Lecture 13: Computational hardness in PAC learning

Recap: Classification Setting

- Instance space: $\mathcal{X}$ e.g. $\{0,1\}^d, \mathbb{R}^d$ etc., label space: $\{\mathcal{Y} = +1, 0\}$
- Hypothesis/Concept class over $\mathcal{X}$: $\mathcal{C}, \mathcal{H} \subseteq \mathcal{Y}^{\mathcal{X}}$ E.g.
  - 2d - Axis-aligned rectangles e.g.$((a_1, b_1), (a_2, b_2))$
  - CONJUNCTIONS e.g. $x_1 \wedge x_3 \wedge x_5$
  - Linear halfspaces e.g. $\sum_{i=1}^{d} w_i x_i \geq b$
- Distribition $\mathbb{P}_x$ over $\mathcal{X}$
- For any concept $c \in \mathcal{C}, \mathbb{P}_x$ Example Oracle: $\mathrm{Ex}(c; \mathbb{P}_x)$ samples $x \sim \mathbb{P}_x$ and returns $(x, c(x))$. Refer to c as the "target concept".
- Learning algorithm $\mathcal{A}$ for learning concept class $\mathcal{C}$ with hypothesis class $\mathcal{H}$ can call the example oracle $\mathrm{Ex}(c; \mathbb{P}_x)$ and must return some $h \in \mathcal{H}$. Joint distribution over randomness in $\mathrm{Ex}(c; \mathbb{P}_x)$ and algorithm is $\mathbb{P}$.
- Size of concept: A size of a concept is the minimum size over all representations in that representation scheme
  $\mathrm{size}(c) = \min_{\sigma:\rho(\sigma)=c} .\mathrm{size}(\sigma)$
- Instance size: We denote $\mathcal{X}_d$ as an instance space where all $x \in \mathcal{X}_d$ has size d.

Recap: PAC Learning

For $d \geq 1$, let $\mathcal{C}_d$ be a concept class over $\mathcal{X}_d$. Consider instance space $\mathcal{X} = \bigcup_{d=1}^{\infty} \mathcal{X}_d$ and the corresponding concept class $\mathcal{C} = \bigcup_{d=1}^{\infty} \mathcal{C}_d$.

Definition (PAC learning)

A concept class $\mathcal{C}$ is PAC learnable with hypothesis class $\mathcal{H}$ if there exists a learning algorithm $\mathcal{A}$ such that for all $d > 0$, all distributions $\mathbb{P}_x$ over $\mathcal{X}_d$, concept $c \in \mathcal{C}_d$, and $\epsilon, \delta > 0$, if $\mathcal{A}$ is given access to $\mathrm{Ex}(c; \mathbb{P}_x)$ and knows ϵ, δ,size(c), and d, $\mathcal{A}$ returns $h \in \mathcal{H}$ such that with probability at least $1 - \delta$, over inner randomisation of $\mathrm{Ex}(c; \mathbb{P}_x)$ and $\mathcal{A}$ we have that $\mathbb{P}_x[h(x) \neq c(x)] \leq \epsilon$. Further, the number of calls made to $\mathrm{Ex}(c; \mathbb{P}_x)$ should be polynomial in size(c), d, $\frac{1}{\epsilon}$, $\frac{1}{\delta}$.

Efficient PAC learnability: $\mathcal{A}$ should run in time polynomial in $\frac{1}{\epsilon}$, $\frac{1}{\delta}$, size(c), and d. Usually size(c) is bounded by some polynomial in d and hence can be ignored.

Learning k-CNF

A different approach to proving learnability — By Reduction

- Let $\mathcal{X}_d = \{0,1\}^d$, $\mathcal{Y} = \{0,1\}$
- k-CNF_d over d boolean variables.
  - Set of k-tuples $S = \{S^1, \ldots S^p\}$. $\forall i: \ S^i \subset [d]$ and $\left|S^i\right| = k$.
  - $c_S(x) = \bigwedge_{i=1}^{p}(\bigvee_{j=1}^{k} x\left[S_j^i\right])$

Theorem (learning k-CNF)

The concept class $\mathcal{C} = \bigcup_{d\geq 1} k\text{-}CNF_d$ is efficiently PAC learnable.

Proof by Reduction We need the concept of *monotone conjunctions*. *Monotone conjunctions* are conjunctions without negated literals.

- **Step 1: (Reduction)** Show these exist: instance space Z_k, a map $\phi : \{0,1\}^d \to Z_k$, and a bijection φ between k-CNF_d and monotone conjunctions on Z_k.
- **Step 2: (Algorithm)** Show that the algorithm for learning conjunctions also learns monotone conjunctions. Hence, $\mathcal{C}$ is PAC learnable.
- **Step 3: (Reduction is polynomial)** Show that ϕ, ϕ^{-1}, and φ are polynomially evaluable and thus $\mathcal{C}$ is efficiently PAC learnable.

Proof: Learning k-CNF

Step 2 is easy to prove and Step 3 will follow from step 1. So, we look at Step 1.

Step 1: (Reduction) Show these exist: instance space Z_k, a map $\phi : \{0,1\}^d \to Z_k$, and a bijection φ between k-CNF_d and monotone conjunctions on Z_k.

- Z_k: Construct new boolean variables $z_{\ell_1,\ldots,\ell_k}$ where each ℓ_i is a boolean variable or its negation in the original instance space $\mathcal{X}_d$.
- ϕ: Given an assignment to the boolean vector $x \in \{0,1\}^d$, the variable $z_{\ell_1,\ldots,\ell_k}$ is assigned a value of 1 iff $\bigvee_{i=1}^{k} x[\ell_i] = 1$.
- φ The variable $z_{\ell_1,\ldots,\ell_k}$ is present in the monotone conjunction iff there is k-tuple $\{\ell_1, \ldots, \ell_k\}$ in the set S of the k-CNF.

Consistent learner

An important concept is that of **consistent learner**. Not to be confused with the **consistent estimator** used frequently in statistics.

Definition (Consistent Learner)

An algorithm $\mathcal{A}$ is a consistent learner for concept class $\mathcal{C}$ using hypothesis class $\mathcal{H}$, if $\forall d \geq 1, c \in \mathcal{C}_d$, and for all $m \geq 1$, given as input the sequence of examples $(x_1, c(x_1)), \ldots, (x_m, c(x_m))$, $\mathcal{A}$ outputs $h \in \mathcal{H}_d$ such that $h(x_i) = c(x_i)$ for all $i = 1, \ldots, d$.

- If $\mathcal{A}$ runs in time poly in size$(c), d, m$ then $\mathcal{A}$ is an efficient consistent learner.
- This is closely related to the notion of **Empirical Risk Minimisation** with 0-1 loss.
- Thus, an efficient consistent learner can be converted to an efficient PAC learner using uniform convergence results. In fact, all the PAC learners studied so far are consistent learners.

Upper and Lower bound on sample complexity

Theorem (Upper bound for PAC Learning)

Let $\mathcal{C}$ be a concept class. Let $\mathcal{H}$ be any hypothesis class of VC dimension d_{VC}. Let $\mathcal{A}$ be a consistent learner for $\mathcal{C}$ using $\mathcal{H}$. Then for all $\epsilon, \delta > 0$, $\mathcal{A}$ is a PAC learning algorithm for $\mathcal{C}$ using $\mathcal{H}$ provided it can call $\mathrm{Ex}(c; \mathbb{P}_x)$ m times where $m \geq \kappa \left(\frac{1}{\epsilon} \log \frac{1}{\delta} + \frac{d_{VC}}{\epsilon} \log \frac{1}{\epsilon} \right)$ and κ is some universal constant.

Upper and Lower bound on sample complexity

Theorem (Upper bound for PAC Learning)

Let $\mathcal{C}$ be a concept class. Let $\mathcal{H}$ be any hypothesis class of VC dimension d_{VC}. Let $\mathcal{A}$ be a consistent learner for $\mathcal{C}$ using $\mathcal{H}$. Then for all $\epsilon, \delta > 0$, $\mathcal{A}$ is a PAC learning algorithm for $\mathcal{C}$ using $\mathcal{H}$ provided it can call $\mathrm{Ex}(c; \mathbb{P}_x)$ m times where $m \geq \kappa \left(\frac{1}{\epsilon} \log \frac{1}{\delta} + \frac{d_{VC}}{\epsilon} \log \frac{1}{\epsilon} \right)$ and κ is some universal constant.

Theorem (Lower bound for PAC Learning)

Let $\mathcal{C}$ be a concept class with VC dim $d_{VC} > 25$. Then, any PAC learning algorithm for $\mathcal{C}$ must make at least $\frac{d_{VC}-1}{32\epsilon}$ calls to $\mathrm{Ex}(c; \mathbb{P}_x)$.

For proof, see Chapter 3 in KV.

Lower bound on sample complexity for PAC learning

- Thus VC dimension exactly characterises the **statistical complexity** of PAC learning.
  - All classes with finite d_{VC} are PAC learnable. E.g. Axis Aligned rectangles, threshold functions, Finite classs (k-CNFs, k-DNFs).
  - All classes with infinite d_{VC} are not PAC learnable. e.g. Convex sets.
- This is an information-theoretic lower bound. Thus, there maybe classes with finite d_{VC} **that are PAC learnable but not efficiently**.

To prove that a certain concept class is not *efficiently* PAC learnable, we need to prove a computational hardness result which will rely on a *computational hardness assumption*. There are various such assumptions including cryptographic, complexity-theoretic etc. Today, we will use a common complexity theoretic assumption NP $\neq$ RP

Hardness of learning 3-DNFs

- Let $\mathcal{X}_d = \{0,1\}^d$, $\mathcal{Y} = \{0,1\}$
- 3-DNF_d over d boolean variables is the following hypothesis class.
  - $3\text{-DNF}_d = \{T_1 \vee T_2 \vee T_3 | T_i \in \text{CONJUNCTIONS}_d\}$
- Three conjunctions: T_1, T_2, T_3, each having at most $2d$ literals. Therefore, representation size of any $c \in 3\text{-DNF}$ is bounded by $6d$.
- VC dimension of 3-DNF_d is at most $6d$. Hence, by the above theorem, 3-DNF is PAC learnable. However, we show below it is not **efficiently** PAC learnable.

Theorem (Hardness of learning 3-DNF)

The concept class $3\text{-}DNF$ is not efficiently PAC learnable unless RP=NP.

P, NP, and RP (Basics)

The following is not a rigorous introduction but sufficient for us. Refer to the book by Arora and Barak if you are curious to learn more.

- We consider the problem of computing a boolean function f whose input is a finite sized strings of bits. This is also known as a **decision problem**.
- f can be identified with a **language** $L_f = \{\sigma \in \{0,1\}^* : f(\sigma) = 1\}$. The central question of complexity theory is how long does it take to evaluate $f(\sigma)$ i.e. whether x is in L_f, as a function of size(σ)

P, NP, and RP (Basics)

The following is not a rigorous introduction but sufficient for us. Refer to the book by Arora and Barak if you are curious to learn more.

- We consider the problem of computing a boolean function f whose input is a finite sized strings of bits. This is also known as a **decision problem**.
- f can be identified with a **language** $L_f = \{\sigma \in \{0,1\}^* : f(\sigma) = 1\}$. The central question of complexity theory is how long does it take to evaluate $f(\sigma)$ i.e. whether x is in L_f, as a function of size(σ)

- A language L is said to be in
  - **NP** if $\exists$ polynomial p and poly-time algorithm π s.t. $\forall \sigma \in \{0,1\}^*$,

$$\sigma \in L \Leftrightarrow \exists u \in \{0,1\}^{p(|\sigma|)} \text{ s.t } \pi(\sigma, u) = 1$$

P, NP, and RP (Basics)

The following is not a rigorous introduction but sufficient for us. Refer to the book by Arora and Barak if you are curious to learn more.

- We consider the problem of computing a boolean function f whose input is a finite sized strings of bits. This is also known as a **decision problem**.
- f can be identified with a **language** $L_f = \{\sigma \in \{0,1\}^* : f(\sigma) = 1\}$. The central question of complexity theory is how long does it take to evaluate $f(\sigma)$ i.e. whether x is in L_f, as a function of size(σ)

- A language L is said to be in
  - **NP** if $\exists$ polynomial p and poly-time algorithm π s.t. $\forall \sigma \in \{0,1\}^*$,

$$\sigma \in L \Leftrightarrow \exists u \in \{0,1\}^{p(|\sigma|)} \text{ s.t } \pi(\sigma, u) = 1$$

  - **RP** if $\exists$ a randomised poly-time algorithm π s.t. for all
    - $\sigma \notin L \implies \pi(\sigma) = 0$
    - $\sigma \in L \implies \pi(\sigma) = 1$ w.p. at least $\frac{1}{2}$
- Fact: **RP** $\subseteq$ **NP**. Widely believed: **RP** $\neq$ **NP**.
- A language L is **NP Complete** if L is in **NP** and every problem in **NP** can be reduced to L in poly-time.

Proof of hardness of learning 3-DNF

Theorem (Hardness of learning 3-DNF)

The concept class $3\text{-}DNF$ is not efficiently PAC learnable unless RP=NP.

Proof Strategy We will use the technique of reduction by reducing an NP-Complete language L to the problem of PAC learning 3-DNF. The proof has two steps.

- **Step 1** Choose a NP-Complete language L. Then, given any string σ **construct a dataset** $\mathcal{D}$ of +ve and -ve examples such that $\exists \phi \in$ 3-DNF where ϕ is consistent with the examples iff $\sigma \in L$.
- **Step 2** Show that if $\mathcal{D}$ can be constructed in polynomial time and there exists an efficient PAC learning algorithm for 3-DNF then **RP=NP**.

Proof of hardness of learning 3-DNF (Continued)

We will prove Step 2 first

- Consider dist $\mathbb{P}$ that assigns uniform mass over $\mathcal{D}$ and zero elsewhere. Then, run a PAC learning algorithm with $\epsilon = \frac{1}{2|\mathcal{D}|}, \delta = \frac{1}{2}$.
- If $\sigma \in L$, with probability $\frac{1}{2}$, the algorithm returns $h \in 3\text{-DNF}$ with $\mathbb{P}_x[h(x) \neq \phi(x)] \leq \frac{1}{2|\mathcal{D}|} = 0$. Thus, h is consistent with $\mathcal{D}$.
- If $\sigma \notin L$, then by definition no consistent 3-DNF exists, therefore the algorithm cannot output any such h.
- Checking whther h is consistent with
- Thus, if a PAC learning algorithm for 3-DNF exists and constructing $\mathcal{D}$ takes poly-time, then we have a randomised poly-time algorithm for solving the decision problem of whether $\sigma \in L$. This implies **RP=NP.**

Proof of hardness of learning 3-DNF (Continued)

Now we prove Step 1

We choose the language of 3-COLOURABLE graphs. A graph is 3-COLOURABLE if there is an assignment of the vertices of the graph to the set (r,g,b) s.t. no two adjacent vertices have the same color.

Given a graph G, we need to construct $\mathcal{D}$ such that $\exists \phi \in$ 3-DNF where ϕ is consistent with $\mathcal{D} \iff G$ is 3-COLOURABLE.

Given a graph $G = (V, E)$ with $|V| = d$ vertices, construct $\mathcal{D}$:

- Denote $v(i) = [\underbrace{\ldots 1 \ldots}_{0 \ldots i-1} \underbrace{0}_{i} \underbrace{\ldots 1 \ldots}_{i+1 \ldots d}] \in \{0,1\}^d$
- For $i < j$, denote $e(i,j) = [\underbrace{\ldots 1 \ldots}_{0 \ldots i-1} \underbrace{0}_{i} \underbrace{\ldots 1 \ldots}_{i+1 \ldots j-1} \underbrace{0}_{j} \underbrace{\ldots 1 \ldots}_{j+1 \ldots d}] \in \{0,1\}^d$
- Denote $\mathcal{D}_+ = \{(v(i), 1) \mid i \in V\}$, $\mathcal{D}_- = \{(e(i,j), 0) \mid (i,j) \in E\}$.
- Construct the dataset $\mathcal{D} = \mathcal{D}_+ \bigcup \mathcal{D}_-$

Proof of hardness of learning 3-DNF (Continued)

Part I: G is 3-COLOURABLE $\implies$ $\exists \phi \in$ 3-DNF consistent with $\mathcal{D}$

Suppose that G is 3-COLOURABLE. Let V_r, V_b, and V_g be the sets of vertices of respective colors. Let $z_1, \ldots, z_d$ denote the d boolean variables corresponding to the d vertices.

Let $T_r = \bigwedge_{i \notin V_r} z_i$, $T_b = \bigwedge_{i \notin V_b} z_i$, and $T_g = \bigwedge_{i \notin V_g} z_i$.

Now we need to show that $\phi = T_r \vee T_g \vee T_b$ is consistent with $\mathcal{D}$.

- For all $i \in V_r$, $z_i \notin T_r \implies T_r(v_i) = 1$. Similarly for V_b and V_g.
- For every edge $(i, j) \in E$, s.t. $i \in V_g$ and $j \in V_b$,
  - $z_i \in T_b \implies T_b(e(i,j)) = 0$,
  - $z_j \in T_g \implies T_b(e(i,j)) = 0$, and
  - $z_i, z_j \in T_r \implies T_r(e(i,j)) = 0$.

This completes this part of the proof.

Proof of hardness of learning 3-DNF (Continued)

Part II: $\exists \phi \in$ 3-DNF consistent with $\mathcal{D} \implies$ G is 3-COLOURABLE

Suppose $\phi = T_r \vee T_b \vee T_g$ is consistent with $\mathcal{D}$

To assign colours, note that $\forall i$, $\phi(v(i)) = 1 \implies \exists j \in \{r,g,b\}$ s.t. $T_j(v(i)) = 1$. Colour the vertex i with colour j. Now, we need to prove that this is a 3-colouring of the graph. We will proceed via contradiction —

- Assume that two adjacent vertices i, j are assigned the same colour (say red), then $T_r(v(i)) = T_r(v_j) = 1$.
- This implies that the literals z_i, z_j, and $\bar{z_k}$ $\forall k \neq i, j$ are not present in T_r.
- However, if T_r does not contain any negated literals except $\bar{z_i}, \bar{z_j}$ or the positive literals z_i, z_j, then $T_r(e(i,j)) = 1 \implies \phi(e(i,j)) = 1$.
- This presents a contradiction to ϕ being consistent with $\mathcal{D}$.

This proves that the coloring scheme is a 3-colouring of the graph.

Statistical Computation Trade-off

i) Computational Complexity

- Concept class k-CNF and hence 3-CNF is efficiently PAC learnable.

Statistical Computation Trade-off

i) Computational Complexity

- Concept class k-CNF and hence 3-CNF is efficiently PAC learnable.
- Under widely believed assumption RP$\neq$NP, 3-DNF is not efficiently PAC learnable.

Statistical Computation Trade-off

i) Computational Complexity

- Concept class k-CNF and hence 3-CNF is efficiently PAC learnable.
- Under widely believed assumption RP$\neq$NP, 3-DNF is not efficiently PAC learnable.

However, note that by the distributive law of boolean operations, every $\phi \in 3\text{-DNF}$ can be represented as some $\varphi \in$ 3-CNF.

Statistical Computation Trade-off

i) Computational Complexity

- Concept class k-CNF and hence 3-CNF is efficiently PAC learnable.
- Under widely believed assumption RP$\neq$NP, 3-DNF is not efficiently PAC learnable.

However, note that by the distributive law of boolean operations, every $\phi \in 3\text{-DNF}$ can be represented as some $\varphi \in$ 3-CNF.

$$\phi = T_1 \vee T_2 \vee T_3 = \bigwedge_{\ell_1 \in T_1, \ell_2 \in T_2, \ell_3 \in T_3} (\ell_1 \vee \ell_2 \vee \ell_3) = \varphi$$

Statistical Computation Trade-off

i) Computational Complexity

- Concept class k-CNF and hence 3-CNF is efficiently PAC learnable.
- Under widely believed assumption RP$\neq$NP, 3-DNF is not efficiently PAC learnable.

However, note that by the distributive law of boolean operations, every $\phi \in 3\text{-DNF}$ can be represented as some $\varphi \in$ 3-CNF.

$$\phi = T_1 \vee T_2 \vee T_3 = \bigwedge_{\ell_1 \in T_1, \ell_2 \in T_2, \ell_3 \in T_3} (\ell_1 \vee \ell_2 \vee \ell_3) = \varphi$$

Thus, we can learn to output a 3-CNF instead, which is computationally feasible. So, if we are allowed to output a CNF, then there is no problem in learning 3-DNF.

Statistical Computation Trade-off

ii) Statistical Complexity Both 3-CNF and 3-DNF have a finite VC dimension and are hence PAC learnable (inefficiently for 3-DNF).

Statistical Computation Trade-off

ii) Statistical Complexity Both 3-CNF and 3-DNF have a finite VC dimension and are hence PAC learnable (inefficiently for 3-DNF).

- **3-DNF** Using upper bound on sample complexity for PAC learning, $\phi \in 3\text{-DNF}_d$ can be inefficiently PAC learned with statistical complexity $O(\frac{d}{\epsilon})$ calls to example oracle.

Statistical Computation Trade-off

ii) Statistical Complexity Both 3-CNF and 3-DNF have a finite VC dimension and are hence PAC learnable (inefficiently for 3-DNF).

- **3-DNF** Using upper bound on sample complexity for PAC learning, $\phi \in 3\text{-DNF}_d$ can be inefficiently PAC learned with statistical complexity $O(\frac{d}{\epsilon})$ calls to example oracle.

- **3-CNF** By lower bound on sample compexity for PAC learning, any $\varphi \in 3\text{-CNF}_d$ can be efficiently PAC learned with $\Omega\left(\frac{|3\text{-CNF}_d|}{\epsilon}\right) = \Omega\left(\frac{|d^3|}{\epsilon}\right)$ calls to example oracle.

Statistical Computation Trade-off

ii) Statistical Complexity Both 3-CNF and 3-DNF have a finite VC dimension and are hence PAC learnable (inefficiently for 3-DNF).

- **3-DNF** Using upper bound on sample complexity for PAC learning, $\phi \in 3\text{-DNF}_d$ can be inefficiently PAC learned with statistical complexity $O(\frac{d}{\epsilon})$ calls to example oracle.

- **3-CNF** By lower bound on sample compexity for PAC learning, any $\varphi \in 3\text{-CNF}_d$ can be efficiently PAC learned with $\Omega\left(\frac{|3\text{-CNF}_d|}{\epsilon}\right) = \Omega\left(\frac{|d^3|}{\epsilon}\right)$ calls to example oracle.

- While 3-DNF_d could not be learned efficiently **properly**, it can be learned **efficiently improperly** with more samples — d^3 vs d.

Statistical Computation Trade-off

ii) Statistical Complexity Both 3-CNF and 3-DNF have a finite VC dimension and are hence PAC learnable (inefficiently for 3-DNF).

- **3-DNF** Using upper bound on sample complexity for PAC learning, $\phi \in 3\text{-DNF}_d$ can be inefficiently PAC learned with statistical complexity $O(\frac{d}{\epsilon})$ calls to example oracle.

- **3-CNF** By lower bound on sample compexity for PAC learning, any $\varphi \in 3\text{-CNF}_d$ can be efficiently PAC learned with $\Omega\left(\frac{|3\text{-CNF}_d|}{\epsilon}\right) = \Omega\left(\frac{|d^3|}{\epsilon}\right)$ calls to example oracle.

- While 3-DNF_d could not be learned efficiently **properly**, it can be learned **efficiently improperly** with more samples — d^3 vs d.

- Whether this statistical gap can be reduced while maintaining computational efficiency remains an open question.

Statistical Computation Trade-off

ii) Statistical Complexity Both 3-CNF and 3-DNF have a finite VC dimension and are hence PAC learnable (inefficiently for 3-DNF).

- **3-DNF** Using upper bound on sample complexity for PAC learning, $\phi \in 3\text{-DNF}_d$ can be inefficiently PAC learned with statistical complexity $O(\frac{d}{\epsilon})$ calls to example oracle.

- **3-CNF** By lower bound on sample compexity for PAC learning, any $\varphi \in 3\text{-CNF}_d$ can be efficiently PAC learned with $\Omega\left(\frac{|3\text{-CNF}_d|}{\epsilon}\right) = \Omega\left(\frac{|d^3|}{\epsilon}\right)$ calls to example oracle.

- While 3-DNF_d could not be learned efficiently **properly**, it can be learned **efficiently improperly** with more samples — d^3 vs d.

- Whether this statistical gap can be reduced while maintaining computational efficiency remains an open question.

This raises the question whether all hypothesis classes can be learned efficiently (albeit improperly). The answer is no under some cryptographic hadness assumptions.