PAC Learning

francesco1093

May 2017

1 PAC learning

In this presentation we deal with the process of learning. We want to define a framework in which to define what has to be learn, how it is to be learn and when does it make sense to say that the learning has been accomplished.

- Define the learning framework - Introduce Probably Approximately Correct learning. To motivate our definitions we'll start with an example. - General theorems on concept that can be learned and sample complexity (learning guarantees for finite hypothesis sets) - Is PAC-learning an efficient definition? Stochastic learning and noise Alternative to the worst-case scenario

1.1 Introductory example

Suppose we have two people: Alice and Bob. We play the role of the latter, while Alice represents a concept we want to model (e.g. in nature)

Alice thinks an interval in R, say $I = [a, b]$. Let's denote with c the characteristic function of I. Then she picks a sample S constituted by m iid numbers $x(i) \in R$ according to a probability distribution D and computes the values $y(i) := c(x(i))$. And she gives the m pairs $(x(i), y(i))$ to Bob.

Bob has to use this information to "learn" the interval Alice thought and then use it to guess the label of future data that Alice will pick. This is of course impossible to do exactly with a finite number of data, but still he can develop an algorithm A, whose precision increases with m, namely the hypothesis $A(S)$ produced by A predicts c better as far as m increases.

1.2 Defining the setting

We will work with the following **learning scheme**: let X be a set called the instance space or the encoding of all the samples and Y be the set of labels we can assign to every instance. A function $c : X \to Y$ is called a concept and represents something we want to learn. Bob (us) considers a fixed set H of possible concepts and developes an algorithm A that given a sample S produces a concept $h \in H$ that we call *hypothesis*. We assume the sample S (of length m) to be drawn i.i.d. from a distribution D.

Remark 1 (Our example) *In our example $X = R$, $Y = 0, 1$ and the interval Bob has to guess is a concept.*

1.3 Defining learning

How to measure the distance between h and c? Obviously, with a finite set of training data, there is no algorithm able to exactly predict the interval. We should then decide what we consider a good algorithm. When can we say that he has "learned" the interval?

We want to "measure" how the concept c and our hypothesis h are "equal". If they were to be the same, we would have c(x)=h(x) for every x. So in general we want to measure how big is the set on which c(x)!=h(x). The question is now: according to which measure?

Uniform distribution? Example explaining why not, stressing the dependence on the distribution D. If some intervals are poorly represented in the distribution our algorithm cannot hope to decide clearly whether they are in the interval or not. A natural choice is then the probability measure D, that Alice uses to pick the x's.

How to minimize this distance? We aim then at minimizing the **generalization error**:

$$R_{c,D}(h_S) = P_D(h(x) \neq c(x))$$

as the dimension of the sample increases. Note that it depends on the concept to learn c, the distribution D used to pick the samples and, trough h, on the picked sample S. So, we have to decide in which sense we want this minimization to happen.

We first define PAC-learning and then motivate the definiton:

Definition 1 (PAC-learnable concept class) *A concept class is PAC-learnable if there exists an algorithm A and a polynomial function $poly(.,.,.,.)$ if for every $\epsilon > 0$, $\delta > 0$, for every distribution D on X and every concept $c \in C$, the following holds for every $m \geq poly(1/\epsilon, 1/\delta, n, size(c))$:*

$$P_{D-S^m}[R(h_S) \leq \epsilon] \geq 1 - \delta$$

If A further runs in $poly(1/\epsilon, 1/\delta, n, size(c))$, C is said to be efficiently PAC-learnable. When such an algorithm A exists, it is called a PAC-learning algorithm for C.

Remark 2 (Worst-case scenario) *A priori we have no reason to assume that we know the distribution D. Hence, we decide to minimize the err uniformly in the distribution D.*

We obviously don't know c, otherwise we wouldn't need an algorithm to find it. And we want our algorithm to conveniently learn c, whatever the choices

Alice makes. So $R_{c,D}(h)$ should also be minimized uniformly in c, for every $c \in C$.

Remark 3 (Hypothesis set) *It is in general useful to consider a hypothesis set different from C, maybe s.t. C¡H for reason of not making problems harder (we might not really know to which class the concept we want to learn belongs)*

Remark 4 (Convergence in probability) *The condition*

$$P_{D-S^m}[R(h_S) \leq \epsilon] \geq 1 - \delta$$

can be interpreted as requiring the random variables $R(h_S)$ to converge in probability to 0 as the number of samples $m \to \infty$. In this sense we require the generalization error to be small.

Remark 5 *Note that we assumed that the distribution of the training sample and the test sample used to define the data are the same. This is not a priori necessary, but it will for generalizing such a concept.*

2 Example: intervals are PAC-learnable

We will show now that intervals are a first example of PAC-learnable concept. Here's a proof

First, we define the algorithm A. Set $H = [a, b] : a, b \in R$ given a sample $S = (x(i), c(x(i)), i \in I$, define $a_1 = min\{x_i : c(x_i) = 1\}$ and $b_1 = max\{x_i : c(x_i) = 1\}$ and set $A(S) = [a_1, b_1]$.

Consider $c = [a_0, b_0] \in C = H$ and a distribution D on $X = R$. Then the error $R(h) = P_D(A = [a0, a1)B = (b1, b0])$. We can always choose a_1, b_1 s.t. $P(A' = [a_0, y)) < \epsilon/2$ and $P(B' = (y', b_0]) < \epsilon/2$. Now, $P(A' < A) \leq (1 - \epsilon/2)^m$, so it sufficies to solve $2(1 - \epsilon/2)^m \leq \delta$ for m to get for example $m \geq 2/mlog((2/\delta)$

Remark 6 *The same proof, with some modification, holds for axis-aligned rectangles and higher dimensional generalizations*

3 Classification: finite H - consistent

We want now to prove general theorem to guarantee that a given concept class is PAC-learnable.

Definition 2 (Empirical error) $\hat{R}(h) = 1/m \sum_i h(x(i)) \neq c(x(i))$

Definition 3 (Consistent algorithm) *An algorithm is consistent if for any $c \in C$ and iid sample set S produces a hypothesis h_S with empirical error $\hat{R}(h_S) = 0$.*

Theorem 1 *Let H be a finite hypothesis set (a set of functions $X \to Y$). Let A be an algorithm that for any concept $c \in H$ and i.i.d. sample S returns a consistent hypothesis h_S, i.e $\hat{R}(h_S) = 0$. Then $\forall\, \epsilon,\ \delta > 0$, the inequality*

$$P_{S-D^m}[R(h_S) <= \epsilon] >= 1 - \delta$$

holds for $m \geq 1/\epsilon(log(|H|) + log(1/\delta)$

This sample complexity result admits the following equivalent statement as a generalization bound: for any, $\epsilon > 0, \delta > 0$, with probability at least 1δ,

$$R(h_S) \leq 1/m(log(|H|) + log(1/\delta)$$

Proof. We need a uniform bound w.r.t. h on the error $R(h)$, since we do not know a priori which one is the hypothesis chosen by the algorithm.
$P(\hat{R} = 0 \wedge R(h_S) \geq \epsilon) \leq \sum_h (\hat{R} = 0 \wedge R(h) \geq \epsilon) \leq \sum_h (\hat{R} = 0 | R(h) \geq \epsilon)$ which can be bounded from above by $|H|(1-\epsilon)^m$. Solving for we get the claim.

Note that the bound on $R(h_S)$ shows that the error decreases as m increases, as we expect. The price to pay to get consistent hypothesis is to enlarge the space H. This increases the error proportionally to $log|H|$, that can be interpreted as the number of bits needed to represent H.

3.1 Example: Boolean literals

Boolean literals are variables $x_i \in 0, 1$ and their negations nx_i. A conjuction of boolean literals is for example $x_1 \wedge nx_2 \wedge x_3 \wedge \bar{}_{x_4}$ and it is represented as (in this example) a string (1,0,1,0). Suppose we want to learn the concept class C_n of the conjunction of boolean literals up to n elements.

We now want to show that this concept class is PAC-learnable. To use the previous theorem we have to compute the cardinality of the hypothesis space H, which is 3^n and to give an algorithm that is consistent with any sample data. We decide to ignore negative samples and accept all the values that are the same in every positive sample. This allows us to prove the claim, with $m \geq 1/\epsilon((log3)n + log(1/\delta)$

4 Classification: H finite - inconsistent

In the most general case, there may be no hypothesis in H consistent with the labeled training sample. This, in fact, is the typical case in practice, where the learning problems may be somewhat difficult or the concept classes more complex than the hypothesis set used by the learning algorithm.

Using Hoeffding's inequality one can prove:

Theorem 2 (Learning bound, finite hypothesis, inconsistent) *Let H be a finite hypothesis set. Then, for any $\delta > 0$, with probability at least 1δ, the following inequality holds:*

$$\forall h \in H, R(h) \leq \hat{R}(h) + \sqrt{(log|H| + log(2/\delta))/(2m)}$$

5 Criticism

There are few criticism moved to the PAC-learning approach.

5.1 Worst-case scenario

First of all, as we have seen, the PAC-learning approach deals with the worst case scenario w.r.t. to the distribution D and the concept c. In such an approach the sample complexity of a problem is represented by the number of samples m required to satisfy the bound for every concept and distribution. For this reason, in applications, where a particular choice for c and D is made, the sample complexity is expected to overestimate the real complexity of the algorithm. (Not to mention the fact that in general one only has bounds on the complexity)

For this reason PAC theory is **not good at predicting learning curves** (ratio of accuracy vs number of samples), since it usually overestimates the number of samples needed for a certain accuracy.

Possible solutions of this problem are:
1-distribution-specific models, with the backdrop of being subject to errors in modeling the distribution and on feasibility of the actual computations;

2-a Bayesian approach.

3-study of "probability of mistake", namely the probability that, after looking at the first m samples, the algorithm predicts wrongly the next sample.

5.2 Noise-free model

Another criticism to the PAC-learning is that it is a noise-free model, it does not take into account the possibility that some labels might be wrong. A solution to this problem is given by agnostic PAC-learning.

The most general setting in which we can formulate the PAC-learning is giving a distribution not only on X, but on X x Y. This allows to study more general cases, without really having to change the theory that much. This is called the stochastic scenario. If y(i) can be predicted with a function f:X-¿Y with no generalization error, we call it deterministic scenario and it suffices to give a distribution on X. (i.e. the deterministic scenario is characterized by a f,

which is the function realizing R(f), the one we hopefully want to find)

Definition 4 (Agnostic PAC-learning) *A concept class is agnostic PAC-learnable if there exists an algorithm A and a polynomial function poly(.,.,.,.) if for every $\epsilon > 0$, $\delta > 0$, for every distribution D over $X \times Y$ and every concept $c \in C$, the following holds for every $m \geq poly(1/\epsilon, 1/\delta, n, size(c))$:*

$$P_{D-S^m}[R(h_S) - min_{h \in H} R(h) \leq \epsilon] \geq 1 - \delta$$

If A further runs in $poly(1/\epsilon, 1/\delta, n, size(c))$, it is said to be an efficient agnostic PAC-learning algorithm.

5.2.1 Bayes error and noise

Definition 5 (Bayes error)

Definition 6 (Noise)