CS 880: Advanced Complexity Theory 2/8/2008

Lecture 7: Passive Learning

Instructor: Dieter van Melkebeek Scribe: Tom Watson

In the previous lectures, we studied harmonic analysis as a tool for analyzing structural properties of Boolean functions, and we developed some property testers as applications of this tool. In the next few lectures, we study applications in computational learning theory. Harmonic analysis turns out to be useful for designing and analyzing efficient learning algorithms under the uniform distribution. As a concrete example, we develop algorithms for learning decision trees. Later, we will see applications to other concept classes.

1 Computational Learning Theory

Recall that in the setting of property testing, we have an underlying class of Boolean functions and oracle access to a Boolean function f, and we want a test that accepts if f is in the class and rejects with high probability if f is far from being in the class. In computational learning theory, we again have an underlying class and a function c, but now we are guaranteed that c is in the class, and we wish to know which function c is. More formally, we have the following ingredients.

- A *concept class* $\{\mathcal{C}_{n,s}\}$ where $\mathcal{C}_{n,s} \subseteq \big\{c : \{-1,1\}^n \to \{-1,1\}\big\}$. The elements of $\mathcal{C}_{n,s}$ are called *concepts*. The parameter n is the input length of the concepts, and the parameter s is a measure of the complexity of the concepts. For example, $\mathcal{C}_{n,s}$ might be the class of n-variable functions computable by decision trees of size at most s.
- A *hypothesis class* $\{\mathcal{H}_{n,s}\}$ where $\mathcal{H}_{n,s} \subseteq \big\{h : \{-1,1\}^n \to \{-1,1\}\big\}$. Given a concept $c \in \mathcal{C}_{n,s}$, a learning algorithm will output a *hypothesis* $h \in \mathcal{H}_{n,s}$, which is its guess for what c is. Ideally we would like $\mathcal{H}_{n,s} = \mathcal{C}_{n,s}$ (this setting is called *proper learning*). However, in general we have $\mathcal{H}_{n,s} \supseteq \mathcal{C}_{n,s}$; i.e., a learning algorithm may be allowed to output a hypothesis that is not a concept.
- A *learning process*. There are several ways in which a learning algorithm may have access to the concept c.
  - *Labeled samples* $(x, c(x))$ where x is drawn from some distribution D on $\{-1,1\}^n$.
  - *Membership queries*, i.e., oracle access to c.

  Note that these two options are incomparable; the former cannot simulate the latter because the algorithm has no control over the samples it receives, and the latter cannot simulate the former if the distribution D is not known. There is a third commonly studied type of access known as equivalence queries, but we will not study this type.

  Learning from labeled samples only is often referred to as "passive learning," whereas learning using membership queries is referred to as "active learning."

- A *measure of success*. We measure success in terms of h being close to c. We would like $\Pr_{x \leftarrow D}[h(x) \neq c(x)] \leq \epsilon$ for some *error parameter* ϵ; note that the probability is with respect to the same distribution D that the labeled samples are drawn from. As learning algorithms are generally randomized, the hypothesis h may or may not satisfy the above inequality, but we would like it to hold with high probability, say at least $1 - \delta$ where δ is a *confidence parameter*.

We use the running time of a learning algorithm as our measure of efficiency. We would like the running time to be polynomial in n, s, $\frac{1}{\epsilon}$, and $\frac{1}{\delta}$. Typically, however, the dependence on δ is polynomial in $\log \frac{1}{\delta}$ rather than in $\frac{1}{\delta}$. This is because if we can learn with some particular confidence parameter less than $1/3$, say, then we can iterate the process to drive the confidence parameter down exponentially in the number of iterations. In each iteration we find a new hypothesis h and test whether it is sufficiently close to c using labeled samples (incurring only a slight loss in confidence); if so we output h and otherwise we continue. For this reason, we will usually ignore the dependence on the confidence parameter δ.

These ingredients allow us to define the model of *distribution-free learning* or *PAC (probably approximately correct) learning*.

Definition 1. *A* PAC learner for $\{\mathcal{C}_{n,s}\}$ by $\{\mathcal{H}_{n,s}\}$ *is a randomized oracle Turing machine L such that for all n, s, $\epsilon > 0$, $\delta > 0$, distributions D on $\{-1,1\}^n$, and concepts $c \in \mathcal{C}_{n,s}$, if L is given access to random labeled samples $(x, c(x))$ where x is drawn from D, then it produces a hypothesis $h \in \mathcal{H}_{n,s}$ such that*

$$\Pr\left[\Pr_{x \leftarrow D}[h(x) \neq c(x)] \leq \epsilon\right] \geq 1 - \delta. \tag{1}$$

We say that L is efficient *if it runs in time poly$(n, s, \frac{1}{\epsilon}, \frac{1}{\delta})$. A* PAC learner with membership queries *is defined in the same way, except that the learner L is allowed to make oracle queries to c in addition to receiving random labeled samples.*

Note that the outer probability in equation (1) is over the randomness used by L and the randomness used to generate labeled samples, and the inner probability is an independent experiment for measuring the quality of h. Also note that L does not know D, but is still required to succeed for any D.

In this class, we will only study learning algorithms that work when the random samples are guaranteed to come from the uniform distribution (and may fail when D is not uniform).

Definition 2. *We say a machine L* learns with respect to the uniform distribution *if L satisfies the conditions of Definition 1 for D the uniform distribution.*

In this class, we will also assume that the learner L is given both n and s as input. In practical settings, the complexity s of the concept may not be known. In this case, the learner can just try $s = 2^i$, $i = 0, 1, 2, \ldots$ until it finds a hypothesis sufficiently close to the concept (which it can test using random samples); this incurs only a small penalty in running time. Thus, our assumption about knowing s is essentially without loss of generality.

2 Learning Using Harmonic Analysis

It is intuitively clear that if $\mathcal{C}_{n,s}$ has no structure, for example is the set of all n-variable functions, then efficient learning is impossible. This is because after seeing a small set of samples, the learning

algorithm has no way of guessing the value of the concept on other inputs. (This intuition can be turned into a formal argument.)

The structure we impose on $\mathcal{C}_{n,s}$ in this lecture is that all concepts must have their Fourier weight concentrated on only a few coefficients. That is, we only allow concepts $c = \sum_S \hat{c}(S)\chi_S$ such that for some small set $\mathcal{S} \subseteq 2^{[n]}$, $\sum_{S \notin \mathcal{S}} \left(\hat{c}(S)\right)^2 \leq \epsilon$. (We are using ϵ for both this and the error parameter of a learning algorithm; however, these two quantities will be closely related so we do not introduce new notation.)

How does this structural property help us learn c? We know that the (in general non-Boolean) function $f = \sum_{S \in \mathcal{S}} \hat{c}(S)\chi_S$ is a good approximation to c in the sense that $||c - f||^2 \leq \epsilon$. Thus, as a first step, we can focus on finding an approximation to f. Assuming we know $\mathcal{S}$, we need to approximate $\hat{f}(S) = \hat{c}(S) = E_x[c(x)\chi_S(x)]$ for $S \in \mathcal{S}$ (where the expectation is over the uniform distribution). If we have access to uniform samples $(x, c(x))$, then we can also get uniform samples $(x, c(x)\chi_S(x))$ because $\chi_S(x)$ is trivially efficiently computable. We can thus approximate $E_x[c(x)\chi_S(x)]$ by taking the average of many samples of $c(x)\chi_S(x)$.

More formally, since $|c(x)\chi_S(x)| \leq 1$ for all x, a Chernoff bound shows that with probability at least $1 - \exp\left(\frac{-\eta^2 k}{2}\right)$, the average of k samples is within an additive $\pm\eta$ of the desired expectation, where η is a parameter we will specify shortly. We want this probability to be at least $1 - \frac{\delta}{|\mathcal{S}|}$. Picking $k = \Theta(\frac{1}{\eta^2} \log \frac{|\mathcal{S}|}{\delta})$ suffices for this.

If we let $\hat{g}(S)$ denote our approximation of $\hat{f}(S)$ for $S \in \mathcal{S}$, then the function $g = \sum_{S \in \mathcal{S}} \hat{g}(S)\chi_S$ satisfies

$$||c - g||^2 \;=\; \sum_{S \in \mathcal{S}} \left(\hat{f}(S) - \hat{g}(S)\right)^2 + \sum_{S \notin \mathcal{S}} \left(\hat{c}(S)\right)^2 \;\leq\; |\mathcal{S}| \cdot \eta^2 + \epsilon \;\leq\; 2\epsilon,$$

where the last inequality holds provided we pick $\eta \leq \sqrt{\frac{\epsilon}{|\mathcal{S}|}}$.

Thus, assuming we know $\mathcal{S}$ and our labeled samples are drawn from the uniform distribution, we can find g in time $poly(n, |\mathcal{S}|, \frac{1}{\epsilon}, \log \frac{1}{\delta})$: we need to process $|\mathcal{S}|$ coefficients, for each one we need $O(\frac{1}{\eta^2} \log \frac{|\mathcal{S}|}{\delta}) = O(\frac{|\mathcal{S}|}{\epsilon} \log \frac{|\mathcal{S}|}{\delta})$ samples, and for each sample we need $O(n)$ time. After a union bound over $S \in \mathcal{S}$, with probability at least $1 - \delta$, we have $||c - g||^2 \leq 2\epsilon$.

The only problem with g is that it might not be Boolean. The most natural thing to do is to obtain hypothesis h by rounding g to the nearest Boolean function: $h(x) = sign(g(x))$ (the case $g(x) = 0$ can be decided arbitrarily). How well does h approximate c? Assuming we achieved $||c - g||^2 \leq 2\epsilon$, we have

$$\Pr[h(x) \neq c(x)] \leq \Pr[|c(x) - g(x)| \geq 1] \leq E[(c - g)^2] \leq 2\epsilon.$$

We won't care exactly what hypothesis class $\{\mathcal{H}_{n,s}\}$ we are using, although it does follow from the above that $\sum_{S \notin \mathcal{S}} \left(\hat{h}(S)\right)^2 \leq 2\epsilon$. This is because $\sum_{S \notin \mathcal{S}} \left(\hat{h}(S)\right)^2 \leq \sum_S \left(\hat{h}(S) - \hat{g}(S)\right)^2 = E[(h - g)^2]$ and over all Boolean functions $\tilde{c}$, h minimizes $E[(\tilde{c} - g)^2]$ but c achieves $E[(c - g)^2] \leq 2\epsilon$.

To summarize where we are, we have four functions involved: Given the concept c, f is obtained by annhilating the Fourier coefficients not in $\mathcal{S}$, g is obtained by approximating the remaining Fourier coefficients, and then h is obtained by rounding to a Boolean function. We argued that by choosing the approximation parameter appropriately, we achieve at most 2ϵ error with probability at least $1-\delta$ in time $poly(n, |\mathcal{S}|, \frac{1}{\epsilon}, \log \frac{1}{\delta})$. This result requires the samples to come from the uniform distribution.

The above learning algorithm needs to somehow know a set $\mathcal{S}$ such that $\sum_{S \notin \mathcal{S}} \big(\hat{c}(S)\big)^2 \leq \epsilon$. There are two cases:

- $\mathcal{S}$ is known; i.e., it can be computed efficiently given n and s. In this case, as we showed above, we can learn efficiently using only random samples, provided $\mathcal{S}$ is not too large.
- $\mathcal{S}$ is unknown. In this case, provided we have a good upper bound on $|\mathcal{S}|$, we can compute a suitable alternative set $\mathcal{S}'$ using membership queries and then run the above algorithm with $\mathcal{S}'$. We show how to do this in the next lecture.

Note that a randomized learning algorithm that can make membership queries automatically has access to labeled samples from the uniform distribution.

3 Learning From Samples

For the rest of this lecture, we focus on special concept classes where an appropriate set $\mathcal{S}$ is trivial to find. We begin by noting that the set of all small sets S forms an appropriate $\mathcal{S}$ when the average sensitivity $I(c)$ is low, because if a large set has a large Fourier coefficient, then $I(c)$ must also be large because of the formula $I(c) = \sum_S |S| \big(\hat{c}(S)\big)^2$, proven in the previous lecture.

Theorem 1. *If $I(c) \leq d$ for all $c \in \mathcal{C}_{n,s}$ then there is an algorithm that learns $\mathcal{C}_{n,s}$ with respect to the uniform distribution using random samples only and running in time poly$(n^{d/\epsilon}, \log \frac{1}{\delta})$.*

Proof. For all t we have

$$\sum_{|S| \geq t} \big(\hat{c}(S)\big)^2 \;\leq\; \frac{1}{t} \sum_{|S| \geq t} |S| \big(\hat{c}(S)\big)^2 \;\leq\; \frac{1}{t} I(c) \;\leq\; \frac{d}{t} \;\leq\; \epsilon,$$

where the last inequality holds provided $t \geq \frac{d}{\epsilon}$. Thus we can apply the learning algorithm from the previous section with $\mathcal{S} = \{S : |S| < \frac{d}{\epsilon}\}$. A trivial upper bound of $|\mathcal{S}| \leq (n+1)^{d/\epsilon}$ yields the running time. □

We can translate this result to decision trees using the fact that functions computable by small decision trees have low average sensitivity. If a decision tree has depth d then the average sensitivity of the function computed by it is at most d; in fact, the sensitivity at any input is at most the number of nodes on the corresponding path in the decision tree, which is at most d. Thus we have the following.

Theorem 2. *If $\mathcal{C}_{n,d}$ is the set of functions computed by decision trees of depth at most d, then there is an algorithm that learns $\mathcal{C}_{n,d}$ with respect to the uniform distribution using random samples only and running in time poly$(n^{d/\epsilon}, \log \frac{1}{\delta})$.*

What can we say if s refers to the size of a decision tree, not its depth? Suppose we truncate the original tree T at some level ℓ to obtain tree T'. (The behavior of T' on severed paths is arbitrary.) For large ℓ, T' is a good approximation to T because each of the leaves at depth greater than ℓ is reached with only small probability, and there are few such leaves when T is small. How large does ℓ need to be? We have

$$\Pr_x[T'(x) \neq T(x)] \;\leq \sum_{\substack{\text{leaves } v \\ \text{depth} > \ell}} \Pr_x[v \text{ reached}] \;<\; \sum_{\substack{\text{leaves } v \\ \text{depth} > \ell}} 2^{-\ell} \;\leq\; s \cdot 2^{-\ell} \;\leq\; \epsilon, \tag{2}$$

where the last inequality holds provided $\ell \geq \log \frac{s}{\epsilon}$.

Now, suppose we apply our learning algorithm for decision trees of depth at most ℓ to T. The approximation f in the first step satisfies

$$\begin{aligned} ||T - f|| &= ||\sum_{S \notin \mathcal{S}} \hat{T}(S)\chi_S|| && (3) \\ &\leq ||\sum_{S \notin \mathcal{S}} (\hat{T}(S) - \hat{T}'(S))\chi_S|| + ||\sum_{S \notin \mathcal{S}} \hat{T}'(S)\chi_S|| && (4) \\ &\leq ||T - T'|| + ||\sum_{S \notin \mathcal{S}} \hat{T}'(S)\chi_S|| && (5) \\ &\leq 2\sqrt{\epsilon} + \sqrt{\epsilon} = 3\sqrt{\epsilon}, && (6) \end{aligned}$$

Step (3) follows because the Fourier coefficients of f agree with those of T on $\mathcal{S}$ and vanish elsewhere. Step (4) follows from the triangle inequality, and step (5) from the orthonormality of the characters. Step (6) follows from the fact that $||T - T'|| = 2\sqrt{\Pr[T \neq T']}$ and (2), and from the defining property of $\mathcal{S}$. Thus, $||T - f||^2 \leq 9\epsilon$. By approximating each of the Fourier coefficients of T in $\mathcal{S}$ to within η as above, we obtain an approximation g for which $||T - g||^2 \leq 10\epsilon$, and our resulting hypothesis $h = sign(g)$ differs from T on no more than a fraction 10ϵ of the inputs. We have derived the following result about learning decision trees from random samples.

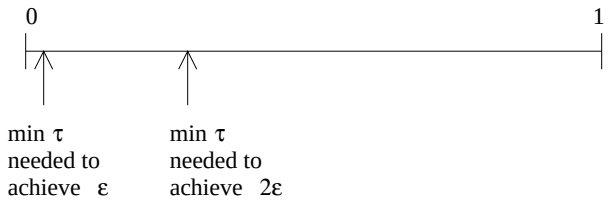
Theorem 3. *If $\mathcal{C}_{n,s}$ is the set of functions computed by decision trees of size at most s, then there is an algorithm that learns $\mathcal{C}_{n,s}$ with respect to the uniform distribution using random samples only and running in time $poly(n^{(\log \frac{s}{\epsilon})/\epsilon}, \log \frac{1}{\delta})$.*

In particular, if ϵ is a constant then we get a learning algorithm that runs in time quasipolynomial in $n + s$. As we begin to show next, we can bring the running time down to polynomial if membership queries are available.

4 Learning From Queries

Let us return to our generic learning algorithm that uses random samples from the uniform distribution and achieves 2ϵ error and δ confidence in time $poly(n, |\mathcal{S}|, \frac{1}{\epsilon}, \log \frac{1}{\delta})$ provided we know a set $\mathcal{S}$ such that $\sum_{S \notin \mathcal{S}} \left(\hat{c}(S)\right)^2 \leq \epsilon$. If all we know is that such a set $\mathcal{S}$ exists and has size at most some value M, then we can attempt to find $\mathcal{S}$ or another suitable set.

The idea is to define $\mathcal{S}' = \{S : \left(\hat{c}(S)\right)^2 \geq \tau\}$ for some threshold τ and compute $\mathcal{S}'$ using the list decoding algorithm we describe in the next lecture. Without loss of generality we can assume that $\mathcal{S}$ contains the M sets with the largest coefficients in absolute value. A first attempt at choosing τ would be to set it low enough to ensure that $\sum_{S \notin \mathcal{S}'} \left(\hat{c}(S)\right)^2 \leq \epsilon$. Setting $\tau = \frac{\epsilon}{2^n}$ would certainly achieve this, since then $\sum_{S \notin \mathcal{S}'} \left(\hat{c}(S)\right)^2 \leq 2^n \cdot \tau = \epsilon$. However, since the list decoding algorithm runs in time polynomial in τ^{-1}, this would result in an exponential-time learning algorithm. By setting τ higher, we may not be able to guarantee our ϵ bound, but we won't be too far off, since of the sets not in $\mathcal{S}'$, those not in $\mathcal{S}$ only contribute a small amount (by hypothesis), and those in $\mathcal{S}$ also only contribute a small amount (since there are only a small number M of them and each has weight at most τ).



Formally,

$$\sum_{S \notin \mathcal{S}'} \left(\hat{c}(S)\right)^2 \leq \sum_{S \in \mathcal{S} \setminus \mathcal{S}'} \left(\hat{c}(S)\right)^2 + \sum_{S \notin \mathcal{S}} \left(\hat{c}(S)\right)^2 \leq M \cdot \tau + \epsilon.$$

Choosing $\tau = \frac{\epsilon}{M}$ ensures that this quantity is at most 2ϵ, and it ensures that $|\mathcal{S}'| \leq \frac{M}{\epsilon}$, so $\mathcal{S}'$ is a suitable alternative for $\mathcal{S}$. That is, we can learn in roughly the same time as if we knew $\mathcal{S}$, provided we can find $\mathcal{S}'$. But $\left(\hat{c}(S)\right)^2 \geq \frac{\epsilon}{M}$ is equivalent to saying that χ_S or $\overline{\chi_S}$ correlates well with c, and since the characters are exactly the Hadamard codewords, we can use a list decoding algorithm for the Hadamard code to efficiently find the set $\mathcal{S}'$ using membership queries.

In the next lecture, we will spell out the list decoding algorithm in more detail, and we will apply this approach to learning decision trees.