Chapter 6

Learning Theory

In this lecture we give a brief introduction to learning theory. Let X be a finite set; we call any function $c : X \to \{0,1\}$ a *concept*. A set of concepts $\mathcal{C}$ is called a *concept class*. What does it mean for $\mathcal{C}$ to be *learnable*?

First a comment — why do we focus on learning a class instead of a particular function? The problem of "learning a function" is just an instance of "learning a concept class" when the class is the set of all possible functions. But that is intractable, as the function could just be arbitrary on unseen examples. Restricting to a class is a way of promising that the function we're looking for has some kind of structure. The question is whether that structure can be leveraged to learn (i) from few samples (*sample complexity*) and (ii) efficiently (*time and space complexity*).

6.1 The mistake-bounded model

Definition 6.1 (Consistency)**.** Given a set of labeled examples $\{(x_1, y_1), (x_2, y_2), ..., (x_m, y_m)\}$, where $x_i \in X$ are instances and $y_i \in \{0,1\}$ are Boolean labels, a *consistent* concept c is one such that $\forall i, c(x_i) = y_i$.

A basic model of learning consists in thinking of an online model where a set of labeled examples $\{(x_1, y_1), (x_2, y_2), ..., (x_m, y_m)\}$ are presented in sequence, and the goal of the algorithm is to make a prediction for the label y_i, given an example x_i, under the promise that there exists some concept in $\mathcal{C}$ such that $c(x_i) = y_i$ for all i.

Let's work out an example. Say $\mathcal{C}$ is the class of all conjunctions — formulas of the form $X_1 \wedge \overline{X_3} \wedge X_7 \wedge \cdots$. A simple algorithm would be the following:

(1) Start with the guess $c = X_1 \wedge \overline{X_1} \wedge X_2 \wedge \overline{X_2} \wedge \cdots \wedge X_n \wedge \overline{X_n}$ (yes, this returns zero all the time!);

(2) Upon seeing a *positive* example $(x, 1)$, strike out all literals remaining in c that are set to 0 in x.

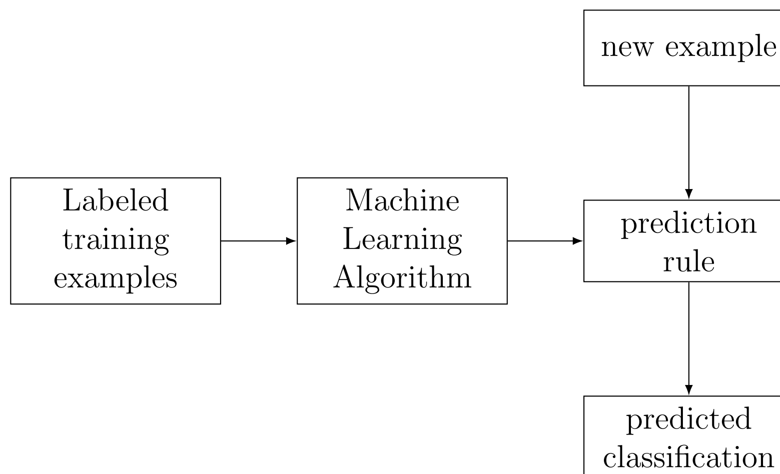
How well does this do? Well, a first observation is that in the worst case it could take foreover to get to the right c, as we simply could not be provided with the "right" examples. But that's ok — in this model we're just measuring mistakes. How many errors can we make in the worst case?

Exercise 1. Show that the above algorithm *never* makes more than $n + 1$ mistakes.

This model of learning has a number of drawbacks:

- There is no real notion of generalization — evaluation is measured against the past (the given data points) rather than some hypothetical "future" (this is similar to the experts algorithm for online learning that we saw: there also the only measure we cared about was the "regret" of the algorithm; here we're trying to go beyond such "backwards-looking" measure);
- We can't really claim that a successful algorithm has "learned" the "underlying structure" of the concept c (it's just able to make good predictions);
- The model doesn't incorporate the possibility for noisy data;
- It also doesn't incorporate randomness: in practice data will be presented according to some (possibly unknown) distribution, and we only care about our success in making accurate predictions under that particular distribution.

The following two-stage procedure perhaps describes better what we'd really want out of a *learning* procedure:



Here are some desirable characteristics of the model:

- Examples should be drawn from some target distribution $\mathcal{D}$, and the same distribution should be used to test the hypothesis produced by the algorithm;

- But the model should be distribution-free (a learning algorithm should work for any distribution);
- We should assume there does exist a target concept $c \in \mathcal{C}$ that labels the examples correctly (up to noise), and our goal is to learn it.

To summarize, our goal is to find an h (output hypothesis) that *generalizes*, meaning has low prediction error

$$err_{\mathcal{D}}(h) = \Pr_{\mathcal{D}}\big(h(x) \neq c(x)\big). \tag{6.1}$$

We want h with low error $\epsilon > 0$ (should be "approximately correct"), but we should also allow the algorithm to fail (i.e. return a completely irrelevant h) with some small probability (over the samples) $\delta > 0$ (should be "probably correct"). This is the framework of Probably Approximately Correct (PAC) learning.

6.2 PAC Learnability

Let $\mathcal{C}, \mathcal{H}$ be concept classes. They need not be the same, in which case we have what's called "agnostic" learning (typically we'll have $\mathcal{C} \subseteq \mathcal{H}$, but not necessarily). If the concept classes are the same, then it is called "proper" learning.

Definition 6.2. $\mathcal{C}$ is PAC learnable by $\mathcal{H}$ if there exists an algorithm A such that $\forall c \in \mathcal{C}$, for any distribution $\mathcal{D}$ over labeled examples, $\forall \epsilon, \delta > 0$, after having seen

$$m = \text{poly}(\varepsilon^{-1}, \delta^{-1}, \text{size of an example}, \text{size of } c)$$

many examples taken from $\mathcal{D}$, A produces a hypothesis $h \in \mathcal{H}$ such that

$$\Pr\left[err_{\mathcal{D}}(h) \leq \epsilon\right] \geq 1 - \delta,$$

where the probability is taken over the choice of the samples and the randomness of A.

A simple example of a learnable class is the set of all conjunctions, i.e. ANDs of a subset of the variables. The same algorithm we saw earlier works, but the analysis is a tiny bit more subtle: here we really want to bound the number of examples that's needed for us to produce a hypothesis which has a small probability of prediction error. In the worst case this could take forever, as the examples could just be unhelpful. Of course this is why we're measuring prediction error under the same distribution $\mathcal{D}$ as the samples come from. We're going to analyze a more general case soon, so we'll skip the analysis of this one — the bottom line is that $m = O((n/\varepsilon) \log 1/\delta)$ samples are sufficient to come up with a hypothesis such that, with probability at least $1 - \delta$, the hypothesis will correctly label a fraction $(1 - \varepsilon)$ of all possible examples — where the fraction is measured under the sample distribution $\mathcal{D}$, but the same algorithm works under *any* $\mathcal{D}$!

Other examples of classes that are efficiently learnable include

- Monotone conjunctions or disjunctions,

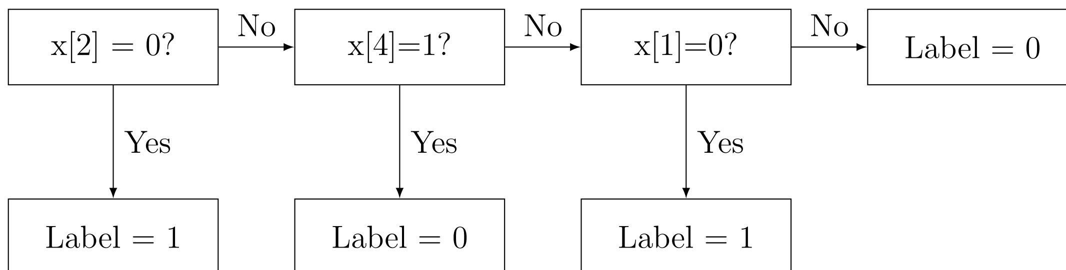
- k-CNF,
- Rectangles in the plane.

An example of class we don't know how to learn efficiently is the class of 2-term DNF (i.e. a single AND of two large disjunctions)! For 3-term DNF we know it is NP-hard (there is a reduction from 3SAT), but for 2-term DNF the question is open. Interestingly, there is a *agnostic* learning algorithm for learning 3-term DNF using 3-CNF (conjunctions of 3-term disjunctions).

Exercise 2. Give an agnostic learning algorithm for learning 3-term DNF using 3-CNF. *[Hint: show that any 3-term DNF over n Boolean variables can be expressed as a big conjunction on $O(n^3)$ variables, then use the learning algorithm from earlier.]*

6.2.1 Decision lists

Let's work out an example in detail: k-decision lists (k-DL). This is a pretty general class that includes k-CNF and k-DNF. An example of a 1-DL is something like this:



Here we wrote x_i for the ith example and $x_i[j]$ for the jth bit of ith example. A k-DL more generally allows each of the questions to depend on k bits of x.

Suppose we suspect that there exists a good decision list, we would like to:

- Design an algorithm A that finds a consistent decision list, if one exists.
- Show that if S is large enough (but hopefully not too large), then the probability that there exists a consistent decision list h with $err_{\mathcal{D}}(h) > \epsilon$ is at most δ.
- A is a good algorithm to use if c is actually a decision list. Meaning that if S is of reasonable size, A indeed PAC-learns a decision list.

For example, suppose we are given the following list of samples:

Example	Label
1 0 0 1 1	$+$
0 1 1 0 0	$-$
1 1 1 0 0	$+$
0 0 0 1 0	$-$
1 1 0 1 1	$+$
1 0 0 0 1	$-$

Then a good decision list reproducing this table could be the following sequence of decisions: If $x[1] = 0$ then "$-$", else if $x[2] = 1$ then "$+$" else if $x[4] = 1$ then "$+$" else "$-$". So here is a candidate algorithm:

(1) Start with an empty list;

(2) Build up rules top to bottom, at each step finding a rule that is consistent with the remaining data and correctly labels at least one point.

By definition this algorithm will always return a consistent decision list. Moreover, as long as there exists a consistent decision list, we will find one (there is always a valid DL for the as-yet-unclassified examples: the original DL that worked for everything). But does the algorithm *generalize* — does the decision list it produces "probably approximately correctly" predict future examples? Note that the more samples we take the more likely this is to be the case — so the question is, how many samples do we need before we can produce a good hypothesis?

We're going to see a very general method to place a bound on the sample size; it is called "Occam's razor" (="the simplest hypothesis works best"!). Let $\mathcal{D}$ be some (unknown) distribution, and suppose some decision list h has $err_{\mathcal{D}}(h) > \epsilon$. Then assuming the samples are taken according to $\mathcal{D}$, we get

$$\Pr\big(h \text{ is consistent with } S\big) \leq (1-\epsilon)^{|S|}.$$

Let $|\mathcal{H}|$ be the number of decision lists over n Boolean variables. For each question, there are at most $2n^k$ possibilities: $\binom{n}{k} \leq n^k$ possibilities for the check, and 2 choices for the label. Thus a gross overestimate of $|\mathcal{H}|$ is $(2n^k)^{n^k}$, as there will be at most n^k (the number of possible questions) steps in the list. Applying a union bound,

$$\begin{aligned}\Pr\big(\exists \text{ some decision list } h \text{ with } err(h) > \epsilon \text{ and } h \text{ is consistent with } S\big) &\leq |\mathcal{H}|(1-\epsilon)^{|S|} \\ &\leq |\mathcal{H}|e^{-\epsilon|S|}.\end{aligned}$$

This last quantity is less than δ for $|S| > \frac{1}{\epsilon}(\ln|\mathcal{H}| + \ln(\frac{1}{\delta}))$. Therefore for any constant k the algorithm we described earlier is efficient: it only requires about $O(n^k/\varepsilon)$ to generate a hypothesis that generalizes.

The technique we used to bound the sample size is very general: as long as we can bound the number of concepts in the class, then we can state a bound on the number of required samples similar to the above. In particular, if computation is not a factor then any *finite* class is PAC-learnable.

In practice, the notion of PAC learning has some drawbacks:

- The cost of labeled examples can be much higher than time (or vice-versa);
- The model ignores other sources of information, such as prior on $\mathcal{D}$, accessibility of cheap unlabeled data, etc;
- We might want a stronger notion where a "good" hypothesis is guaranteed to be found even if there does not exist a perfect hypothesis in $\mathcal{C}$;
- $\mathcal{H}$ may not be finite.

6.2.2 Infinite concept classes and the VC dimension

Let's end by discussing the last issue, that of learning infinite concept classes. Is this even possible? The notion of VC dimension is the appropriate measure of "size" of a class.

Definition 6.3. A set of points S is *shattered* by $\mathcal{C}$ if there exists concepts in $\mathcal{C}$ that split S in all $2^{|S|}$ possible ways.

The VC-dimension of a concept class $\mathcal{C}$ is the size of the largest set of points that can be shattered by $\mathcal{C}$. (If there is no largest such set then the VC-dimensions is infinite.)

Example 6.4.

- The VC-dimension of $\mathcal{C} = \{[0, a],\ 0 < a < 1\}$ (where examples are all points in $(0, 1]$) is 1, as single points can be shattered, but no two distinct points can (it is impossible to include the larger one while excluding the smaller).
- If we allow all $[a, b]$, $0 < a < b < 1$, then the VC-dimension is 2.
- The VC-dimension of half-spaces in the plane is 3. More generally, for half-spaces in $\mathbb{R}^n$ the VC-dimension is $n + 1$.

Exercise 3. Determine the VC-dimension of the class of all axis-aligned rectangles in the plane.

The main result which generalizes the bound we showed for the finite case is the following.

Theorem 6.5. *For any class $\mathcal{C}$ and distribution $\mathcal{D}$, if*

$$m = |S| > \frac{2}{\varepsilon}\Big(\log_2(2\,\mathcal{C}[2m]) + \log_2\big(\frac{1}{\delta}\big)\Big),$$

where for a sample S, $\mathcal{C}[S]$ is the set of possible labellings of S using concepts in $\mathcal{C}$ and $\mathcal{C}[m] = \max_{|S|=m} |\mathcal{C}[S]|$, then with probability at least $1 - \delta$ all $h \in \mathcal{C}$ with $err_{\mathcal{D}}(h) > \epsilon$ are inconsistent with the data.

Now the key point that makes the theorem useful is that the growth of $|\mathcal{C}[m]|$ can be bounded as a function of the VC-dimension. This is called Sauer's lemma:

Lemma 6.6 (Sauer)**.** *For any concept class $\mathcal{C}$ with finite VC-dimension d and any $m \geq 1$,*

$$|\mathcal{C}[m]| \leq \sum_{i=0}^{d} \binom{m}{i} \; = \; O(m^d).$$

As an immediate corollary, we see that the bound from Theorem 6.5 can be replaced by $m = \Omega(\varepsilon^{-1}(d \log \varepsilon^{-1} + \log \delta^{-1}))$, showing that any concept class can be learned using a number of samples that is linear in the VC-dimension of the class. Note that the theorem implies that *any* algorithm which produces a *consistent* hypothesis after that number of samples will automatically have produced a hypothesis that generalizes with error at most ε, *whatever* the underlying distribution $\mathcal{D}$!

6.3 Analysis of Boolean Functions

We will study learnability of certain classes of Boolean functions. Our main tool is Fourier analysis, which we review in this section.

6.3.1 Fourier Expansions

The main tool in the analysis of boolean functions is the representation of boolean functions as real, multilinear polynomials: the Fourier expansion. A multivariable polynomial is multilinear if $\forall i$, the exponent of x_i in each term is 0 or 1.

6.3.2 Method for finding Fourier expansions

Notice that the following function f checks if $x_j = 1$, since $f(1) = 1$ and $f(-1) = 0$:

$$f = \frac{1}{2} + \frac{1}{2}x_j \tag{6.2}$$

This gives us a method for checking if the inputs to a Boolean function f take on a specific value on a vertex of the Hamming cube.

Definition 6.7. For $a = (a_1, a_2, \ldots a_n) \in \{-1, 1\}^n$, the indicator polynomial

$$1_a(x) = (\frac{1 + a_1x_1}{2})(\frac{1 + a_2x_2}{2}) \cdots (\frac{1 + a_nx_n}{2})$$

satisfies $1_a(a) = 1$, and $1_a(x) = 0$ for $x \neq a$.

Example 6.8. Suppose a Boolean function f satisfies $f(1, 1, 1) = 1$. The indicator function

$$1_{(1,1,1)}(x) = (\frac{1 + x_1}{2})(\frac{1 + x_2}{2})(\frac{1 + x_3}{2})$$

satisfies $1_{(1,1,1)}(1, 1, 1) = 1$, and is 0 otherwise.

Theorem 6.9. *A Boolean function f has the representation $f(x) = \sum_{a \in \{-1,1\}^n} f(a)1_a(x)$.*

Notice that this holds for real-valued Boolean functions as well. Also, each term in the representation is a product of x_i's.

Definition 6.10. For each subset $S \subseteq [n]$, let $\chi_S(x) = \prod_{i \in S} x_i$, and $\chi_\emptyset(x) = 1$. Let $\hat{f}(S)$ be the coefficient on the monomial $\chi_S(x)$ in the above multilinear representation of f.

Theorem 6.11. *Every function $f : \{-1,1\}^n \to \mathbb{R}$ can be uniquely expressed as a multilinear polynomial:*

$$f(x) = \sum_{S \subseteq [n]} \hat{f}(S)\chi_S(x) \ .$$

This expression is the Fourier expansion of f, and $\hat{f}(S)$ are the Fourier coefficients. Collectively, the Fourier coefficients are the spectrum of f.

Example 6.12.

$$\widehat{\max_2}(\emptyset) = \frac{1}{2}, \quad \widehat{\max_2}(\{2\}) = \frac{1}{2}, \quad \widehat{\max_2}(\{1,2\}) = -\frac{1}{2}.$$

$$\widehat{\mathrm{maj}_3}(\emptyset) = 0, \quad \widehat{\mathrm{maj}_3}(\{1,2\}) = 0, \quad \widehat{\mathrm{maj}_3}(\{1,2,3\}) = -\frac{1}{2}.$$

Definition 6.13.

$$\mathrm{mean}(f) = \mathop{\mathbf{E}}_x[f(x)].$$

$$\mathrm{variance}(f) = \mathop{\mathbf{E}}_x[f(x)^2] - \mathop{\mathbf{E}}_x[f(x)]^2.$$

Notice that if f maps to $\{-1,1\}$ then $\forall x,\ f(x)^2 = 1$ so $\mathop{\mathbf{E}}_x[f(x)^2] = 1$. The set of all functions $\{f : \{-1,1\}^n \to \mathbb{R}\}$ forms a vector space with dimension 2^n. Specifically, an inner product space of 2 functions:

$$\langle f, g \rangle = \mathop{\mathbf{E}}_x[f(x) \cdot g(x)]$$

For $\{-1,1\}$-valued boolean functions, the dot product $\langle f, g \rangle$ measures the "closeness" of f, g. When $f(x) = g(x)$, $f(x)g(x) = 1$. When $f(x) \neq g(x)$, $f(x)g(x) = -1$.

$$\langle f, g \rangle = (1 - \mathop{\mathbf{Pr}}_x[f(x) \neq g(x)]) - \mathop{\mathbf{Pr}}_x[f(x) \neq g(x)] = 1 - 2\mathop{\mathbf{Pr}}_x[f(x) \neq g(x)]$$

So, each function can be represented as a linear combination of other functions, and the parity coefficients form a basis that spans the vector space.

6.4 Learning juntas

Let $f : \{-1,1\}^n \to \{-1,1\}$ be a Boolean function. Any such f has a Fourier expansion $f = \sum \hat{f}(S)\chi_S$, with $\chi_S(x) = \prod_{i \in S} x_i$.

Definition 6.14. We say that f *depends* on the i-th variable (or, that the i-th variable is *relevant* for f) if there exists $x, x' \in \{-1,1\}^n$ such that x, x' differ only on the i-th coordinate and $f(x) \neq f(x')$.

A function that depends on at most k variables is called a k-junta.

Our goal in this lecture is to develop an algorithm for learning k-juntas, where you should think of $k \ll \log n$, but k could still grow with n. We will do this in the PAC learning model, under the uniform distribution $\mathcal{U}$. That is, we will only prove that the algorithm works if the distribution $\mathcal{D}$ used for the samples is uniform.

6.4.1 Estimating Fourier coefficients

We start off with a general-purpose algorithm for learning any desired Fourier coefficient. So suppose that we fix an $S \subseteq \{1, \ldots, n\}$ and we wish to estimate $\hat{f}(S)$, given uniform samples $(x, f(x))$. By definition,

$$\hat{f}(S) = \frac{1}{2^n} \sum_{x \in \{-1,1\}^n} f(x)\chi_S(x) = \mathop{\mathbf{E}}_{x \sim \mathcal{U}} \big[f(x)\chi_S(x)\big].$$

If we are given independent and uniformly distributed samples $\{(x_i, f(x_i))\}_{i=1,\ldots,m}$ for $x_i \sim \mathcal{U}$ we can compute the average over the samples of $f(x_i)\chi_S(x_i)$. This will give the correct value on expectation. Moreover by the Chernoff bound, for any k,

$$\mathbf{Pr}\left(\Big|\frac{1}{m}\sum_{i=1}^{m} f(x_i)\chi_S(x_i) - \hat{f}(S)\Big| > \frac{1}{2^{k+1}}\right) \leq 2e^{-\frac{m}{32^{2(k+1)}}},$$

so if we take $m = \mathrm{poly}(2^k, \log(1/\delta))$ samples we can get an estimate that is correct within $\pm\frac{1}{2^{k+1}}$ with probability $1-\delta$. This is enough to infer the *exact* value for the Fourier coefficient:

Lemma 6.15. *Suppose f is a k-junta. Then for any $S \subseteq \{1, \ldots, n\}$ it is possible to determine $\hat{f}(S)$* exactly*, with probability $1 - \delta$, using $m = \mathrm{poly}(2^k, \log(1/\delta))$ i.i.d. samples $\{(x_i, f(x_i))\}_{i=1,\ldots,m}$ where $x_i \sim \mathcal{U}$.*

Proof. We already saw that $\hat{f}(S)$ could be *estimated* to within $\pm 2^{-(k+1)}$. Now call f' the function $f' : \{-1,1\}^k \to \{-1,1\}$ that corresponds to the restriction of f to the k variables on which it depends. f and f' have exactly the same Fourier coefficients. Moreover, any Fourier coefficient of f' is $\hat{f'}(S) = \mathbf{E}_{x \sim \mathcal{U}} f'(x)\chi_S(x)$, an expectation over 2^k variables, each ranging over $\{\pm 1\}$. Thus $\hat{f'}(S)$ can only take values in increments of $\pm 2^{-k}$, and knowing it to within $\pm 2^{-(k+1)}$ suffices to determine it exactly. $\square$

Lemma 6.21 already gives an algorithm for learning *low-degree* functions, where given an $f : \{-1,1\}^n \to \{-1,1\}$ its degree is defined as follows. (Soon we will introduce a similar, but very different (!), notion of degree for a Boolean function, its $\mathbb{F}_2$-*degree*.)

Definition 6.16. If $f : \{-1,1\}^n \to \{-1,1\}$, $f = \sum_S \hat{f}(S)\chi_S$, we call *real degree* of f and denote $\deg_{\mathbb{R}}(f)$ the largest $|S|$ such that $\hat{f}(S) \neq 0$.

Suppose f has real degree d. Then we know the only nonzero Fourier coefficients are $\hat{f}(S)$ for $|S| \leq d$. There are at most $\binom{n}{d} \leq n^d$ of those, so that applying Lemma 6.21 with a $\delta' = \delta/n^d$ together with a union bound we can conclude that all nonzero Fourier coefficients

of f can be learned exactly with probability $1-\delta$ from $\text{poly}(2^k, d\log n\log(1/\delta))$ samples and time $n^d\text{poly}(2^k, d\log n\log(1/\delta))$.

Is this good? Well, if f depends on at most k variables, then certainly it has degree $d \leq k$. How many functions are there that depend on at most k variables? At most $n^k 2^{2^k}$: choose the k variables (as an ordered set) and then specify an arbitrary k-bit function. Thus using Occam's razor from the past lecture, we can learn this concept class under *any* distribution from $\text{poly}(\varepsilon^{-1}, \log(\delta^{-1}), k2^k\log n)$ samples using the brute-force algorithm (simply take that many samples and output any hypothesis consistent with the samples). This will also take time $O(n^k 2^{2^k})$ as given the samples we need to brute-force through all possible functions.

So we're not doing such an impressive job here: if f is a k-junta *and* f has low real degree $d \ll k$, then we're doing well, but if say f is parity on k variables then the real degree is k and we're not doing anything interesting. So, what about this case, can we do better, and then somehow combine the two procedures?

6.4.2 Learning parities

First we show that we can learn parities from few samples, efficiently.

Lemma 6.17. *Suppose $f : \{-1,1\}^m \to \{-1,1\}$ is a parity on an unknown subset of variables. Then it is possible to PAC-learn a parity h such that $err_{\mathcal{D}}(h) \leq \varepsilon$ with probability $1-\delta$ using $O(\varepsilon^{-1}(m+\log\delta^{-1}))$ samples and time $O((\varepsilon^{-1}(m+\log\delta^{-1}))^3)$.*

Proof. The sample size bound is just Occam's razor, since there are 2^m parities on m bits. Once we have enough samples, finding a consistent hypothesis is solving a system of linear equations, which can be done in cubic time (and even matrix multiplication time N^ω, for $\omega < 2.376$). □

Now let's generalize this to functions that are parities of ANDs of a few variables. To study such functions we introduce a second notion of degree, the $\mathbb{F}_2$-*degree*.

Definition 6.18. The $\mathbb{F}_2$-degree of $f : \{0,1\}^n \to \{0,1\}$ is the degree of f as a polynomial over $\mathbb{F}_2$, i.e. the largest $|T|$ such that $a_T \neq 0$ in an expansion $f(x) = \oplus_{T\subseteq\{1,\ldots,n\}} a_T(\wedge_{i\in T} x_i)$, where $a_T \in \{0,1\}$.

Exercise 4. Show that for any Boolean f, $\deg_{\mathbb{F}_2}(f) \leq \deg_{\mathbb{R}}(f)$.

Lemma 6.19. *Suppose $\deg_{\mathbb{F}_2}(f) = \alpha \leq k$. Then we can learn f exactly from $N = n^\alpha\text{poly}(2^k, n, \log\delta^{-1})$ uniform samples in time $O(N^3)$, with success probability at least $1-\delta$.*

Proof. Consider a new set of $\binom{n}{\alpha}$ variables to represent all monomials of degree at most α. Then run the algorithm for learning parities, with ε set to $2^{-(k+1)}$. This will let us recover a correct hypothesis using $N = O(2^k(n^\alpha + \log\delta^{-1}))$ samples in time N^3, as claimed.

Next we need to argue that our hypothesis h is *exactly* f. Both f and h have $\mathbb{F}_2$-degree at most α, and the guarantee is that, over $\mathbb{F}_2$,

$$err_{\mathcal{U}}(h) = \Pr_x(h(x) + f(x) \neq 0) \leq 2^{-(k+1)}.$$

Using a slightly non-standard version of the Schwartz-Zippel lemma we know that if $f+h$ is nonzero it must be nonzero on at least a fraction $2^{-\alpha} \geq 2^{-k}$ of points in $(\mathbb{F}_2)^n$. So the only possibility for $err_{\mathcal{U}}(h) \leq 2^{-(k+1)}$ is that in fact the error is 0, i.e. we recovered f exactly. □

6.4.3 Learning juntas

So far we've seen two learning algorithms. One is able to learn k-juntas that have low degree over $\mathbb{R}$, and the other is good at learning juntas that have low degree over $\mathbb{F}_2$. Each algorithm works at an opposite end of the spectrum: can we combine them into a single efficient algorithm, that works well (meaning it beats the trivial $O(n^k)$ bound) for all k-juntas?

Indeed this is the case: we will show that any k-junta can be learned efficiently by one of the two learning algorithms: either it has low $\mathbb{F}_2$-degree, or it has at least one large Fourier coefficient (it does not necessarily have low $\mathbb{R}$-degree, but this will be enough). We will make the distinction between the two options using the notion of being *strongly balanced*.

Definition 6.20. Suppose f is such that $\hat{f}(S) = 0$ for all $1 \leq |S| \leq t$. If $\hat{f}(\emptyset) = 0$ also then we say f is strongly balanced up to size t. If $\hat{f}(\emptyset) \neq 0$ we say f is strongly biased up to size t.

The justification for the definition is that, if f is strongly balanced, not only is the average value of f equal to 0 (this is just a consequence of $\hat{f}(\emptyset) = 0$), but also it is easy to see that the average of f, when up to $t-1$ variables have been fixed arbitrarily, is still 0 (as the resulting function of $n-(t-1))$ variables will still have its Fourier coefficient associated to the empty set equal to 0). The same observation holds for strongly biased functions.

First let's see that strongly balanced functions have low $\mathbb{F}_2$-degree.

Lemma 6.21. *Suppose f is strongly balanced up to size $t \leq n-1$. Then $\deg_{\mathbb{F}_2}(f) \leq n-t$. (If f is strongly balanced up to size $t = n$ then necessarily f is parity on all variables, or its negation.)*

Proof. Let $f = \sum \hat{f}(S)\chi_S$ be the representation of f as a polynomial over $\mathbb{R}$, and $g = f \cdot x_1 \cdots x_n$. Then g has zero coefficient on all monomials of size $|S| > n-t$. Then $\deg_{\mathbb{R}}(g) \leq n-t$, which by Exercise 4 implies $\deg_{\mathbb{F}_2}(g) \leq n-t$. Since (in the Boolean representation) g is obtained by adding a degree-1 function (parity) to f, it follows that also $\deg_{\mathbb{F}_2}(f) \leq n-t$ (provided $n-t \geq 1$). □

Next we look at strongly biased functions.

Lemma 6.22. *If f is strongly biased up to size t, then $t \leq 2n/3$.*

Proof. Let $f = \sum \hat{f}(S)\chi_S$ be the representation of f as a polynomial over $\mathbb{R}$, and U of maximal size such that $\hat{f}(U) \neq 0$; necessarily $|U| \geq t$. Expanding $f_{\mathbb{R}}^2$, there will be a nonzero cross-term $\hat{f}(\emptyset)\hat{f}(U)x_U$. But $f^2 \equiv 1$, so this term must cancel out after multilinear reduction (using $x_i^2 = 1$ for each i). If $t > 2n/3$ then $\hat{f}(T) = 0$ for all $|T| \leq 2n/3$, so nonzero cross-terms not involving $\hat{f}(\emptyset)$ are necessarily on sets of size $|V| < 2n/3$ (draw a picture!), hence cannot cancel out the x_U term. □

We now have everything we need to conclude.

Theorem 6.23. *k-juntas over n bits can be learned exactly from uniform samples with probability $1-\delta$, using $N = n^{k/4}\mathrm{poly}(2^k, n, \log 1/\delta)$ samples and time $O(N^3)$.*

The way to think about the running time here is $\sim n^{3k/4}$, which is an improvement over the naïve brute-force algorithm for ranges of k such as $k \sim \log n$.

Proof. Let f' be the k-variable function induced by f on its k relevant variables. Let $t = 3k/4 > 2k/3$. If f' is strongly balanced up to size t then by Lemma 6.27 $\deg_{\mathbb{F}_2}(f') \le k/4$, and by Lemma 6.25 it can be learned from $N_1 = n^{k/4}\mathrm{poly}(2^k, n, \log 1/\delta)$ samples in time $O(N^3)$.

Now suppose f' is not strongly balanced up to size t. By Lemma 6.28, since $t > 2k/3$ it is also not strongly biased up to size t, and so f' must have a nonzero Fourier coefficient associated with a set of size at most $t = 3k/4$. Using the discussion after Lemma 6.21 we can estimate all such coefficients, hence find a nonzero one, using $N_2 = \mathrm{poly}(2^k, n, \log 1/\delta)$ in time $O(n^t N_2)$. Once we have found a nonzero coefficient, we have in particular identified a relevant variable and we can recurse: if $k' < k$ relevant variables have been identified, we can simply learn all induced functions obtained by setting the values of the k' variables to all $2^{k'}$ possibilities. We'll have a complexity blow-up of $k2^k$, but that's ok since we think of k as small compared to n, and the sample complexity is already $\mathrm{poly}(2^k)$ anyways. □