Computational Learning Theory

Kurt Mehlhorn

June 24, 2013

In 1984, Leslie Valiant wrote a very influential article (close to 7000 citations) with the title a Theory of the Learnable [Val84], in which he proposes an approach to study the phenomenon of learning from a computational point of view. He suggests to view learning as the task of acquiring an algorithm without explicit programming.

1 Valiant's 84 article: The birth of PAC-learning

Valiant introduces a precise computational model for *concept learning*. Concept learning is about learning to decide whether a certain data belongs to a certain concept (is this a table? Is there an elephant in the data?). The paper is the birth of PAC-learning (PAC = Probably Approximately Correct).

What is a concept? A concept is simply a subset of some domain. A concept comes from a some concept class C known to the learner. So the learner is not completely in the dark. Rather, he/she knows that the concept to learn comes from a certain concept class.

How does one get information about a concept? *Either passively by examples*. The learner is shown examples drawn according to some unknown probability distribution D. With each example comes the information, whether the example belongs to the concept or not. *Or actively, by oracle calls*. The learner can ask whether an specific object which he/she creates belongs to the concept or not. In later papers there is also supervised learning where a teacher provides strategically chosen examples.

What does it mean to have learned a concept? It means to produce a hypothesis h from some hypothesis class H that is close to c, i.e., is able to classify unknown examples with high confidence (= small error probability). The examples on which the learner has to show its skills are drawn according to the *same* probability distribution D which is used in the learning phase.

What does it mean to have a learning algorithm for a concept class? The algorithm should be able to learn any concept in the concept class with high probability of success (= small failure probability) after seeing a polynomial number of examples.

A Formalization: Let C be the concept class, let $c \in C$ be the concept to be learned, and let D be the probability distribution according to which examples are drawn. Let H be the set of possible hypothesis of the learner. The error probability of a hypothesis $h \in H$ is defined as the probability of c and h differing on a random example drawn according to D.

$$\text{err}(h) := \text{prob}_{x\sim D}[h(x) \neq c(x)]$$

If the error probability is small, say at most ε, the hypothesis is *approximately correct*. We say that h is ε-good if $\text{err}(h) \leq \varepsilon$ and ε-bad otherwise.

Since the algorithm learns through random examples, one can never guarantee that an algorithm always finds an approximately correct hypothesis. The best one can hope for is that the algorithm fails with small probability, say at most δ, i.e.,

$$\text{prob}(\text{err}(h) > \varepsilon) \leq \delta.$$

The probability is with respect to the runs of the algorithm. If the algorithm is deterministic, this is simply the probability of seeing the particular sequence of examples that lead the algorithm to the hypothesis h.

We are now ready for the formal definition of PAC-learning.

Definition 1 *A concept class C is PAC-learnable by a hypothesis class H, if there is an algorithm A with the following properties: For all concepts $c \in C$, positive $\varepsilon > 0$, positive $\delta > 0$, and distributions D on the examples, the algorithm takes some number m of examples $(x_1, c(x_1))$, $(x_2, c(x_2))$, $\ldots$, $(x_m, c(x_m))$ and produces a hypothesis h such that* $\text{prob}(\text{err}(h) > \varepsilon) \leq \delta$. *The number m of examples must be polynomial in $1/\varepsilon$ and $1/\delta$.*

If a particular sequence of examples causes the algorithm to produce an incorrect hypothesis, the probability of this sample sequence contributes to the error probability.

The algorithm is efficient if it runs in time polynomial in $1/\varepsilon$ and $1/\delta$ (equivalently, polynomial in m).

A First Example (not from Valiant's article): Our concept class C is equal to the set of positive rays on the real line, i.e., if $r \in \mathbb{R}$ is a real number, it defines the concept $c = [r_c, \infty]$. A positive example is any number greater or equal to r, and a negative example is any number smaller than r.

The set of hypothesis is the same set of intervals. It is clear what a learning algorithm should do. After seeing some number of example, it should output an interval $[h, \infty]$, where h is larger than all the negative examples and no larger than any of the positive examples. For concreteness, we let the algorithm output the smallest possible example as its hypothesis.

For how many examples should the algorithm ask before it commits?

Which examples are misclassified by a hypothesis h? Exactly the examples in the interval $I = [r_c, h)$. Examples less than r_c are classified as negative by c and h, examples no smaller than h are classified as positive by c and h, and points in I are classified as positive by c and as negative by h.

Let $b^+ > r_c$ be such that the interval $R^+ = [r_c, b^+]$ has probability ε according to D (= the unknown distribution according to which samples are drawn).

h is a bad hypothesis, if $h > b^+$. This can only be the case no sample in R^+ was drawn. Since samples are drawn independently, we have

$$\text{prob}(x_1 \notin R^+ \wedge \ldots x_n \notin R^+) \leq (1-\varepsilon)^m.$$

We want the error probability to be at most δ and hence should choose m such that $(1-\varepsilon)^m \leq \delta$. Taking logarithm gives us $m \ln(1-\varepsilon) \leq \ln \delta$. Since $\ln(1-\varepsilon) \leq -\varepsilon$, we have $m \ln(1-\varepsilon) \leq \ln \delta$ if $-m\varepsilon \leq \ln \delta$ or $m \geq \frac{1}{\varepsilon} \ln(1/\delta)$. We summarize the discussion in:

The algorithm above is a PAC-learning algorithm: $m = \frac{1}{\varepsilon} \ln(1/\delta)$ questions are sufficient.

Learning Monotone Boolean Formulae (from Valiant's article): The concept class C is the set of all monotone boolean functions on t variables. We extend the boolean domain $\{0, 1\}$ by an element $*$ with $* < 0 < 1$ and extend f as follows: $f(v) = 1$ if and only if $f(w) = 1$ for all vectors w obtained

from v by replacing every $*$ by a concrete value. Samples are in $\{*,0,1\}^t$. If a position in a sample is equal to $*$, the position is irrelevant for the concept.

Let f be the concept to be learned. I write f instead of c since the concept is a function. Here is the learning algorithm. The algorithm starts with the hypothesis equal to the all-zero function and maintains the invariant $h \leq f$. In each iteration it asks for a positive example v, i.e., an example with $f(v) = 1$. Negative examples are not needed for this algorithm. It then finds out whether the example can be generalized. It sets every position in v which is determined to $*$ and asks whether the generalized example belongs to the concept.

```
1 initialize h to the all-zero function;
2 for L times do
3     v := some positive example, i.e., f(v) = 1;
4     if h(v) = 0 then
5         for i = 1 to t do
6             obtain ṽ from v by setting v_i to *;
7             if f(ṽ) = 1 (a call to ORACLE) then v := ṽ
8         let m be the product of all variables with v_i = 1;
9         h := h + m; // m is a prime implicant of f
```

A prime implicant of a function f is a minimal (= least number of variables) monomial m with $m \leq f$. Any function can be written as a disjunction of its prime implicants. In general, a subset of the prime implicants suffices. Let the degree d of a function be the smallest number of prime implicants that suffice to express f as their sum. For monotone functions all prime implicants a required in the sum.

Theorem 1 *Let f be a monotone function of degree d in t variables and let $L \geq \frac{2}{\varepsilon} max(d, 4\ln\frac{1}{\delta})$. Then the algorithm above terminates with a hypothesis h with $\mathrm{err}(h) \leq \varepsilon$ with probability at least $1-\delta$.*

We proceed in two steps:

(a) We show that each iteration with $h(v) = 0$ adds a prime implicant to h. In particular, $h \leq f$ always.

(b) We determine the right value of L.

Let v^0 be the example produced in line 3 and let v^i be the value of v after the i-th execution of the inner for-loop; v^t is the final value of v. In v^t any component is either $*$ or 1. Moreover, $f(v^t) = 1$, and if z is obtained from v^t by setting a component which is 1 to 0, $f(z) = 0$. This is easily seen. Consider the i-th iteration of the loop. Let $w = v^{i-1}$ and let w_b be obtained from w by setting the i-th bit of w to $b \in \{0,1,*\}$. Then $f(w^0) \leq f(w^1) = 1$. The inequality and the equality follows from monotonicity of f; observe that $w \leq w^1$. If $f(w^0) = 1$, then $f(w^*) = 1$ by definition and hence $v^t_i = *$. If $f(w^0) = 0$, $v^t_i = 1$. Moreover, if z is obtained from v^t by turning the i-th component to zero, then $z \leq w^0$ and hence $f(z) = 0$. We conclude that each execution of the inner for-loop adds a prime implicant of f to h. Thus the inner for-loop is executed at most d times and there are at most dt calls to the ORACLE.

What is the correct value of L? Let R be the set of examples x with $h(x) = 0 < 1 = f(x)$. The set R shrinks over time. If $\mathrm{prob}(x \in R) \leq \varepsilon$ at termination, the algorithm succeeds. If $\mathrm{prob}(x \in \mathbb{R}) > \varepsilon$ at termination, the algorithm fails.

Let X_i be a 0,1-random variable with $X_i = 1$ if the i-th execution of line 3 generates a v with $h(v) = 0$. If the algorithm fails, $X = X_1 + \ldots + X_L < d$ and $\text{prob}(X_i = 1) \geq \varepsilon$ for all i. The following Lemma tells us how to choose L.

Lemma 1 (Chernoff Bound) *Let $X_1, \ldots, X_L$ be L independent 0,1-valued random variables with* $\text{prob}(X_i = 1) \geq \varepsilon$ *for all i and let $X = X_1 + \ldots + X_L$ be the total number of successes. Then for any $\gamma > 0$*

$$\text{prob}(X \leq (1-\gamma)\varepsilon L) \leq e^{-\gamma^2 \varepsilon L/2}.$$

We apply the Lemma with $d = (1-\gamma)\varepsilon L$ or $\gamma = 1 - d/(\varepsilon L)$. We want $e^{-\gamma^2 \varepsilon L/2} \leq \delta$. For $\varepsilon L \geq 2d$ we have $\gamma \geq 1/2$ and hence $e^{-\gamma^2 \varepsilon L/2} \leq e^{\varepsilon L/8}$. The latter quantity is bounded by δ for $L \geq (8/eps)\ln(1/\delta)$. We therefore work with $L = \max(2d/\varepsilon, (8/\varepsilon)\ln(1/\delta))$.

Is the restriction to monotone function necessary? At least the algorithm above does not generalize. Observe that we need to test whether $h(v) = 1$, where v is a vector over $\{0, 1, *\}$ and h is given as a sum of monomials. This test is an NP-complete problem. Observe if $v = *^t$, the question amounts to asking whether h is identically 1, i.e., is a tautology. However, if one restricts to boolean functions in the strict sense, the algorithm works. It is only the ORACLE that has to solve an NP-complete problem.

Valiant gives more positive examples in his paper.

Negative Results: Can everything be learned in the PAC-model. In his original paper, Valiant argues informally that the answer is no. The argument was made precise by Kearns and Valiant in 1989.

Valiant argues the class of functions computable by polynomial-sized circuits is not learnable. Consider a cryptographic scheme that uses an encryption function E_k, where k is the key. Also assume that the encryption function can be computed by a polynomial-size circuit. Learnability of E_k would imply that E_k is vulnerable by a chosen plaintext attack. Observe that PAC-learnability of E_k means that with good probability the learner can deduce a good approximation of E_k after choosing a polynomial number of plaintexts m_1 to m_L to which the ORACLE provides the corresponding encoding. If cryptographic schemes immune to chosen plaintext attacks exist (which is a common assumption in cryptography), there are easy to compute functions that cannot be learned.

2 Further Literature

The book by Kearns and Vazirani [KV94] is an excellent textbook on computational learning theory. Valiant has extended his theory of learning to a theory of evolution (= learning by a population)[Val13].

References

[KV94] Michael J. Kearns and Umesh V. Vazirani. *An Introduction to Computational Learning Theory*. MIT Press, 1994.

[Val84] Leslie Valiant. A theory of the learnable. *Communications of the ACM*, (11):1134–1142, 1984.

[Val13] Leslie Valiant. *Probably Approximately Correct – Nature's Algorithms for Learning and Prospering in a Complex World*. 2013.