1 Lecture 7: 2017.03.13

Outline

- SQ model: review and lower bounds
- PAC learning and cryptography
- Boosting Part I

1.1 Query complexity in SQ

We saw that any PAC-learnable model can be framed in the SQ model by simulating the queries with enough draws from the distribution i.e calls to $EX(c, D)$. Intuitively, the query complexity of the obtained model should not violate the lower bounds from previous lectures on the VCD.

Claim 1.1. *Assume $VCD(\mathcal{C}) = d$, then at least $\Omega\left(\frac{d}{\log d}\right)$ queries with $\tau = O(\varepsilon)$ are required in the SQ model.*

Sketch. Let $\{x_1, \ldots, x_d\}$ be shattered by $\mathcal{C}$. Let D such that x_1 has weight $1 - 3\varepsilon$ and the rest have uniform weight $\frac{3d}{d-1}$. Consider the query $\chi = 1 \iff x = x_1 \ \& \ y = 0$, this allows the algorithm to learn the label of x_1. Now let $b_i = c(x_i)$, we must learn the labels of the majority of these labels. Specifying some other labels c_i, we can make statistical queries like $\mathbb{P}_i[b_i = c_i]$. For example, "what is the proportion of 1's among b_i's?". This statistical query can be thought of as the inner product of this unknown $\vec{b}$ and any $\vec{c}$ that we want. Therefore, the learning problem reduces to a linear algebra problem. □

Recall the following remark from last time.

Claim 1.2. *"Almost" every PAC-learning algorithm has an SQ-algorithm.*

We can actually find counterexamples to the claim, hence the "almost" in the statement.

1.1.1 PAC-learnable but not SQ-learnable

Let $X = \{0, 1\}$ and consider the parity functions over X. Namely, for any subset $S \subset \{1, \ldots, n\}$ we can define

$$f_S(\vec{x}) = \sum_i x_i \mod 2 \in \{0, 1\}, \tag{1.1}$$

and the associated pairs $\langle \vec{x}, f_S(\vec{x}) \rangle$. They count the parity of the 1 bits in the set S. Note that $|\mathcal{C}| = 2^n$. It turns out that this is PAC-learnable because examples correspond to linear equations but you can't can get the same information statistically. We have the info-theoretical result.

Theorem 1.3. *Parity functions satisfy the following:*

i) PAC-learnable (poly time & samples)

ii) Provably require exponential in n queries in SQ.

Intuition. Let D be uniform over X.

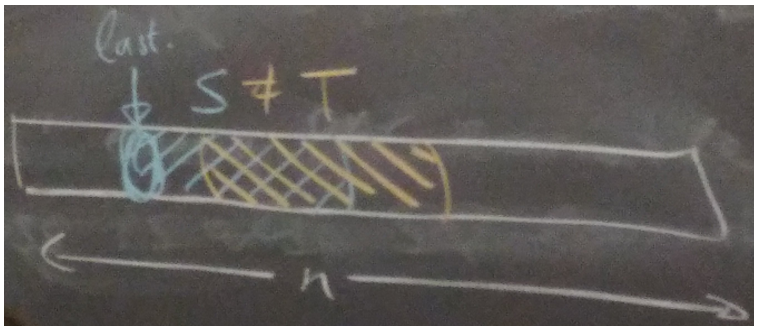

Figure 1: Parity functions.

If we flip coins to decide the bits, by the time we reach the last one, T is completely determined but S relies on the result completely.

$$\forall S \neq T, \quad \mathbb{P}[f_S(\vec{x}) = f_T(\vec{x})] = \frac{1}{2} = \mathbb{P}[f_S(\vec{x}) \neq f_T(\vec{x})] \tag{1.2}$$

Basically, the crux of the proof of ii) is proving that no matter what kind of queries we ask they reduce to the "Is the answer this function?" type of questions. Since, there are 2^n different functions we need an exponential amount of queries. □

We can rephrase parity functions in the following way. Take a target vector $\vec{c}$ in X,

$$f_S(\vec{x}) = \sum_i c_i x_i \mod 2, \tag{1.3}$$

then any pair $\langle \vec{x}, f_{\vec{c}}(\vec{x}) \rangle$ represents an algebraic equation on the unknowns c_i's. Therefore, this can be learned in polynomial time by a PAC-learning algorithm.

1.1.2 Characterizing Query Complexity in SQ

Definition 1.4. We define the SQ-dimension as

$$SQD(\mathcal{C}, D) \coloneqq \max_d \{\exists f_1, \ldots, f_d \in \mathcal{C} : \forall 1 \leq i \neq j \leq d, \\ \left|\mathbb{P}_D[f_i(x) = f_j(x)] - \mathbb{P}_D[f_i(x) \neq f_j(x)]\right| \leq \frac{1}{d^3}\Big\}. \tag{1.4}$$

Note that in the case of $\{0, 1\}$, we can think of f_i with labels ± 1 as vectors with 2^n entries (corresponding to all subsets). Then, $\mathbb{P}_D[f_i(x) = f_j(x)] - \mathbb{P}_D[f_i(x) \neq f_j(x)]$ corresponds to the inner product of f_i and f_j. With this in mind, this definition is only saying that we want to find a large set of almost orthogonal vectors.

Theorem 1.5. *Let $SQ(\mathcal{C}, D) = d$, then at least $\Omega\left(d^{\frac{1}{3}}\right)$ queries with $\tau = O\left(\frac{1}{d^{\frac{1}{3}}}\right)$ are required to SQ-learn $\mathcal{C}$ wrt. D.*

Recall some of the PAC-learnable classes of functions we have seen.

- Conjunctions over $\{0,1\}^n$
- decisions lists (problem in the book)
- 3-term DNF by 3CNF.

The problem is that none of the are "universal" in the sense that they cannot represent all the boolean functions. In contrast,

general DNF functions: Expressions of the form $T_1 \vee \ldots \vee T_m$ where each T_i is a conjunction over $x_1, \ldots, x_n, \neg x_1, \ldots, \neg x_n$. Now, any boolean function can be represented as a look-up table. Since we can add a T_i for each of the entries of this table, they all can be represented by general DNF.

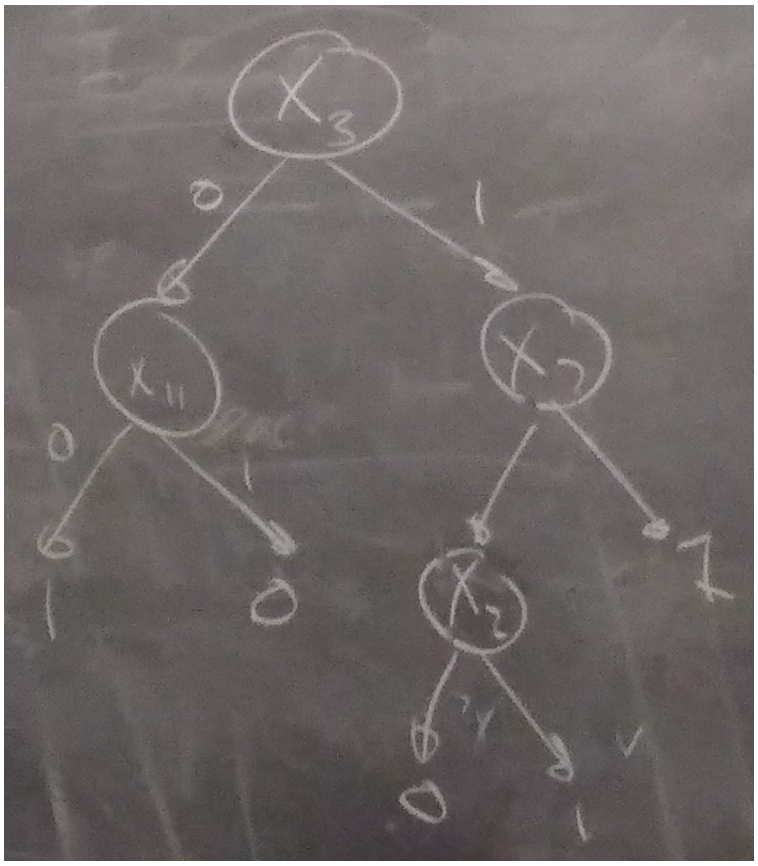

Figure 2: Decision tree.

decision trees: As before, we can copy the look-up table with a big enough table.

Remark 1.6. *It is a big open problem whether these two classes are PAC-learnable. At least, we can use the theorem above to prove that they cannot be SQ learnable. This is in sharp contrast with the current state of the ML literature where most algorithms are SQ. This means that to PAC-learn these classes we would need radically different algorithms to the ones available at the moment.*

Consider parity functions $f_S(\vec{x})$ where $|S| = \log(n)$. We have $\binom{n}{\log(n)} \approx n^{\log(n)}$. Then, $SQD = n^{\log(n)} \to \Omega(n^{\frac{\log(n)}{3}})$. Consider the parity function that looks at the first $\log(n)$ bits. We can write a decision-tree to compute the parity. This tree is a full binary tree of depth $\log(n)$.

1.2 Learning & Cryptography

Recall that we saw that PAC-learning 3-term DNF by 3-term DNF is hard assuming RP $\neq$ NP. We sidestepped this by showing that 3-term DNF is PAC-learnable by 3CNF. This raises the question

Q: Are there classes of functions that are hard to PAC-learn *no matter* what $\mathcal{H}$ is?

If we allow *any* hypothesis class we can always produce hypotheses that perform exhaustive search on the target class $\mathcal{C}$ until they find a consistent class with the sample. Then, everything is PAC learnable. We want to forbid these shenanigans.

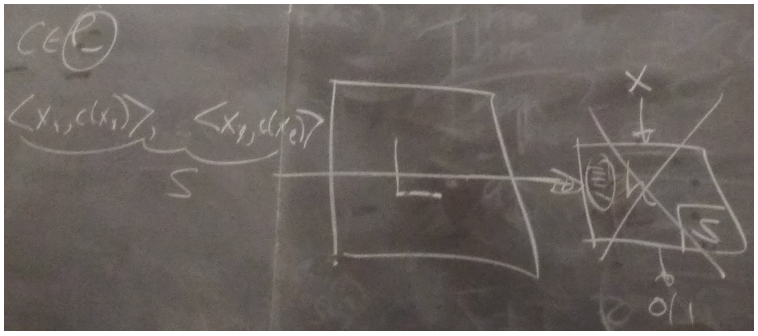

Figure 3: Overpowered hypotheses.

Q: Are there classes of functions that are hard to PAC-learn *no matter* what polynomially evaluable $\mathcal{H}$ we choose?

1.2.1 Cryptography Sidebar

- Alice wants to send Bob a message $\vec{x} \in \{0,1\}^n$ to Bob.
- Eve is listening to the conversation between them.
- Alice wants to encrypt her message.

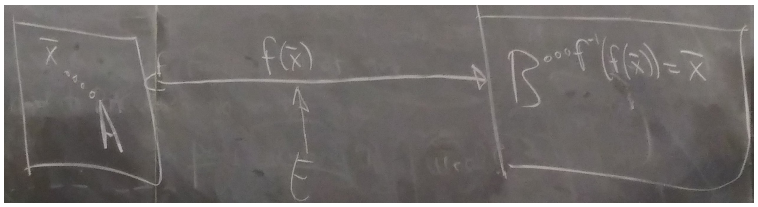

Figure 4: Cryptography layout.

One-time pad

- A & B meet privately and generate a long random bit string, $\vec{z} = z_1 \dots z_n$ where $z_i \in \{0, 1\}$.
- Then A sends B $f(\vec{x}) = \vec{y}$ where $y_i = x_i \oplus z_i$.
- B can decrypt the message by doing another XOR with $\vec{z}$ since $(x_i \oplus z_i) \oplus z_i = x_i$
- The message appears indistinguishable from a random one. However, it can only be used once because Eve can take XOR with previous encryptions to find out the message.
- Not practical in the real world because you need to meet and agree on a $\vec{z}$ beforehand. This gives rise to public key cryptography.

Eve has a learning problem from previous deciphered messages.

Public Key Cryptography

- Replace shared key/secret with *two* keys for Bob:
  1. public encryption key, e
  2. private decryption key, d
- Desired data:
  - i) encryption $f_e(\vec{x})$ is easy to compute from e
  - ii) $g_d(f_e(\vec{x})) = \vec{x}$ easy to compute from d
  - iii) But $g_d(\vec{y})$ *hard* to compute from only e, even given many encrypted/decrypted pairs (bc anybody has the public keys and can generate examples of decrypted/encrypted messages)

Example 1.7 (RSA). Bob chooses two random n-bit prime numbers, p and q and computes $N = pq$. Bob also generates a random n-bit string, e.

1. Bob's public key is (N, e)
2. To send a message x to Bob, you send $x^e \mod N$

3. From e, p and q Bob can compute a number d such that $(x^e \mod N)^d \mod N = x$. This easily satisfies *i*) and *ii*). Regarding *iii*), we only need that factorizing large integers is hard.

   Remark 1.8. *For a slight modification of RSA, breaking it is equivalent to polynomial factorization algorithms.*

Theorem 1.9 (Not exact statement)**.** *If factoring is hard (not in P), then the following classes are not PAC-learnable (even if we only ask for $\varepsilon < \frac{1}{2}$):*

- *neural networks (complex enough)*
- *DFA*
- *boolean formuale*

which are universal representations.