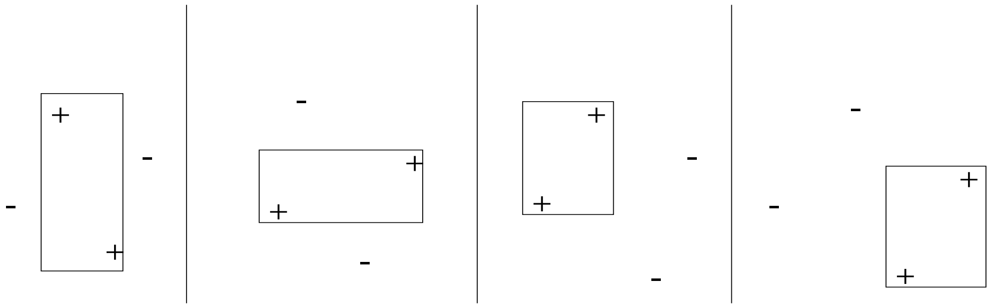

Figure 5: rectangle covering the two points labeled "+"

It remains to prove that for every set of five points in $[0,1]^2$, there exists is a labeling, such that it can not be induced by any rectangle in $[0,1]^2$.

By the proof of Claim 2.1 (the learning algorithm for axis aligned rectangles), for every set of five points in $[0,1]^2$, at most four of them determine the minimal rectangle that contains the whole set. Then no rectangle is consistent with the labeling that assigns these 4 boundary points '+' and assigns the remaining point (which must reside inside any rectangle coving these points) a '-'. ■

5.2 General bounds

VC dimension is linked to the sample complexity of PAC learning via the following two theorems:

Theorem 13 (upper bound on sample complexity)*: Let H, F be function classes such that $F \subseteq H$. Let A be a algorithm that given a labeled sample always outputs a hypothesis $h \in H$ consistent with the sample. Then there exists a constant c_0, such that for any target function $f \in F$, for any underlying distribution D, any error parameter $0 < \epsilon < 1$ and any confidence parameter $0 < \delta < 1$, if the number of examples is greater or equal:*

$$\frac{c_0}{\epsilon} \cdot \left(VC\text{-}dim(H) \cdot \log \frac{1}{\epsilon} + \log \frac{1}{\delta} \right)$$

then with probability greater then $1 - \delta$ we get

$$err_{D,f}(h) \leq \epsilon$$

where h is the hypothesis output by A.

Theorem 14 (lower bound on sample complexity)*: Any PAC learning algorithm for learning a concept class F such that $d = VC\text{-}dim(F)$ requires $m = \Omega(\frac{1}{\epsilon} \cdot (\log \frac{1}{\delta} + d))$ examples.*

We note that there is a gap of factor $\log \frac{1}{\epsilon}$ between the bounds. The proof of the these theorems is complex. Instead we prove a slightly weaker lower bound theorem:

Theorem 15 (lower bound, slightly weaker form): *Any PAC learning algorithm for learning a concept class F such that $d = VC\text{-}dim(F)$ requires requires $\Omega(\frac{d}{\epsilon})$ examples in the worst case.*

Oded's Note: *The proof was revised by me.*

Proof: Since the VC dimension is d, there exists a set of d points shattered by F. We denote this set by $S = \{e^1, ..., e^d\}$. Since S is shattered by F, for each possible labeling of S there exists a function $f \in F$ which is consistent with this labeling. Let us denote by $f_\alpha \in F$ a function consistent with the labeling $\alpha = \alpha_1 \cdots \alpha_d \in \{0,1\}^d$; i.e., $f_\alpha(e^i) = \alpha_i$ for each $i = 1, ..., d$. Let $F' = \{f_\alpha : \alpha \in \{0,1\}^d\} \subseteq F$.

We consider error parameter $\epsilon \leq \frac{1}{8}$, and an arbitrary confidence parameter $0 < \delta < \frac{1}{2}$.

We start by proving a bound of $\Omega(d)$. Towards this goal, we define the underlying distribution D to be uniform over S; that is, every point in S is assigned probability $\frac{1}{d}$, and all other instances are assigned zero probability.

Let us assume, in contrary to the claimed bound, that $\frac{d}{2}$ examples suffice. Under this assumption in the best case, $\frac{d}{2}$ different examples where drawn. We denote this set by S_1, and let $S_2 \stackrel{\rm def}{=} S \setminus S_1$. Intutively, the algforithm only got the labeling of S_1 and so there is no way it can distinguish between the $2^{d-|S_1|}$ functions in F' consistent with S_1.

Formally, we consider a target function f chosen uniformly in F'. The algorithm then obtain a sample S_1 labeled by f, and outputs a hypothesis h. We are interested in the behavior of the random variable $err_{f,D}(h)$. Recall that this random varaible is defined over the probability space defined by the uniform choice of $f \in F'$, the uniform choice of $d/2$ points in S (yielding S_1), and additional coin tosses the learning algorithm may do. Let us reverse the "natural" order of the randomization, and consider what happens when S_1 is selected first, and $f \in F'$ is selected next. Furthermore, we select f in two phases: first we select at random the value of f on S_1, and we postpone for later the random choice of the value of f on S_2. (Together, the values of f on S_1 and S_2 will determine a unique $f \in F'$, which will be uniformly distributed.) However, the learning algorithm is oblivious of the latter choices (i.e., of the values of f on S_2), and so the hypothesis h is stochastically independent of the latter. So the full order of events we consider is:

1. d points are selected uniformly in S, determining S_1 and S_2.
2. one assigns uniformly Boolean values to the points in S_1 and presents these to the learning algorithm.
3. the learning algorithm outputs a hypothesis h.
4. one assigns uniformly Boolean values to the points in S_2, thus determining a unique function $f \in F'$.

Since f is determined on S_2 only after h is fixed, and its values on S_2 are uniformly distributed in $\{0,1\}$, the expected value of $err_{f,D}(h)$ is at least

$$\frac{1}{2} \cdot \frac{|S_2|}{|S|} \geq \frac{1}{2} \cdot \frac{d/2}{d} = \frac{1}{4}$$

Thus, there exists $f \in F' \subseteq F$ so that with probability at least $1/2$,

$$err_{f,D}(h) \geq \frac{1}{8}$$

This establishes a lower bound of $d/2$ on the sample complexity, for any $\epsilon \leq 1/8$ (and $\delta \leq 1/2$).

In order to prove the stronger bound, stated in the theorem, we modify the distribution D as follows. Fixing an arbitrary element $e^1 \in S$, we let D assign it probability $1 - 8\epsilon$, and assign each other element of S probability $8\epsilon/(d-1)$. (Again, only points in S are assigned non-zero probability.) If we take a sample of m points from D, we expect only $8\epsilon m$ points to be different than e^1. Thus, if $m < (d-1)/(20\epsilon)$ then with very high probability only $10\epsilon m < (d-1)/2$ points will be different than e^1. Applying an argument as above, we conclude that our error with respect to D is expected to be the probability assigned to points not seen by the algorithm times one half; that is,

$$\left(\frac{d-1}{2} \cdot \frac{8\epsilon}{d-1}\right) \cdot \frac{1}{2} = 2 \cdot \epsilon$$

Thus, there exists $f \in F' \subseteq F$ so that with probability at least $1/2$, $err_{f,D}(h) \geq \epsilon$. This establishes a lower bound of $(d-1)/(20\epsilon)$ on the sample complexity, for any $\epsilon \leq 1/8$ (and $\delta \leq 1/2$). ■

6 Appendix: Filling-up gaps for the proof of Claim 2.1

Oded's Note: *The gap left open in the proof was the assumption that we can slides the border of the A_i's so that the weight of each of them is exactly $\epsilon/4$. Firstly, we repeat the comment by which it is not essential to have these rectanges have weight exactly $\epsilon/4$, and it suffices to have each of the A_i's have weight $\Theta(\epsilon)$. But still it may be that there is a probability mass of $\Omega(\epsilon)$ residing on one single axis-allinged line. This problem can be resolved is several ways. For example, one may pertub all points at random, and argue that the performance of the algorithm cannot be improved (a proof is indeed called for!). Alternatively, one may argue seperately for these patalogical cases of single lines having probability mass of $\Omega(\epsilon)$.*