VIII Efficient Learning III – EPAC Learning

1. Introduction
2. Review: Notation
3. Review: Learning C^k is Hard
4. Learning $D_{n,k}$ is Not Hard
5. Learning $(C^k, D_{n,k})$ is Not Hard
6. Efficiency, Confidence and Accuracy
7. Efficiency and Representation Size
8. EPAC Learning
9. Summary of Lecture

1 Introduction

In lecture VI we saw the importance of efficiency and that some concept spaces are harder to learn than others.

In this lecture we see some other factors which affect efficiency:

- the size of the learner's hypothesis space.
- accuracy ϵ.
- confidence δ.
- the representation used by the learner.

We'll combine all these factors and define EPAC learning: Efficient PAC learning.

Note: we'll skip sections 6.4, 6.5 and 6.6 of Anthony and Biggs.

2 Review: Notation

- We have a concept space C and a hypothesis space H.
- The teacher has a target concept $t \in C$ in mind.
- The learner searches for an approximation $h \in H$.

If $C = H$ we speak of learning algorithms for H:

$$H \to \boxed{\text{learning}} \to H$$

If $C \neq H$ we speak of learning algorithms for (C, H):

$$C \to \boxed{\text{learning}} \to H$$

3 Review: Learning C^k is Hard

Last lecture we saw that learning C^k (in other words $C = H = C^k$) is hard for $k \geq 2$. That is,

- There is no efficient algorithm for learning C^k (unless P = NP).

4 Learning $D_{n,k}$ is Not Hard

In lecture 2 we saw an algorithm for learning in $D_{n,k}$ (at most k disjunctions of n-variable monomials).

This algorithm has running time $O(mn^k)$, i.e., polynomial in m (number of examples) and n (number of bits per example) (see Anthony and Biggs, page 40).

Note, however, that $C_n^k \subseteq D_{n,k}$.

So this algorithm will, for any concept in C^k, efficiently learn a hypothesis in $D_{n,k}$.

In summary:

- $C_n^k \to \boxed{\text{learning}} \to C_n^k$ is hard.
- $D_{n,k} \to \boxed{\text{learning}} \to D_{n,k}$ is not hard.

But $C_n^k \subseteq D_{n,k}$ so:

- $C_n^k \to \boxed{\text{learning}} \to D_{n,k}$ is not hard.

Why?

5 Learning $(C^k, D_{n,k})$ is Not Hard

Why is it hard to learn C^k, but not $(C_n^k, D_{n,k})$?

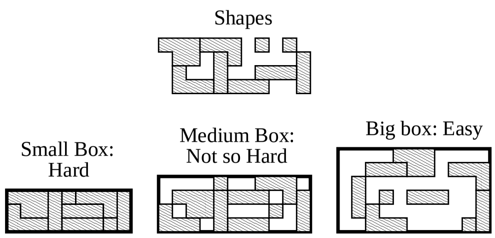
$C_n^k \subseteq D_{n,k}$ so $D_{n,k}$ is bigger than C_n^k.

Why, in this case, it is easier to learn when H is bigger?

Intuitively, it should be *harder*, since there are more points (formulae) in the search space to consider.

Why it is easier to learn when H is bigger?

Problem: put all the shapes inside a box.



Informally, $D_{n,k}$ is a bigger box than C_n^k.

Conclusion: Concept Spaces and Difficulty

- The difficulty of learning concepts in C^k depends partly on the learner's choice of hypothesis space H.
- A space which is too small may make learning more difficult.

6 Efficiency, Confidence and Accuracy

In PAC learning we use δ to define *confidence* and ϵ to define *accuracy*:

- Smaller δ means we want to be *more* confident.
- Smaller ϵ means the learner must be *more* accurate.

Confidence, accuracy and efficiency are related:

- To increase confidence we need to train the learner more.
- To decrease error we need to train the learner more.

More training requires:

- More labelled examples.
- More time.

So δ and ϵ affect efficiency in terms of both:

- Sample complexity.
- Running time.

Efficiency with Respect to Accuracy

We say an algorithm L is efficient with respect to accuracy if:

- its running time is polynomial in m and ...
- the sample complexity depends polynomially on $1/\epsilon$.

Efficiency with Respect to Confidence

We say an algorithm L is efficient with respect to confidence if:

- its running time is polynomial in m and ...
- the sample complexity depends polynomially on $1/\delta$.

Note: Anthony and Biggs specify $\ln(1/\delta)$ on p. 51 but explain on p. 68 that it has been shown that $1/\delta$ is just as good.

Consistent Learners are Efficient in a Finite H for any δ, ϵ

Recall our upper bound on sample complexity for a consistent learner in a finite space:

$$m_0 = \left\lceil \frac{1}{\epsilon} \ln \frac{|H|}{\delta} \right\rceil$$

and observe that sample complexity *does* increase polynomially with $1/\epsilon$ and $1/\delta$.

I.e. a consistent algorithm will learn efficiently in a finite space for any values of ϵ and δ such that $0 \leq \delta, \epsilon \leq 1$.

7 Efficiency and Representation Size

A function can often be written in more than one way in the same language. E.g., this function:

u_1	u_2	u_3	$f(u)$
0	0	0	0
0	0	1	0
0	1	0	0
0	1	1	0
1	0	0	1
1	0	1	1
1	1	0	1
1	1	1	1

can be written in different ways in DNF:

$$f(u) \quad = \quad u_1 \wedge (u_1 \vee u_2) \wedge (u_3 \vee \bar{u}_3) \quad = \quad u_1$$

Minimal Representations

A function has a *minimal representation* in a given language: the smallest/shortest way of representing it in that language.

E.g., u_1 is the minimal representation of the concept on the previous slide, when using DNF.

Note: Sometimes there is more than one minimal representation of a function using the same language.

Function Complexity

Some concepts are more complex than others. It takes more information to describe complex functions.

$f_1 = \langle u_1 \rangle$ has 1 literal.
$f_2 = \langle u_1 u_2 u_3 \rangle$ has 3 literals.
$f_3 = (u_1 \bar{u}_2 \bar{u}_3) \vee (\bar{u}_1 u_2 \bar{u}_3) \vee (\bar{u}_1 \bar{u}_2 u_3) \vee (u_1 u_2 u_3)$ has 12 literals.

Function Complexity and Efficiency

We can measure the complexity of a function by finding the length of its minimal representation.

- Let r be the minimal representation size of a concept.

The difficulty of learning a concept depends partly on its complexity.

- To be efficient, an algorithm's sample complexity should be polynomial in r.

Note: Finding the minimal representation of a function is a hard problem (Anthony and Biggs, pages 57–58).

Different Languages

In different languages the minimal representation of a function will have different lengths.

Example 1: DNF and CNF

u_1	u_2	u_3	$f(u)$
0	0	0	0
0	0	1	0
0	1	0	0
0	1	1	0
1	0	0	0
1	0	1	1
1	1	0	1
1	1	1	1

f in minimal DNF has 4 literals: $\langle u_1 u_2 \vee u_1 u_3 \rangle$.
f in minimal CNF has 3 literals: $\langle u_2 u_3 \wedge u_1 \rangle$.

Example 2: The Parity Function

The n-bit parity function can be computed by:

- just n XOR gates.
- a circuit of AND, OR and NOT gates whose size is bounded by a fixed polynomial in n.
- a DNF expression whose size is exponential in n.

So, depending on the language used, the size of the minimal representation of parity is linear, polynomial or exponential in n.

8 EPAC Learning

We say an algorithm L is *efficiently* PAC or EPAC if:

- its running time is polynomial in m and n,
- its sample complexity is polynomial in $n, r, 1/\delta$, and $1/\epsilon$.

9 Summary of Lecture

Difficulty depends on ϵ and δ; higher confidence and greater accuracy require more examples (increase the sample complexity), and the running time.

Efficiency also depends on r, the minimal representation size of a concept, since some concepts have exponential representations in some languages.

Putting things together we get the idea of *efficient* PAC (EPAC) learning.

Summary Continued

We also saw that the difficulty of learning concepts from C^k depends on the hypothesis space used:

- learning C^k is hard
- but learning $(C^k_n, D_{n,k})$ is not hard.