CS 880: Advanced Complexity Theory 2/11/2008

Lecture 8: Active Learning

Instructor: Dieter van Melkebeek Scribe: Chi Man Liu

Last time we studied computational learning theory and saw how harmonic analysis could be used to design and analyze efficient learning algorithms with respect to the uniform distribution. We developed a generic passive learning algorithm for concepts whose Fourier spectrum is concentrated on a known set, and applied it to decision trees. We also started developing an approach for the case where the concentration set is not known but we can use membership queries, i.e., and active learning algorithm. In this lecture, we continue that approach and use it to devise a polynomial-time algorithm for learning decision trees using membership queries.

1 Learning with Membership Queries

We return to our general setting of learning concept classes with concentrated Fourier spectra. Let $C_{n,s}$ be a concept class with a concentrated Fourier spectrum, i.e., for each $c \in \mathcal{C}_{n,s}$, there exists some small set $\mathcal{S} \subseteq 2^{[n]}$ satisfying $\sum_{S \notin \mathcal{S}} (\hat{c}(S))^2 \leq \epsilon$. Last time we showed that if we know $\mathcal{S}$, we can learn $C_{n,s}$ in time $\text{poly}(n, |\mathcal{S}|, \frac{1}{\epsilon}, \log \frac{1}{\delta})$ from random samples. We now give a learning algorithm for the case where $\mathcal{S}$ is unknown and we can use membership queries. Suppose that we know a value M such that $|\mathcal{S}| \leq M$. We are going to use an alternative for $\mathcal{S}$ which we can efficiently compute. Define

$$\mathcal{S}' = \{S \subseteq [n] : (\hat{c}(S))^2 \geq \tau\},$$

where $\tau = \frac{\epsilon}{M}$. Last time we argued that

$$\sum_{S \notin \mathcal{S}'} (\hat{c}(S))^2 \leq 2\epsilon.$$

Since $|\mathcal{S}'| \leq \frac{M}{\epsilon}$, $\mathcal{S}'$ is a suitable alternative for $\mathcal{S}$. The next step is to approximate $\mathcal{S}'$. The algorithm uses a procedure which is essentially the same as the list decoding algorithm for the Hadamard code. We formally state the algorithm.

Lemma 1. *There exists an algorithm that finds, in time* $\text{poly}(n, \frac{1}{\tau}, \log \frac{1}{\delta})$*, a set* $\mathcal{S}'' \subseteq 2^{[n]}$ *such that with probability at least* $1 - \delta$*,*

$$\{S \subseteq [n] : (\hat{c}(S))^2 \geq \tau\} \subseteq \mathcal{S}'' \subseteq \{S \subseteq [n] : (\hat{c}(S))^2 \geq \frac{\tau}{2}\}. \tag{1}$$

Before we prove Lemma 1, let us see why it suffices to find such a set $\mathcal{S}''$. The left inclusion ensures that $\mathcal{S}''$ is good enough for our application — its weight is no smaller than that of $\mathcal{S}'$, so the weight outside $\mathcal{S}''$ is again at most 2ϵ. The right inclusion bounds the number of Fourier coefficients we need to consider. By Parseval's equality, the rightmost set has size at most $\frac{2}{\tau}$.

Proof of Lemma 1. We think of each set $S \subseteq [n]$ as a binary string of length n. To be precise, we view it as the characteristic vector of the set S over $[n]$. Our algorithm grows a binary tree level by level where each node is associated with a string v, which is the concatenation of the edge labels

along the path from the root to the node. v represents the set of subsets of $[n]$ whose prefix is v. More formally, we let $\mathcal{S}_v = \{S \subseteq [n] : S \cap \{1, 2, \ldots, |v|\} = v\}$. We define the weight of v as $\sum_{S \in \mathcal{S}_v} (\hat{c}(S))^2$. Observe that if the weight of v is less than τ, then every node under v must have weight less than τ. Hence we stop growing from a node (and remove that node as well) once its weight is less than τ. We count the number of nodes in the tree. Since the nodes at the same level represent disjoint subsets of $2^{[n]}$, and that each node has weight at least τ, it follows that each level has at most $\frac{1}{\tau}$ nodes. There are at most n levels, therefore the total number of nodes cannot exceed $\frac{n}{\tau}$.

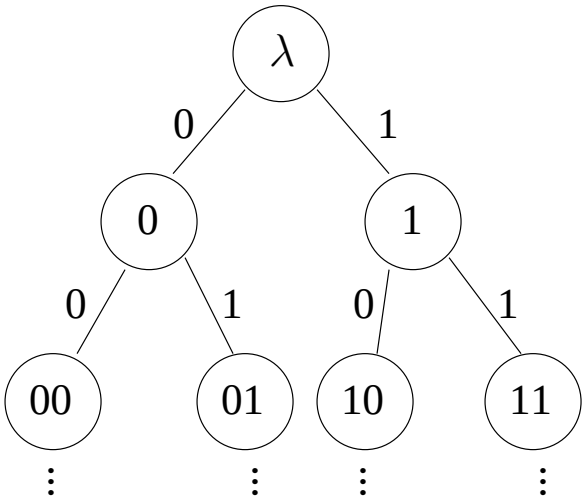

Figure 1: Growing the binary tree.

For each node, we need to compare its weight with τ. We don't know how to do this exactly, because the nodes near the root represent very large subsets of $2^{[n]}$. However, if we allow small errors, we can do it efficiently with high probability. In particular, we want to compute the weight with error at most $\frac{\tau}{4}$ with high probability. We put in $\mathcal{S}''$ only the leaves (or equivalently, subsets of $2^{[n]}$) with approximated weight at least $\frac{3\tau}{4}$. If a leaf has weight at least τ, its corresponding subset (which is in the left set in Equation (1)) is likely to be included in $\mathcal{S}''$; if its weight is less than $\frac{\tau}{2}$, then its corresponding subset (which is *not* in the right set in Equation (1)) is not likely to be included in $\mathcal{S}''$. In conclusion, Equation (1) holds with high probability.

We now give the details of the above procedure for approximating the weight of a node v. Let $k = |v|$ be the level which the node is in. Then the weight of v can be written as

$$\begin{aligned}
\sum_{S \cap [k] = v} (\hat{c}(S))^2 &= \sum_{T \subseteq \{k+1, \ldots, n\}} (\hat{c}(v \cup T))^2 && (2)\\
&= \sum_{T \subseteq \{k+1, \ldots, n\}} \mathrm{E}_x[c(x)\chi_{v \cup T}(x)]\mathrm{E}_y[c(y)\chi_{v \cup T}(y)] && (3)\\
&= \sum_{T \subseteq \{k+1, \ldots, n\}} \mathrm{E}_{x,y}[c(x)\chi_{v \cup T}(x)c(y)\chi_{v \cup T}(y)]. && (4)
\end{aligned}$$

The second equality follows from that $\hat{c}(S) = \mathrm{E}_x[c(x)\chi_S(x)]$ for all $S \subseteq 2^{[n]}$. Although each of the expectations in Equation (4) can be approximated with high probability by taking random samples

and bounding the error with Chernoff's bound, approximating the sum of 2^{n-k} such expectations would require 2^{n-k} times as many samples to guarantee the same error bound. Therefore, our goal is to get rid of the large multiplicative factor.

We write x as the concatenation of two strings, namely $x = x_1x_2$ where $|x_1| = k$. Similarly, we write $y = y_1y_2$ where $|y_1| = k$. Then, the above quantity becomes

$$\mathrm{E}_{x_1,x_2,y_1,y_2}[c(x_1x_2)\chi_v(x_1)c(y_1y_2)\chi_v(y_1)\sum_{T\subseteq\{k+1,\ldots,n\}}\chi_T(x_2)\chi_T(y_2)]. \tag{5}$$

Note that $\chi_T(x_2)$ and $\chi_T(y_2)$ are the same as $\chi_{x_2}(T)$ and $\chi_{y_2}(T)$, respectively. So we get

$$\sum_{T\subseteq\{k+1,\ldots,n\}}\chi_T(x_2)\chi_T(y_2) = \sum_{T\subseteq\{k+1,\ldots,n\}}\chi_{x_2}(T)\chi_{y_2}(T).$$

Moreover, $\sum_T \chi_{x_2}(T)\chi_{y_2}(T) = 2^{n-k}\cdot \mathrm{E}_T[\chi_{x_2}(T)\chi_{y_2}(T)] = 2^{n-k}\cdot\langle\chi_{x_2},\chi_{y_2}\rangle$. Since χ_{x_2} and χ_{y_2} are characters, their inner product is 1 if $x_2 = y_2$ and 0 otherwise. Hence, $\sum_T \chi_{x_2}(T)\chi_{y_2}(T) = 2^{n-k}\cdot\delta_{x_2,y_2}$. Equation (5) becomes

$$2^{n-k}\cdot\mathrm{E}_{x_1,x_2,y_1,y_2}[c(x_1x_2)\chi_v(x_1)c(y_1y_2)\chi_v(y_1)\delta_{x_2,y_2}]. \tag{6}$$

Note that the expression inside the expectation vanishes if $x_2 \neq y_2$. By expanding the expectation over y_2, we can rewrite Equation (6) as

$$2^{n-k}\cdot\left(\mathrm{E}_{x_1,x_2,y_1}\left[\frac{1}{2^{n-k}}\cdot\sum_{y_2}c(x_1x_2)\chi_v(x_1)c(y_1y_2)\chi_v(y_1)\delta_{x_2,y_2}\right]\right) \tag{7}$$

$$= 2^{n-k}\cdot\left(\mathrm{E}_{x_1,x_2,y_1}\left[\frac{1}{2^{n-k}}\cdot c(x_1x_2)\chi_v(x_1)c(y_1x_2)\chi_v(y_1)\right]\right) \tag{8}$$

$$= \mathrm{E}_{x_1,y_1,z}[c(x_1z)\chi_v(x_1)c(y_1z)\chi_v(y_1)]. \tag{9}$$

We have finally got rid of the factor of 2^{n-k}. The expression inside the expectation in Equation (9) is easy to evaluate given v, x_1, y_1 and z. To approximate Equation (9) within $\frac{\tau}{4}$ with high probability, the number of samples needed is roughly $O((\frac{1}{\tau})^2)$ using Chernoff's bound. □

Note that in Equation (9), $c(x_1z)$ and $c(y_1z)$ are correlated, so they are not exactly random samples from c. This is where membership queries come in.

Combining all the above gives us the following theorem.

Theorem 1. *Let $\mathcal{C}_{n,s}$ be a concept class such that for some value M, for all $c \in \mathcal{C}_{n,s}$, there exists a set $\mathcal{S} \subseteq 2^{[n]}$ with $|\mathcal{S}| \leq M$ satisfying $\sum_{S\notin\mathcal{S}}(\hat{c}(S))^2 \leq \epsilon$. Then there exists an algorithm that learns $\mathcal{C}_{n,s}$ using membership queries and running in time* $\mathrm{poly}(n, M, \frac{1}{\epsilon}, \log\frac{1}{\delta})$.

The algorithm stated in Theorem 1 assumes the knowledge of an upper bound M. When we do not know such an M, picking $M = 2^n$ would certainly work, but resulting in an exponential running time. To find a small M that works, we can start from $M = 1$ and repeatedly run the algorithm, each time doubling the value of M until the algorithm succeeds in learning the concept.

Recall that in order for our algorithm to work, the Fourier spectrum of the concept we want to learn has to be concentrated. If we view $\hat{c}$ as a 2^n-vector (whose 2-norm is 1), then by Cauchy-Schwarz's inequality, its 1-norm is at most $2^{n/2}$. The 1-norm achieves maximum value if and only

if all the Fourier coefficients have the same magnitude, i.e., the Fourier weight is far from being concentrated. Intuitively, we would expect that if the 1-norm of $\hat{c}$ is far from its maximum value, then c has concentrated Fourier weight. We formalize this intuition in the following corollary.

Corollary 1. *If every $f \in \mathcal{C}_{n,s}$ satisfies $\sum_{S\subseteq[n]} |\hat{c}(S)| \leq d$ for some constant d, then we can learn $\mathcal{C}_{n,s}$ in time* $\mathrm{poly}(n, \frac{d}{\epsilon}, \log\frac{1}{\delta})$ *with membership queries.*

Proof. Let $\mathcal{S} = \{S \subseteq [n] : (\hat{c}(S))^2 \geq \tau\}$. Then

$$\sum_{S\notin\mathcal{S}} (\hat{c}(S))^2 \leq \left(\max_{S\notin\mathcal{S}} |\hat{c}(S)|\right) \sum_{S\notin\mathcal{S}} |\hat{c}(S)| \leq \sqrt{\tau}\cdot d \leq \epsilon,$$

where the last inequality holds provided that $\tau \leq (\epsilon/d)^2$. The corollary then follows from Theorem 1. □

2 Learning Decision Trees with Membership Queries

As an application of Corollary 1, we give a polynomial-time algorithm for learning decision trees. We first analyze the spectrum of decision trees. Let c be a function computed by a decision tree. Then

$$c = \sum_{\text{leaves } v} c(v)\chi[\text{path to } v](x), \tag{10}$$

where $\chi[\text{path to } v]$ is the function defined by $\chi[\text{path to } v](x) = 1$ if x leads to v from the root and 0 otherwise. This function can be rewritten as

$$\chi[\text{path to } v](x) = \prod_{i\in V} \left(\frac{1 \pm x_i}{2}\right), \tag{11}$$

where V denotes the set of all variables along the path from the root to v. The sign depends on whether x_i is 1 or -1 along the path: if $i = -1$, we take the sign to be minus, so $(1 - x_i)/2 = 1$ if $x_i = -1$ and 0 otherwise; if $x = 1$, we take the sign to be plus, so $(1 + x_i)/2 = 1$ if $x_i = 1$ and 0 otherwise. We can also rewrite Equation (11) using characters:

$$\chi[\text{path to } v](x) = \sum_{S\subseteq V} \frac{\pm 1}{2^\ell}\chi_S(x), \tag{12}$$

where ℓ is the length of the path from the root to v and the sign depends on S's values along the path.

From Equation (12) we see some properties of the Fourier spectrum of decision trees. Suppose that the decision tree computing c has depth d. Then

1. $\sum_{|S|>d} (\hat{c}(S))^2 = 0$. This is because in Equation (12), the coefficient of $\chi_S(x)$ is nonzero only if $|S| \leq \ell \leq d$.

2. Each $\hat{c}(S)$ is an integer multiple of $\frac{1}{2^d}$. This is obvious from Equation (12).

3. The number of nonzero Fourier coefficients is at most 4^d. This is because there are at most 2^d leaves and each leaf gives at most 2^d nonzero Fourier coefficients.

By property 1, we can pick $\mathcal{S} = \{S \subseteq [n] : |S| \leq d\}$. It is clear that $|\mathcal{S}| \leq (n+1)^d$, and so using the results from last time, we get an algorithm for learning decision trees of depth at most d using only random samples, running in time $\text{poly}(n^d, \frac{1}{\epsilon}, \log \frac{1}{\delta})$. This is an improvement over $\text{poly}(n^{d/\epsilon}, \log \frac{1}{\delta})$ obtained last time. This improvement comes from a better analysis of the Fourier spectrum of decision trees. Property 2 allows us to compute the Fourier coefficients exactly with high probability. By exploiting this property, we can eliminate the error in our algorithm. In other words, we can learn decision trees exactly in time $\text{poly}(n^d, \log \frac{1}{\delta})$ using only random samples from the uniform distribution.

Combining property 3 and Theorem 1 (and eliminating the error) gives us the following result.

Theorem 2. *Let $C_{n,d}$ be the set of functions computed by decision trees of depth at most d. Then there exists an algorithm that learns $C_{n,d}$ exactly with membership queries in time* $\text{poly}(n, 4^d, \log \frac{1}{\delta})$.

A direct consequence of Theorem 2 is that we can learn log-depth decision trees exactly in polynomial time with membership queries.

Now we turn our attention to decision trees with a specific size, say s. Mimicking the analysis in the previous lecture, we can approximate a tree of size n with a truncated tree of depth $\log \frac{s}{\epsilon}$, while introducing an approximation error of at most ϵ. This leads to a polynomial-time algorithm for learning decision trees with membership queries.

Theorem 3. *Let $C_{n,s}$ be the set of functions computed by decision trees of size at most s. Then there exists an algorithm that learns $C_{n,s}$ with membership queries in time* $\text{poly}(n, s, \frac{1}{\epsilon}, \log \frac{1}{\delta})$.

Alternatively, Theorem 3 can be obtained from Corollary 1 since we showed that for decision trees of size s, $\sum_{S \subseteq [n]} |\hat{c}(S)| \leq s$. We can then apply using membership queries that also runs in time $\text{poly}(n, \frac{s}{\epsilon}, \log \frac{1}{\delta})$.

Finally, we mention that the algorithms in Theorems 2 and 3 can be derandomized using small-bias sample space, which will be covered later in the course.

3 Next Time

In the next lecture, we will discuss applications other than decision trees. In particular, we will design learning algorithms for DNFs and constant-depth circuits based on the Switching Lemma.