6.842 Randomness and Computation	March 30, 2022
Lecture 16	
Lecturer: Ronitt Rubinfeld	*Scribe: Nicolas Suter*

In this lecture, we pivot to the new topic of learning. We will start with the topic of learning boolean functions. At a broad level, we cover the following topics related to learning:

1. Model
2. Example
3. Occam's Razor

1 Model

There are many models of learning. We start with one in particular, based on an example oracle, but will explore others in future lectures.

1.1 Model Description

Example Oracle Description: let D be a domain of inputs to a function f, which we seek to learn. An *example oracle* $\mathrm{Ex}(f)$ takes m random samples $x \in D$ according to distribution $\mathcal{D}$ (which, for today, we assume to be the uniform distribution), runs them through f, and outputs m *labelled examples* of the form

$$(x_1, f(x_1))$$
$$(x_2, f(x_2))$$
$$\vdots$$
$$(x_m, f(x_m))$$

On these labelled examples, the learner then makes a hypothesis h, i.e., a guess of the true function f.

1.2 A Good Hypothesis

What would make such a hypothesis h optimal? *Ideally*, we have $h = f$, i.e., the hypothesis function is precisely the function we hoped to learn about. But this may be too ambitious, so *alternatively*, we might have $\mathrm{dist}(h, f) \leq \varepsilon$. That is, our learned hypothesis is at least reasonably close to the true function we wish to learn.

Remark Recall that the following are all equivalent:

- $\mathrm{dist}_{x\in D}(h, f)$
- $\mathrm{err}_{x\in D}(h)$ (implicitly understood to be w.r.t. f)
- h is ε-close to f
- $\Pr_{x\in D}(h(x) \neq f(x)) \leq \varepsilon$

Note that in this last interpretation, in general, we sample $x \in D$ according to distribution $\mathcal{D}$. For today, recall that we assume this is the uniform distribution.

However, in general, learning is *very* difficult. If f is an arbitrary function, there is nothing you can do which is efficient in terms of sample complexity m to learn f. However, if we know $f \in \mathcal{C}$, where $\mathcal{C}$ is a function family such as the class of linear functions, all k-term disjunctive normal forms, etc., then there is hope.

1.3 PAC Learning Algorithm

Definition 1 (uniform distribution learning algorithm for concept class $\mathcal{C}$) *An algorithm $\mathcal{A}$ such that*

- *$\mathcal{A}$ is given $\delta, \varepsilon > 0$ and access to example oracle $Ex(f)$ for $f \in \mathcal{C}$*
- *$\mathcal{A}$ outputs h such that, with probability of at least $1 - \delta$, $err(h) \leq \varepsilon$ with respect to f*

This learning scheme is called *Probably Approximately Correct* Learning: *probably*, referring to probability of at least $1 - \delta$ that the output is *approximately* correct, meaning it's error with respect to f is at most ε.

There are a few interesting parameters of this algorithm:

- m, the number of samples used, called the "sample complexity."
- ε, the accuracy parameter.
- δ, the accuracy parameter.
- runtime of $\mathcal{A}$, ideally something poly $\left(\log(|D|, \frac{1}{\varepsilon}, \frac{1}{\delta}\right)$.
- The description of h.
  - Should h be in $\mathcal{C}$? If so, this is a "Proper Learning" algorithm.
  - Is h compact and efficient to evaluate, meaning doing so takes time $O\left(\log|\mathcal{C}|\right)$?

Remark Note that the algorithm's runtime dependence on δ need not be more than $O\left(\log\frac{1}{\delta}\right)$, since as with other randomized algorithms, we can amplify by repeating the algorithm to improve our confidence.

Remark We've assumed $\mathcal{D}$ as uniform. In general, we measure err(h) with respect to f relative to distribution $\mathcal{D}$.

Remark Proper learning can be hard. In some cases, it's easier to allow h to be a function not in $\mathcal{C}$. However, both algorithms we'll see in this lecture are examples of proper learning algorithms.

2 Example: Learning Conjunctions

Let $\mathcal{C}$ be the set of conjunctions over the $\{0,1\}^n$ hypercube, i.e., the set of formulas of the form

$$f(x) = x_i \wedge x_j \wedge \overline{x_k} \wedge \dots$$

Alternatively, think of conjunctions as DNFs (disjunctive normal forms) of a single term.

Remark We can't do 0-error without incurring exponential sample complexity, since distinguishing between a conjunction of all n variables, such as

$$f(x) = x_1 \wedge x_2 \wedge \cdots \wedge x_n$$

and a conjunction which is always false,

$$f(x) = x_1 \wedge \overline{x_1} = 0$$

is only possible if one of the examples yielded by the oracle sets all n variables to the correct value, which if $\mathcal{D}$ is uniform, only happens with probability $\frac{1}{2^n}$.

2.1 Algorithm for Learning Conjunctions

1. Draw poly $\left(\frac{1}{\varepsilon}\right)$ samples.
2. Estimate $\Pr\left[f(x)=1\right]$ to additive error $\pm\frac{\varepsilon}{4}$, where this probability is over all possible variable assignments.
3. If estimate is less than $\frac{\varepsilon}{2}$, output $h(x)=0$, halt.
4. Else, our estimate was at least $\frac{\varepsilon}{2}$, implying the true probability is at least $\frac{\varepsilon}{4}$. Then, our example oracle will return a positive example (i.e., a $\mathbf{x}\in\{0,1\}^n$ such that $f(\mathbf{x})=1$ every $\frac{4}{\varepsilon}$ tries, in expectation. Thus the expected number of tries before the next positive example is $O\left(\frac{1}{\varepsilon}\right)$.
5. Collect k more positive examples (we'll evaluate what k should be in the analysis).
6. Define $V :=$ {Variables set the same way in every positive assignment}.
7. Output $h(x)=\bigwedge_{i\in V} x_i^{b_i}$, where b_i is negation if the i-th variable in V was negated in every positive example.

Example 2 *Consider the case where $n=6$ and the following examples satisfied f:*

$$\{0,0,1,0,1,1\}$$
$$\{0,1,1,0,1,1\}$$
$$\{1,1,1,0,1,1\}$$
$$\{0,0,1,0,0,1\}$$

Our algorithm would output $x_3\wedge\overline{x_4}\wedge x_6$, since these 3 variables had the same sign (positive for x_3 and x_6, negation for x_4) in every satisfying assignment.

2.2 Conjunctions Algorithm Analysis

Case 1: the algorithm did not halt at step 4 above. Then, for every i in the conjunction, it must be set the same way in every positive example, so it is in V, so our algorithm correctly learns its place in the conjunction.

For every i not in the conjunction, our algorithm's output will incorrectly include it in h only if the variable x_i happened to be set the same way in every positive example, of which there are k. The probability of this, for a specific i not in the conjunction, is

$$\Pr\left[i\in V\right]=2*\frac{1}{2^k}=\frac{1}{2^{k-1}}$$

By the union bound, the probability of this occurring for any of the n variables is at most

$$\Pr\left[\exists i\in V \text{ s.t. } i \text{ is not in the conjunction}\right]\leq\frac{n}{2^{k-1}}$$

So by picking $k=\log\frac{n}{\delta}+1=\Theta\left(\log\frac{n}{\delta}\right)$, this probability is upper bounded by δ.

Since we need $\Omega\left(\log\frac{n}{\delta}\right)$ positive examples, we need $\Omega\left(\frac{1}{\varepsilon}\log\frac{n}{\delta}\right)$ examples in total, and with this, we will return the exactly correct conjunction with probability of at least $1-\delta$, as desired.

Case 2: the algorithm did halt at step 4. Then our estimate of $\Pr\left[f(x)=1\right]$ was less than $\frac{\varepsilon}{2}$, and since we estimated this to additive error $\frac{\varepsilon}{4}$, the true probability is at most $\frac{3\varepsilon}{4}$. Then the error between

$h(x) = 0$ and the true f, $\text{err}(h) \leq \frac{3\varepsilon}{4}$, so we are ε-close to the true f with probability 1.

Thus, we see that in either case, this algorithm is, with high probability, mostly correct, i.e., it outputs a function which, on most inputs, will return the same output as f would.

3 Occam's Razor

Claim 3 *If runtime is ignored, then learning is easy with respect to sample complexity. i.e., by sacrificing runtime efficiency, we can always make learning require few samples*

3.1 Brute Force Algorithm

1. Draw $m = \frac{1}{\varepsilon}\left(\ln|\mathcal{C}| + \ln\frac{1}{\delta}\right)$ uniform examples. (Efficient with respect to sample complexity).
2. Search over all $h \in \mathcal{C}$ until finding one consisten with all examples. If multiple, pick one arbitrarily. (This step is expensive with respect to time).

3.2 Brute Force Algorithm Analysis

Case 1: Because f must be in $\mathcal{C}$, the algorithm must find at least one $h \in \mathcal{C}$ which is consistent with all examples, namely, $h = f$. If we output this h, our output is exactly correct.

Case 2: If we output a different h, we'd like to bound the probability that $\text{err}(h) > \varepsilon$. To do so, consider an arbitrary $h \in \mathcal{C}$ such that $err(h) > \varepsilon$. The probability that, despite this, it happens to agree with f on all m examples is

$$\Pr\left[h \text{ is consistent with } f \text{ on all } m \text{ examples}\right] \leq (1-\varepsilon)^m$$

Then, by the union bound, the probability that there exists any bad h such that it agrees with f on all m examples is at most

$$\Pr\left[\exists h \in \mathcal{C} \mid \text{err}(h) > \varepsilon \wedge h \text{ agrees with } f \text{ on all } m \text{ examples}\right] \leq |\mathcal{C}|\,(1-\varepsilon)^m$$

Substituting in for m,

$$\leq |\mathcal{C}|\,(1-\varepsilon)^{\frac{1}{\varepsilon}\left(\ln|\mathcal{C}|+\ln\frac{1}{\delta}\right)}$$

Note that $\lim_{x\to e}(1-x)^{\frac{1}{x}} = \frac{1}{e}$, and for every $x \in (0,1)$, this function is less than $\frac{1}{e}$, so we can bound the above with

$$\leq |\mathcal{C}|\left(\frac{1}{e}\right)^{\left(\ln|\mathcal{C}|+\ln\frac{1}{\delta}\right)} = |\mathcal{C}|\frac{1}{|\mathcal{C}|}\left(\frac{1}{e}\right)^{\left(\ln\frac{1}{\delta}\right)} = \delta$$

Thus, since this is an upper bound on the probability that such an h exists in $\mathcal{C}$, it is also an upper bound on the probability that such an h is output by the algorithm.

3.3 Closing Remarks

Remark The proof for the brute force algorithm didn't use anything special about the uniform distribution. It would work for any distribution $\mathcal{D}$ so long as the error and the example oracle were both defined with respect to the same $\mathcal{D}$.

Remark In general, once we have a good h, we can *predict* values of h on new random inputs, since by definition, a good h follows $\Pr_{x\in D}\left[f(x) = h(x)\right] \geq 1-\varepsilon$.

Additionally, we can *compress* a description of samples from

$$(x_1, f(x_1)), (x_2, f(x_2)), \ldots, (x_m, f(x_m))$$

Which takes $O\left(m\left(\log|D| + \log|R|\right)\right)$ space, where R is the range of f, to

$$x_1, x_2, \ldots, x_m, \text{description of } h$$

Which takes $O\left(m\log|D| + \log|\mathcal{C}|\right)$ space, assuming the description of h is compact, and thus requires $\log|\mathcal{C}|$ space. Thus, we see that learning and compression are related. There exist additional formal relations between these notions.