CS260: Machine Learning Theory
Lecture 3: PAC Bounds for Finite Concept Classes

October 3, 2011

Lecturer: Jennifer Wortman Vaughan

1 A General Bound for Finite Function Classes

Today we begin with a more general result about the PAC model. This result gives us a method of computing an upper bound on the sample complexity of learning any finite function class. Note that the restriction to finite function classes excludes the class of one-dimensional threshold functions considered in the previous lecture, but includes classes like DNFs, conjunctions, or more generally, anything with (multi-dimensional) binary input.

Theorem 1. *Consider any finite concept classes $\mathcal{C}$ and $\mathcal{H}$ for input space $\mathcal{X}$. Suppose we have an algorithm $\mathcal{A}$ that for any target function $c \in \mathcal{C}$, given a sample $x_1, \ldots, x_m \in \mathcal{X}$ labeled by c, will return a function $h \in \mathcal{H}$ consistent with this data. Then for any distribution $\mathcal{D}$ on $\mathcal{X}$, for any $c \in C$, 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*

$$m \geq \frac{1}{\epsilon}\left(\ln|\mathcal{H}| + \ln\left(\frac{1}{\delta}\right)\right)$$

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

In the proof, we will use the same set of useful facts we used in the proof from the last lecture (the union bound, the implication bound, and the upper bound on $1 + x$).

Proof: Fix any distribution $\mathcal{D}$ on $\mathcal{X}$, target $c \in \mathcal{C}$, and parameters ϵ and δ. Let h be the function that is output by $\mathcal{A}$ after receiving m random samples as input.

We will call a hypothesis $h' \in \mathcal{H}$ "bad" if $err(h') > \epsilon$. Let $B := \{h' \in \mathcal{H} : err(h') > \epsilon\}$ denote the set of bad hypotheses. We begin by making the (obvious) observation that if $err(h) > \epsilon$, then it must be the case that there exists some $h' \in B$ such that h' is consistent with our sample of m points; h itself is one

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.

example, and there could be many more. This implies that

$$\begin{aligned}
\Pr(err(h) > \epsilon) &\leq \Pr(\exists h' \in B \text{ such that } h' \text{ is consistent with the sample}) \\
&\leq \sum_{h' \in B} \Pr(h' \text{ is consistent with the sample}) && \text{(union bound)} \\
&= \sum_{h' \in B} \Pr\left(h'(x_1) = c(x_1) \wedge \ldots \wedge h'(x_m) = c(x_m)\right) && \text{(definition of consistency)} \\
&= \sum_{h' \in B} \prod_{i=1}^{m} \Pr\left(h'(x_i) = c(x_i)\right) && \text{(the sample is i.i.d.)} \\
&\leq \sum_{h' \in B} (1-\epsilon)^m && (h' \text{ is bad}) \\
&\leq |B|(1-\epsilon)^m \\
&\leq |\mathcal{H}|(1-\epsilon)^m \\
&\leq |\mathcal{H}|e^{-\epsilon m}.
\end{aligned}$$

By a similar argument to the one we used in the last lecture, we can guarantee that this value is small (no greater than δ) by choosing an appropriately large sample:

$$|\mathcal{H}|e^{-\epsilon m} \leq \delta \iff m \geq \frac{1}{\epsilon}\left(\ln|\mathcal{H}| + \ln(1/\delta)\right) .$$

$\square$

This result shows a relationship between the size of $\mathcal{H}$ and the parameters we have been examining thus far: m, ϵ, and δ. In particular, small $\mathcal{H}$ are favorable in the sense that a smaller sample size is necessary to achieve a given level of δ and ϵ. This principle is known as "Occam's Razor" – simple is better. This is fairly intuitive. If $\mathcal{H}$ is large, then it is more likely that there will be a hypothesis with high error that just happens to agree with the sample by luck. In this case, we need a large number of examples to rule out those hypotheses. In contrast, if $\mathcal{H}$ is small, then a consistent hypothesis is more likely to actually have low error.

Note that this bound does not say anything explicitly about the efficiency of $\mathcal{A}$.

2 Applying the General Learning Bound

Since the general learning bound applies when we have algorithms that return consistent hypotheses, we can use this result to investigate the algorithms that we studied under the consistency model. In particular, we can try to see what learnability of a class under the consistency model tells us about learnability of the class under the PAC model.

If we have a class that is learnable under the consistency model this means that there is an algorithm for that class that returns a consistent hypothesis if one exists. If $\mathcal{C} \subseteq \mathcal{H}$, then under our assumptions, we are guaranteed that there will always be at least one consistent hypothesis in $\mathcal{H}$. We can immediately apply this result to bound the sample complexity of the concept classes we discussed.

2.1 The Class of Monotone Conjunctions

Consider the class of monotone conjunctions. We showed earlier that this is learnable under the consistency model. For this hypothesis class we have $|\mathcal{C}| = 2^n$ when the input size is n. This gives us a sample complexity of

$$O\left(\frac{1}{\epsilon}\left(n+\ln\left(\frac{1}{\delta}\right)\right)\right).$$

Treating n as fixed (for now), this means that a polynomial (in $1/\epsilon$ and $1/\delta$) number of samples will do to get a good enough hypothesis with high enough probability. Therefore the class of monotone conjunctions with n variables is PAC learnable (using the class of monotone conjunctions with n variables). Since the algorithm we discussed for monotone conjunctions is polynomial in the number of examples and n, it is also efficiently PAC learnable.

2.2 The Class of DNFs

Consider the class of DNFs. Here we have $|\mathcal{C}| = 2^{2^n}$, so we get a sample complexity bound of

$$O\left(\frac{1}{\epsilon}\left(2^n+\ln\left(\frac{1}{\delta}\right)\right)\right).$$

We could treat n as fixed and claim that this class is PAC learnable too, but that is a little unsatisfying. It doesn't seem fair to allow the sample complexity to grow exponentially in n. To handle this concern, we make some important additions to our preliminary definition of PAC learning.

3 PAC Learning: Modified Definition

As the previous example illustrated, it would be useful to incorporate some notion of the size or dimension of the input space $\mathcal{X}$ into the definition of PAC learning. For example, if $\mathcal{X} = \{0,1\}^n$, as it does when we consider learning conjunctions or DNFs, the size of the input space can be defined naturally as n. This is also true when $\mathcal{X} = \mathbb{R}^n$. In the spirit of our computational outlook and asymptotic approach, it would be nice to be able to treat n as a parameter of the learning problem (like ϵ and δ) and require that sample complexity and running time be polynomial in n in addition to $1/\epsilon$ and $1/\delta$. Since our input space will almost always be $\{0,1\}^n$ or $\mathbb{R}^n$, we introduce a modified definition of PAC learning below that is tailored to this notion of dimension.

Going back to the example of DNFs, it may be too strong of a requirement to force that running time be polynomial in n. For some particular functions, the number of terms in a minimal DNF could be as many as 2^n. If this is true for the target function, then it would not even be possible to write down the target function in time polynomial in n. For this reason, in our modified definition, we allow the algorithm sample complexity (and time complexity, for efficient learning) polynomial in both n and the size of the true target function c. In this class we won't discuss this last point in too much detail, but you can think of size(c) as roughly the number of bits required to represent the function c. (Curious students can read more about this in Chapter 1.2.2 of Kearns & Vazirani.)

In the following definition, for every $n \geq 1$, let each $\mathcal{C}_n$ and H_n be sets of functions mapping the input space $\mathcal{X}_n$ (which will typically be $\{0,1\}^n$ or $\mathbb{R}^n$) to $\{0,1\}$. Let $\mathcal{C} = \cup_{n\geq 1}\mathcal{C}_n$, $\mathcal{H} = \cup_{n\geq 1}\mathcal{H}_n$, and $\mathcal{X} = \cup_{n\geq 1}\mathcal{X}_n$.

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

This modification is important if we want to capture the idea of generalizability in our model and rule out algorithms that simply memorize the data. If our input space is $\{0,1\}^n$ then we could always find a hypothesis with low error with $O(2^n)$ examples, but this wouldn't be a very interesting notion of "learning."

3.1 Back to DNFs

Let's return to DNFs. By the modified definition, a PAC algorithm for learning DNFs would be allowed a number of samples polynomial in n and the number of terms in the minimal DNF representation of the target c. For some particular functions, the number of terms in a minimal DNF could be as many as 2^n, in which case we would allow the algorithm time exponential in the size of an example. However, for many simpler target functions, the size of the representation is much smaller, in which case we would not allow this exponential dependence. Since the algorithm must run in polynomial time in these parameters for *every* $c \in \mathcal{C}$, the general bound for finite classes does not imply PAC learnability of DNFs.

This does not tell us that the class is *not* PAC learnable, since the bound is not a tight bound. It only shows that we cannot use the general learning bound to show PAC learnability. The question of whether this class is PAC learnable or not has been an open problem for more than two decades.