6.895 Randomness and Computation February 20, 2008

Lecture 5

Lecturer: Ronitt Rubinfeld *Scribe: Sam McVeety*

1 Uniform Distribution Learning

Definition 1 *Given hypothesis h,* error of h with f *is* $\text{error}(h) = \Pr_{x \in_U D}[f(x) \neq h(x)]$.

Definition 2 *A* uniform distribution learning algorithm *for concept class $\mathcal{C}$ is an algorithm $\mathcal{A}$ such that*

1. *$\mathcal{A}$ is given ϵ, δ, and has access to a set of samples $(x, f(x))$, where each x is chosen uniformly from the domain.*
2. *$\mathcal{A}$ outputs H such that with probability at least $\geq 1 - \delta$, error of h with f is at most ϵ.*

We don't know f, but we do know it is in concept class $\mathcal{C}$. Given access to an example oracle. δ is often referred to as the "security" or "confidence parameter". ϵ is the approximation error or "accuracy parameter".

1.1 Parameters of Interest

- m: "Sample complexity". The number of randomly chosen examples that we get from the oracle.
- ϵ: Accuracy parameter. Discussed above.
- δ: Security parameter. Discussed above.
- Running time? Might be different than the sample complexity or query complexity. Note that, trivially, $m \leq$ running time. We would like to get something polynomial in $\frac{1}{\epsilon}$, $\frac{1}{\delta}$ (actually, $\log \frac{1}{\delta}$) $\log |\mathcal{C}|$, $\log |D|$. Note that the requirement on the domain space results from the fact that we must write out any queries that we perform, so they should be simple to describe.
- Description of h should be relatively compact. Some models restrict this further.

Note that h is not necessarily in $\mathcal{C}$. In some models, "proper learning algorithms," this is the case, but we will not restrict our attention to them.

1.2 Remarks

- Special case of the PAC (probably approximately correct) model. More generally, Ex(f) for unknown distribution $\mathcal{D}$.
$$\text{error}(h) = \Pr_{x \in \mathcal{D}}[f(x) \neq h(x)]$$
- Can get the dependence on δ to be $\log \frac{1}{\delta}$.
- If the running time doesn't matter, only sample complexity, then it is easy to write learning algorithms. As we will see, though the algorithm that makes this possible in infeasible in practice.

2 Brute Force Algorithm

- Draw $M = \frac{1}{\epsilon}(\ln|\mathcal{C}| + \ln\frac{1}{\delta})$ samples. $\mathcal{C}$ is generally regarded as a class of exponential size.
- Search over all $h \in \mathcal{C}$ until one h labels all examples correctly and output it. Given multiple choices, chose arbitrarily. Notice that the sample complexity is fine, because we reuse our samples for each iteration of the search.

Motivation: h is bad, if the error of h with respect to $f \geq \epsilon$. Whether $h = f$ on all inputs or just most, doesn't really matter.

$$\begin{aligned} Pr[\text{bad } h \text{ is consistent with examples}] &\leq (1-\epsilon)^M \\ Pr[\text{bad } h \text{ ``survives''}] &\leq |\mathcal{C}|(1-\epsilon)^M \\ &\leq |\mathcal{C}|(1-\epsilon)^{\frac{1}{\epsilon}(\ln|\mathcal{C}|+\ln\frac{1}{\delta})} \\ &\leq \delta \end{aligned}$$

Therefore, this algorithm is unlikely to output any bad h.

Remark Ignoring the sample size bounds leads to bad statistics, where the sample size is insufficient for an inordinately large class space to rule out all of the bad h possibilities. The key is to define in advance the concept class, rather than expanding the concept class to fit "interesting" results.

3 Benefits of Learning

- Learning allows for prediction of f on other values of x.
- Learning can also allow for (lossy) compression.

4 Example: Monomial Functions

- $\mathcal{C} =$ conjunctions over $\{0,1\}^n$. $|\mathcal{C}| = 2^n$
- Example: $f = x_i x_j x_k$
- Cannot learn efficiently with 0 error. Think about a very long conjunction, which is 0 unless none of the variables are 0. In a polynomial number of queries, there is no way to detect this.
- Brute Force needs $\frac{1}{\epsilon}\left(\ln 2^n + \ln\frac{1}{\delta}\right) = O\left(\frac{n}{\epsilon} + \frac{1}{\epsilon}\ln\frac{1}{\delta}\right)$ samples.

4.1 Poly-time Algorithm

We say that a sample (or an input) $(x, f(x))$ is *positive* if $f(x) = 1$. If $f(x) = 0$, we say that the sample is *negative*. Let us describe the algorithm. We will set k that appears in the algorithm later on.

1. Draw $O(\frac{1}{\epsilon^2}\log\frac{1}{\delta})$ samples to estimate the fraction of positive inputs within $\epsilon/4$ with probability at least $1 - \frac{\delta}{2}$.
2. If in less than $\epsilon/2$ fraction of samples are positive, output $h(x) = 0$.
3. Take $\frac{k}{\epsilon}$ samples.

4. Let $V = \{i : x_i = 1 \text{ in all positive examples}\}$.

5. Output hypothesis $h(x) = \bigwedge_{i \in V} x_i$.

Why does this work? First we check if the function is sufficiently far from the all-zero function. If it is not, we can output the all-zero function as our hypothesis (Step 2). Otherwise, we know in Steps 3–5 that $\Pr_x[f(x) = 1] \geq \epsilon/4$. If x_i is in in f, then it must be turned on for every positive sample. If it isn't, it is on in a positive sample with probability 1/2, and off also with probability 1/2. We say that an index i is *bad*, if x_i does not appear in the monomial. Therefore, for $k = 8 \ln \frac{2n}{\delta}$, we have

$$\begin{aligned}
\Pr[\text{any bad } i \text{ survives}] &\leq \sum_i \Pr[i \text{ survives, given it is bad}] \\
&\leq \sum_i \Pr[x_i = 1 \text{ in all positive samples} \mid i \text{ is bad}] \\
&\leq \sum_i (1 - \Pr[f(x) = 1 \wedge x_i = 0 \mid i \text{ is bad}])^{k/\epsilon} \\
&\leq \sum_i (1 - \Pr[f(x) = 1 \mid i \text{ is bad}] \cdot \Pr[x_i = 0 \mid i \text{ is bad} \wedge f(x) = 1])^{k/\epsilon} \\
&\leq \sum_i \left(1 - \frac{\epsilon}{4} \cdot \frac{1}{2}\right)^{k/\epsilon} \leq \sum_i e^{-\ln \frac{2n}{\delta}} \leq \sum_i \frac{\delta}{2n} \leq \frac{\delta}{2}.
\end{aligned}$$

Note that if we detect all bad indices i, then the hypothesis h output in Step 5 exactly equals f.

In total, we only require $O\left(\frac{1}{\epsilon} \log \frac{n}{\delta} + \frac{1}{\epsilon^2} \log \frac{1}{\delta}\right)$ samples, which means that our algorithm is efficient.

Notice that learning requires logarithmically many queries, even for singletons, while dictator testing takes a constant number of queries. So, testing can be much faster than learning.

5 Learning via Fourier Coefficients

Approximating a single Fourier coefficient of an unknown function.

Lemma 3 *We can approximate any specific Fourier coefficient s to within γ (i.e. $\left|output - \hat{f}(s)\right| < \gamma$) with probability $\geq 1 - \delta$ in $O\left(\frac{1}{\gamma^2} \log \frac{1}{\delta}\right)$ samples.*

Proof We have $\hat{f}(s) = 2 \times \Pr[f = \chi_s] - 1$, and we can estimate $\Pr[f = \chi_s]$ to within $\pm\gamma/2$ using Chernoff bounds. ■

It is thought to be unlikely in this model that one can efficiently find the "heavy" Fourier coefficients with non-trivial values. Exhaustive search is intractable, and no other method is immediately clear. If we are dealing with a function class where most of the weight is in the coefficients with small s, we can exhaustively check these values. We will see this approach in work in the next lecture.