CS260: Machine Learning Theory
Lecture 2: Introduction to the PAC Model
September 28, 2011

Lecturer: Jennifer Wortman Vaughan

1 Generalizability and the PAC Model

In the last lecture, we examined our first learning model, the Consistency Model. As several students pointed out, the Consistency Model has some big flaws. First, it does not capture any notion of generalizability. Second, it does not handle any noise in the data.

We will return to the problem of noise later, but for now, let's consider the first problem: generalizability. To achieve this, it will be necessary to make some assumptions, which we discuss below. Most of these assumptions can be relaxed, and in fact, we will discuss ways to relax them in upcoming lectures.

For now, we assume that each training or test example $\vec{x}$ is generated at random from a fixed but unknown distribution $\mathcal{D}$ and that the data is *independently and identically distributed* (i.i.d.). We next assume that each label is generated by a fixed but unknown *target concept* $c \in \mathcal{C}$. In other words, we assume that the label of $\vec{x}$ is $c(\vec{x})$. (Note that we are again assuming there is no noise in the data. However, it turns out that this assumption will be easy to relax, as we'll see in upcoming lectures.) We then define the *error* of hypothesis h with respect to the target c as

$$\text{err}(h) = \text{Pr}_{\vec{x} \sim \mathcal{D}} \left[h(\vec{x}) \neq c(\vec{x}) \right].$$

Our goal will be to find a hypothesis h such that the probability that $\text{err}(h)$ is large is small. In other words, we would like to claim that h is *probably approximately correct*.

The "approximately" can be quantified through an *accuracy parameter* ϵ. In particular, since we will generally not have enough data to learn the target c perfectly, we require only that $\text{err}(h) \leq \epsilon$.

The "probably" can be quantified through a *confidence parameter* δ. We can never rule out the unlucky event that we draw an unrepresentative training set and are unable to learn a good approximation of c with the data we have. We instead require that we are able to learn a good approximation with high probability. In particular, we require that $\text{err}(h) \leq \epsilon$ with probability at least $1 - \delta$.

This leads to the following definition. (It may seem a little daunting at first, but it's actually quite natural! You will explore this definition in more detail in the first homework assignment.)

Definition 1 (Preliminary definition of PAC learning)**.** *An algorithm $\mathcal{A}$ **PAC-learns** a concept class $\mathcal{C}$ using a hypothesis class $\mathcal{H}$ if for any $c \in \mathcal{C}$, for any distribution $\mathcal{D}$ over the input space, for any $\epsilon \in (0, 1/2)$ and $\delta \in (0, 1/2)$, given access to a polynomial (in $1/\epsilon$ and $1/\delta$) number of examples drawn i.i.d. from $\mathcal{D}$ and labeled by c, $\mathcal{A}$ outputs a function $h \in \mathcal{H}$ such that with probability at least $1 - \delta$, $\text{err}(h) \leq \epsilon$.*

Note that h is chosen from a class $\mathcal{H}$ which may or may not be the same as $\mathcal{C}$. This can occasionally be useful. For example, for computational reasons, it might be more efficient to learn using a hypothesis class

All CS260 lecture notes build on the scribes' notes written by UCLA students in the Fall 2010 offering of this course. Although they have been carefully reviewed, it is entirely possible that some of them contain errors. If you spot an error, please email Jenn.

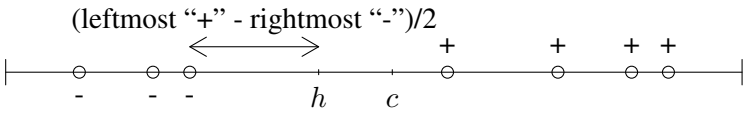

Figure 1: An illustration of the hypothesis h chosen by the algorithm $\mathcal{A}$ on a sample of points.

$\mathcal{H}$ such that $\mathcal{C} \subset \mathcal{H}$ rather than restrict our attention to only functions in $\mathcal{C}$. (Exercise: Try to show that we cannot have an algorithm that PAC-learns $\mathcal{C}$ using $\mathcal{H}$ if $C \not\subseteq \mathcal{H}$.)

We can also define what it means for a concept class to be efficiently learnable in the PAC model.

Definition 2. *A concept class $\mathcal{C}$ is **efficiently PAC-learnable** using $\mathcal{H}$ if there exists an algorithm $\mathcal{A}$ that runs in time polynomial in $1/\epsilon$ and $1/\delta$ and PAC-learns $\mathcal{C}$ using $\mathcal{H}$.*

We remark that there are other more "enhanced" definitions of PAC learning that you may come across in your reading. We will discuss these definitions in upcoming lectures.

2 PAC-Learning One-Dimensional Threshold Functions

We start with the simple example of one-dimensional threshold functions. Each threshold function is defined by a threshold value. Points greater than or equal to this threshold are labeled positive; points less than this threshold are labeled negative. For example, if each input is a person's height, we could perhaps represent the concept of what you consider tall as a threshold function.

Consider the concept class $\mathcal{C}$ of one-dimensional threshold functions on $[0, 1]$. Each function c can be represented by a scalar value, the threshold. We abuse notation and use c and h to refer to both functions in this class and their corresponding scalar threshold values.

Consider the following algorithm $\mathcal{A}$, which takes as input a set of labeled examples S: Set h to be the midpoint between the leftmost positive example (or 1, if no example is positive) and the rightmost negative example (or 0, if no example is negative), as shown in Figure 1. We will show that this algorithm PAC-learns $\mathcal{C}$ using $\mathcal{C}$ (i.e., with $\mathcal{H} = \mathcal{C}$).

In this result, and all other results we discuss for the PAC model, we will assume a model of computation in which real numbers can be stored in constant memory and in which basic operations on real numbers (addition, comparisons, etc.) take constant time.

Theorem 1. *Let $\mathcal{C}$ be the class of one-dimensional threshold functions on $[0, 1]$. For any distribution $\mathcal{D}$ over points in $[0, 1]$ and any target function $c \in \mathcal{C}$, for any $\epsilon \in (0, 1/2)$ and $\delta \in (0, 1/2)$, if $\mathcal{A}$ is run on a sample*

of m points drawn i.i.d. from $\mathcal{D}$ and labeled by c with

$$m \geq \frac{1}{\epsilon} \ln\left(\frac{2}{\delta}\right),$$

then with probability at least $1-\delta$, $\mathcal{A}$ will output a function h such that $err(h) \leq \epsilon$.

A bound of this form on m is typically referred to as the *sample complexity* of a learning problem, the number of samples required to achieve an error of at most ϵ with probability at least $1-\delta$.

Before we prove this result, we state a few useful facts.

2.1 Preliminaries: A review of some useful facts

This first theorem comes up all the time in machine learning. You may also sometimes see it written as $\ln(x) \leq x - 1$, which is equivalent for $x > 0$.

Theorem 2. *For all $x \in \mathbb{R}, 1 + x \leq e^x$.*

Exercise: Try proving this on your own. Hint: Start by taking the derivative of $e^x - 1 - x$.

This bound is very loose when $|x|$ is even moderately large, but not bad when x is near zero (try plotting this!), as it usually will be when we use it in this course.

The next result we will make use of is the Union Bound from probability theory. Recall the the union of two events, $A \cup B$, can be interpreted as an "or." We will frequently abuse notation and write $A \vee B$ in place of the more formal $A \cup B$. It is straight-forward to extend this result to unions of more than two events.

Theorem 3 (The Union Bound)**.** *For any events A and B, $Pr(A \cup B) \leq Pr(A) + Pr(B)$.*

Proof: $A \cup B$ can be disjointly decomposed as

$$(A \cap \neg B) \cup (B \cap \neg A) \cup (A \cap B).$$

Thus

$$\begin{aligned}\Pr(A \cup B) &= \Pr(A \cap \neg B) + \Pr(B \cap \neg A) + \Pr(A \cap B) \\ &\leq \underbrace{(\Pr(A \cap \neg B) + \Pr(A \cap B))}_{\Pr(A)} + \underbrace{(\Pr(\neg A \cap B) + \Pr(A \cap B))}_{\Pr(B)}.\end{aligned}$$

□

Finally, we will make use of this simple fact.

Theorem 4. *For any events A and B, if $A \implies B$ then $Pr(A) \leq Pr(B)$.*

Proof: Because $A \implies B$, we can decompose B as $A \cup (B \cap \neg A)$. Then $\Pr(A) = \Pr(B) - \Pr(B \cap \neg A) \leq \Pr(B)$. □

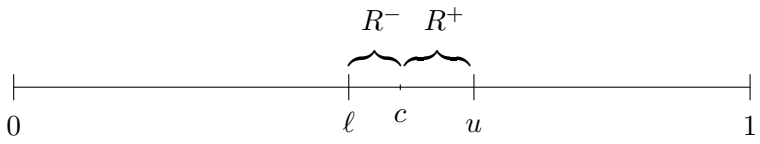

Figure 2: An illustration of the regions R^- and R^+, which both have weight at least ϵ under $\mathcal{D}$. In this example, there is a lighter density above c, so $u - c > c - \ell$ to compensate.

2.2 Proof of Theorem 1

We are now ready to prove Theorem 1.

First note that $\mathrm{err}(h) = \Pr_{x \sim \mathcal{D}}[h(x) \neq c(x)]$ is precisely the amount of weight that $\mathcal{D}$ puts on the region of $[0, 1]$ between h and c. We would like to bound the probability that this weight is larger than ϵ. That is, we would like an upper bound on $\Pr[\mathrm{err}(h) > \epsilon]$.

We begin by defining some useful notation. Let ℓ be the largest value between 0 and c such that $\mathcal{D}$ puts weight at least ϵ on the range $[\ell, c]$; call this range R^-. (Assume for now that such a value exists; below we discuss the case that it does not.) Similarly, let u be the smallest value between c and 1 such that $\mathcal{D}$ puts weight at least ϵ on the range $[c, u]$; call this range R^+. (See Figure 2.) The ranges R^- and R^+ may be different lengths depending on the shape of the distribution $\mathcal{D}$. The important part is that each range has probability at least ϵ under $\mathcal{D}$.

Our upper bound on $Pr[\mathrm{err}(h) > \epsilon]$ is based on the following observation. Suppose that the sample of m points input to the algorithm $\mathcal{A}$ contains both a point in R^- and a point in R^+. Then we know that $\mathcal{A}$ will output a threshold $h \in [\ell, u]$. It is easy to see that any threshold in this range will have error no more than ϵ with respect to c. Therefore, $\mathrm{err}(h) > \epsilon$ implies that the sample does not contain a point in at least one of these regions. Let $x_1, x_2, \cdots, x_m$ denote the sample. By Theorem 4 and the Union Bound (Theorem 3), we have

$$\begin{aligned} Pr[\mathrm{err}(h) > \epsilon] &\leq Pr\left[\left(x_1 \notin R^+ \wedge \ldots \wedge x_m \notin R^+\right) \vee \left(x_1 \notin R^- \wedge \ldots \wedge x_m \notin R^-\right)\right] \\ &\leq Pr[x_1 \notin R^+ \wedge \ldots \wedge x_m \notin R^+] + \Pr[x_1 \notin R^- \wedge \ldots \wedge x_m \notin R^-]. \end{aligned}$$

Let's first calculate the probability that no point in the sample of size m falls in R^+.

$$\begin{aligned} \Pr[x_1 \notin R^+ \wedge \ldots \wedge x_m \notin R^+] &= \prod_{i=1}^{m} \Pr(x_i \notin R^+) \\ &\leq (1 - \epsilon)^m \\ &\leq (e^{-\epsilon})^m = e^{-\epsilon m}\,. \end{aligned}$$

The first equality follows from the independence of the samples. The second inequality follows from Theorem 2. An identical argument can be used to show that

$$\Pr[x_1 \notin R^- \wedge \ldots \wedge x_m \notin R^-] \leq e^{-\epsilon m}.$$

Therefore, we have that

$$Pr[\mathrm{err}(h) > \epsilon] \leq 2e^{-\epsilon m}.$$

To prove the theorem, we need to show that when m is sufficiently large, the probability of error larger than ϵ will be no more than δ. To do this, we must find values of m such that $2e^{-\epsilon m} \leq \delta$. By simple algebraic manipulation, we see that this is true when

$$m \geq \frac{1}{\epsilon} \ln \left(\frac{2}{\delta} \right).$$

Note that the argument above only handles the case when ℓ and u are well-defined, that is, when $\mathcal{D}$ puts weight at least ϵ on the region between 0 and c, and the region between c and 1. Exercise: Complete the proof by considering the case in which this does not hold.