**6.5420 Randomness and Computation** April 2, 2026

# Lecture 14

*Lecturer: Ronitt Rubinfeld* *Scribe: Ellen Wu*

# 1 Introduction

Topics covered in this lecture:

- A model, learning algorithms
- An example: conjunctions over $\{0,1\}^n$
- Occam's Razor
- Fourier-based learning algorithms

Recall our earlier definition of "error" from previous lectures:

**Definition 1.** *We say $h$ is $\epsilon$-close to $f$ when*

$$\epsilon = \text{dist}(f,h) \equiv \Pr_{x\sim\mathcal{D}}[f(x) \neq h(x)] \equiv \text{error}_{x\sim\mathcal{D}}h$$

# 2 A model, learning algorithms

## 2.1 A model

The goal of this lecture is to figure out ways we can learn functions nontrivially given very limited (e.g. polylog in the domain size) random information about the function, for example, pairs $(x, f(x))$ for various input values $x$. Throughout this lecture we are interested in the following parameters:

- $m$: number of samples used (sample complexity)
- $\epsilon$: accuracy (how large our error can be)
- $\delta$: confidence (how confident we are that our error is $\leq \epsilon$)
- runtime: aims to be poly(log(domain size), $\frac{1}{\epsilon}, \log(\frac{1}{\delta})$)

We begin by defining the Example Oracle model.

**Definition 2.** *Example Oracle Ex($f$) description: every time the oracle runs, it does the following steps:*

1. *Picks random input $x$ chosen from domain of inputs using some distribution $\mathcal{D}$. $\mathcal{D}$ may be known or unknown to the user.*
2. *Computes $f(x)$ (where $f$ is unknown to the user)*
3. *Outputs $(x, f(x))$.*

After running the oracle $m$ times, we get $m$ random labeled examples

$$\{(x_1, f(x_1)), (x_2, f(x_2)), \cdots, (x_m, f(x_m))\}$$

After seeing the $m$ random labeled examples, then, given another input $x$ according to the distribution $\mathcal{D}$, the learner should output hypothesis $h(x)$ (with the goal of being equal to $f(x)$). For this lecture, we will mainly focus on when $\mathcal{D}$ is the uniform distribution over the domain.

### 2.2 Learning algorithms

**Definition 3.** *A uniform distribution learning algorithm for concept class $\mathcal{C}$ is an algorithm $\mathcal{A}$ where*

- *$\mathcal{A}$ is given inputs $\epsilon, \delta > 0$, and has access to Ex(f) for $f \in \mathcal{C}$, where Ex(f) uses uniform distribution on the entire domain.*
- *After running $\mathcal{A}$, $\mathcal{A}$ outputs $h$ such that, with probability $\geq 1 - \delta, \text{error}(h) \leq \epsilon$.*

**Example 4.** *If $f$ is a completely random function (e.g. flip a coin for each new domain element), then no oracle or algorithm can help the learner learn the function on new domain elements. In other words, if we have no restrictions as to what the function can be, it would be impossible to construct an efficient learning algorithm to guess the function.*

Since functions like Example 1 render learning oracles useless, we must assume some nice properties about $f$ for learning oracles to be useful. Specifically we will look at generally nice function classes $\mathcal{C}$.

## 3 An example: conjunctions over $\{0,1\}^n$

### 3.1 Conjunctions

For this section, we analyze the function class $\mathcal{C}$ := the set of conjunctions over the domain $\{0,1\}^n$. For example, $f(x) = x_i x_j \overline{x_k}$ returns true if and only if $x_i = 1, x_j = 1, x_k = 0$ (and the values of the other digits do not matter). Some specific examples:

- $f(x) = x_1 \overline{x_1}$ (always evaluates to false)
- $f_0(x) = \emptyset$ (empty conjunction, i.e. always evaluates to true)
- $f_1(x) = x_1 \overline{x_2} x_3 \overline{x_4} \cdots \overline{x_{n-1}} x_n$ (only evaluates true for one input)
- $f_2(x) = x_{10}$ (evaluates true for exactly half the inputs, i.e. all inputs where $x_{10} = 1$)
- $f_3(x) = x_1 \overline{x_2}$ (likewise, evaluates true for exactly a quarter of the inputs)

### 3.2 Learning algorithm for $\mathcal{C}$

1. Draw poly$(1/\epsilon)$ random examples
2. Estimate $\Pr[f(x) = 1]$ to additive error $\pm \frac{\epsilon}{4}$ (follows from Chernoff-Hoeffding bounds)
3. If estimate is $< \epsilon/2$, output $h(x) = 0$ on all new domain elements $x$ and halt.
4. If the algorithm has not halted, then we must know $\Pr[f(x) = 1] \geq \epsilon/4$. So a new random positive example occurs every $O(1/\epsilon)$ times the oracle is run with high probability. Now, collect $k = \Theta(\log(\frac{n}{\delta}))$ more positive examples.
5. Let $V = \{\text{variables set some way in each positive example}\}$. Output $h(x) = \bigwedge_{i \in V} X_i^{b_i}$.

### 3.3 Analysis

In the first case, the algorithm halts in step 3. Then, we must have $\Pr[f(x) = 1] < \frac{3\epsilon}{4}$. So, guessing $h(x) = 0$ for all inputs gives error$(h) \leq \frac{3\epsilon}{4}$.

In the second case, the algorithm does not halt in step 3. Then we have two cases: (1) if $i$ is in the conjunction, then it must be set the same way in each positive example, so $i \in V$. (2) If $i$ is not in the conjunction, then we have

$$\Pr[i \in V] \leq \Pr[i \text{ set the same way in each of } k \text{ examples}] \leq \frac{1}{2^{k-1}}$$

By a union bound, we have

$$\Pr[\text{any } i \text{ not in the conjunction survives}] \leq \frac{n}{2^{k-1}} \leq \delta$$

From this, we have with probability $\geq 1 - \delta$, the algorithm outputs $h$ with error $\leq \epsilon$.

From steps 1 and 4 of the algorithm, we require $O(\frac{1}{\epsilon}\log(\frac{n}{\delta}))$ examples and all other computations are fast, so this satisfies our runtime goals. □

## 4 Occam's Razor

### 4.1 The algorithm

The high level idea of Occam's Razor is that, if we don't have constraints on runtime but only have constraints on sample complexity, then learning is easy, as given by Occam's Razor algorithm, which is a brute force algorithm for any family $\mathcal{C}$ (where the brute force comes from step 2 below).

**Definition 5.** *Occam's Razor Algorithm: a learning algorithm for any function class $\mathcal{C}$:*

1. *Draw $M = \frac{1}{\epsilon}(\ln|\mathcal{C}| + \ln(\frac{1}{\delta}))$ (labeled) examples from Ex(f).*
2. *Search over all $h \in \mathcal{C}$ until we find one consistent with all the examples.*
3. *Output the $h$ found from step 2.*

### 4.2 Analysis

For some intuition as to why this works, note that steps 4 and 5 of the learning algorithm for the family of conjunctions from earlier is somewhat similar to this algorithm. So the analysis below will be pretty similar as well. The motivation for this algorithm is so that the analysis will just be to take a union bound over the smallest number of examples we can to get the easy and nice bound.

We prove that the probability that the output of the above algorithm has more than $\epsilon$ error is at least $1 - \delta$. We call $h$ "bad" if error$(h) \geq \epsilon$. We then have

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

□

**Remark.** Occam's Razor never relies on $\mathcal{D}$ being a uniform distribution, and it in fact works for any distribution $\mathcal{D}$.

**Remark.** The book, *The Bible's Code* written by Michael Drosnin was super popular back when it was published. In it, Drosnin found some interesting pattern in counting every $n$th letter of

the Torah spelling out messages. This led to many people who know how probability and stuff works trying to explain this with statistics. The question was whether it is a cool coincidence (with nonnegligible probability of happening) or is it actually "divine inspiration"?

# 5 Learning via Fourier

In this section we will introduce learning algorithms for Fourier representations of some function $f$. The goal is specifically to approximate one Fourier coefficient.

**Theorem 6.** *Given $S \subseteq [n]$, we can approximate $\hat{f}(S)$ to within error $\pm\gamma$ with probability $\geq 1-\delta$ in $O(\frac{1}{\gamma^2}\log(\frac{1}{\delta}))$ samples.*

*Proof.* Recall from last lecture where we defined $\chi_S(x) = \prod_{i \in S} x_i$. From here, we also proved (last lecture) that

$$\hat{f}(S) = 2\Pr_x[f(x) = \chi_S(x)] - 1$$

So we just need to estimate $\Pr_x[f(x) = \chi_S(x)]$ to within $\pm\gamma/2$. This can be seen with a standard Chernoff bound over testing $O(\frac{1}{\gamma^2}\log(\frac{1}{\delta}))$ random inputs $x$. □

**Remark.** We can't use a single set of samples to approximate all Fourier coefficients (or a large number of them) as the error will add up with a union bound. Also we can think of approximating all Fourier coefficients as equivalent to learning a random function.