Computational and Statistical Learning Theory

TTIC 31120

Prof. Nati Srebro

Lecture 6:

Computational Complexity of Learning—Proper vs Improper Learning

Efficient PAC Learning

- **Definition**: A family $\mathcal{H}_n$ of hypothesis classes is **efficiently properly** PAC-Learnable if there exists a learning rule A such that $\forall_n \forall \epsilon, \delta > 0$, $\exists m(n, \epsilon, \delta)$, $\forall \mathcal{D}$ s.t. $L_{\mathcal{D}}(h) = 0$ for some $h \in \mathcal{H}$, $\forall^{\delta}_{S \sim \mathcal{D}^{m(n,\epsilon,\delta)}}$,
$$L_{\mathcal{D}}\big(A(S)\big) \leq \epsilon$$
and A can be computed in time $poly(n, {}^{1}/_{\epsilon}, log\ {}^{1}/_{\delta})$
and A always outputs a predictor in $\mathcal{H}_n$

- View 1: $A(\cdot)$ outputs a program (e.g. Turing Machine description) mapping $\mathcal{X}_n \to \mathcal{Y}$ that runs in time $poly(n, {}^{1}/_{\epsilon}, log\ {}^{1}/_{\delta})$
- View 2: $A(S, x)$ or $A(\mathcal{D}, \epsilon, \delta, x)$ outputs $y = h(x)$
- View 3: Output description of h. Formally:
  - Output $w \in \{0,1\}^*$ of length $|w| \leq poly(n, {}^{1}/_{\epsilon}, log\ {}^{1}/_{\delta})$
  - w is a description of a hypothesis $h_w \in \boldsymbol{\mathcal{H}_n}$
  - There exists a poly-time algorithm $B(w, x) \mapsto h_w(x)$

Learning Using FIND-CONS

- For any family of hypothesis classes consider:
  - $\mathbf{FINDCONS}_{\mathcal{H}}$: Given a sample S, either return $h \in \mathcal{H}_n$ consistent with the sample (i.e. s.t. $L_S(h) = 0$), or declare that none exists.
  - Decision problem $\mathbf{CONS}_{\mathcal{H}}(S) = 1$ iff $\exists_{h \in \mathcal{H}_n}$ s.t. $L_S(h) = 0$.

- Theorem: If
  - $\mathrm{VCdim}(\mathcal{H}_n) \leq poly(n)$, and
  - There is a poly-time algorithm for $\mathrm{FINDCONS}_{\mathcal{H}}$

  then $\mathcal{H}_n$ is efficiently PAC learnable.

- Theorem: If $\mathcal{H}_n$ is efficiently properly PAC learnable, then
  - $\mathrm{VCdim}(\mathcal{H}_n) \leq poly(n)$, and
  - $CONS_{\mathcal{H}} \in RP$

3-TERM-DNF

- $\mathcal{H}_n = 3 - TERM - DNF_n = \{T_1 \vee T_2 \vee T_3 \mid T_i \in CONJ_n\}$

  E.g. $h(x) = \left(x_1 \wedge \overline{x[7]} \wedge x[23] \wedge x[75]\right) \vee \left(\overline{x[2]} \wedge x[3]\right) \vee (x[5] \wedge x[6] \wedge x[7])$

- $VCdim \leq \log|3 - TERM - DNF_n| \leq \log\left((3^n + 1)^3\right) = O(n)$

- Learnable with $m(n, \epsilon, \delta) \leq poly(n, {}^1/_\epsilon, \log {}^1/_\delta)$.

- Efficiently?

Hardness of 3-TERM-DNF

- Claim: $CONS_{3-TERM-DNF}$ is NP-hard

- Proof: Reduction from graph 3-colorability

  3COLOR = { undirected graph $G(V,E)$ | $\exists_{c:V\to\{1,2,3\}} \forall_{(u-v)\in E} c(u) \neq c(v)$ }

  - Map $G(V,E) \mapsto S_G$ s.t. $G \in 3COLOR \Leftrightarrow S_G \in CONS_{3-T-DNF}$
  - $S_G = \{ (x_i = (1,1,1,\dots,1,\underset{\uparrow i}{0},1,\dots,1,1), y = 1) \mid i = 1..|V| \}$

    $\cup \{ (x_{ij} = (1,1,\dots 1,\underset{\uparrow i}{0},1,1\dots,1,\underset{\uparrow j}{0},1,1\dots,1,1), y = 0 \mid (i-j) \in E \}$

  - Claim: $G \in 3COLOR \Rightarrow S_G \in CONS_{3-T-DNF}$
    - S_G satisfied by $T = T_1 \vee T_2 \vee T_3$ where $T_k = \wedge_{c(i)\neq k} x[i]$
  - Claim: $S_G \in CONS_{3-T-DNF} \Rightarrow G \in 3COLOR$
    - c(i) = min k s.t. T_k satisfies x_i

Hardness of Learning 3-TERM-DNF

- **Conclusion**: If $NP \neq RP$, then $3 - TERM - DNF$ is not efficiently properly PAC learnable

- Proof:

$$3 - TERM - DNF \text{ efficiently properly PAC learnable}$$
$$\Downarrow$$
$$CONST_{3-TERM-DNF} \in RP$$
$$\Downarrow$$
$$3COLOR \in RP$$
$$\Downarrow$$
$$RP = NP$$

Hardness of CONSISTENT

- Axis-aligned rectangles in n dimensions
- Halfspaces in n dimensions
- Conjunctions on n variables

CONSISTENT:
Poly-time
Efficiently Properly
Learnable

- 3-term DNF's
- DNF formulas of size poly(n)
- Generic logical formulas of size poly(n)
- Decision trees of size at most $\mathrm{poly}(n)$
- Neural nets with at most poly(n) units
- Functions computable in poly(n) time
- Python programs of size at most poly(n)

CONSISTENT:
NP-Hard
Not Efficiently
Properly Learnable

Improperly Learnable?

Another Look at 3-TERM-DNF

- $3-TERM-DNF_n = \{T_1 \vee T_2 \vee T_3 \mid T_i \in CONJ_n\}$

  E.g. $h(x) = \left(x[1] \wedge \overline{x[7]} \wedge x[23]\right) \vee \left(\overline{x[2]} \wedge x[3]\right) \vee (x[6] \wedge x[7])$

  Can also write: $h(x) = \left(x[1] \vee \overline{x[2]} \vee x[6]\right) \wedge \left(x[1] \vee \overline{x[2]} \vee x[7]\right) \wedge (x[1] \vee x[3] \vee x[6])$
  $\wedge (x[1] \vee x[3] \vee x[7]) \wedge \left(\overline{x[7]} \vee \overline{x[2]} \vee x[6]\right) \wedge \left(\overline{x[7]} \vee \overline{x[2]} \vee x[7]\right) \wedge \left(\overline{x[7]} \vee x[3] \vee x[6]\right)$
  $\wedge \left(\overline{x[7]} \vee x[3] \vee x[7]\right) \wedge \left(x[23] \vee \overline{x[2]} \vee x[6]\right) \wedge \left(x[23] \vee \overline{x[2]} \vee x[7]\right) \wedge (x[23] \vee x[3] \vee x[6])$
  $\wedge (x[23] \vee x[3] \vee x[7])$

- Claim: $3-TERM-DNF_n \subset 3CNF_n$

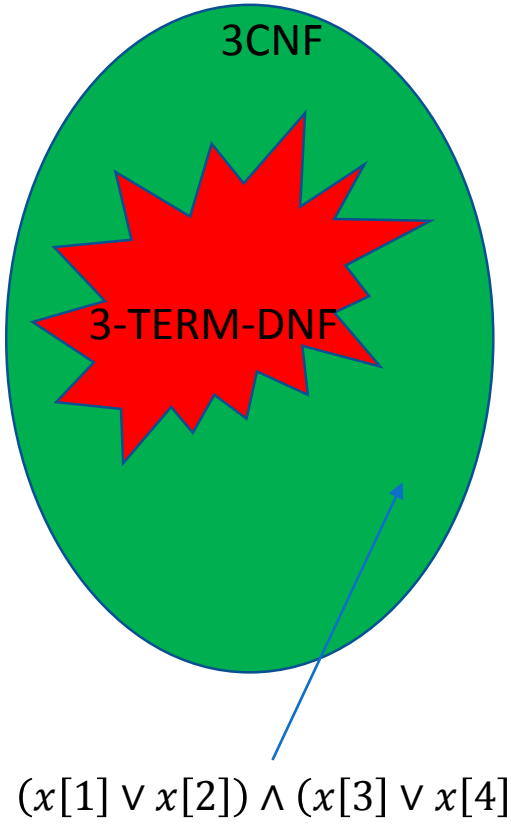
  $3CNF_n = \{ \wedge_{i=1}^{r} T_i \mid T_i = \vee \text{ of three literals}\}$

- But $3CNF_n = CONJ_{\{\text{disjunctions of three literals}\}}$
  - Can be efficiently learned using $FINDCONS_{CONJ}$, treating each $(2n)^3$ 3-disjunction as a single variable
  - I.e. map $x \mapsto \left(x[1] \vee x[2] \vee x[3],\ \ x[1] \vee x[2] \vee x[4],\ \ \ldots, x[1] \vee x[2] \vee \overline{x[3]}\right.$

Efficiently Learning 3-TERM-DNF

- Sample complexity of learning 3-TERM-DNF using $FINDCONS_{3CNF}$: $O\left(n^3/\epsilon\right)$
- Compare with cardinality bound: $O(n/\epsilon)$
- Why the gap?

- Relax statistically easy but computationally hard class to larger class that's statistically harder but computationally easier:



	$CONS_{3-T-DNF}$	$CONS_{3CNF}$
Sample complexity	$O(n/\epsilon)$	$O\left(n^3/\epsilon\right)$
Runtime	$2^{O(n)}/\epsilon$	$O\left(n^6/\epsilon\right)$

Proper vs Improper Learning

- 3-TERM-DNF is not efficiently properly learnable, but is efficiently (improperly) learnable
- Are there classes that are not efficiently learnable, even improperly?
- What about the class of all functions computable in time $poly(n)$?
  - Universal class: any $\mathcal{H}_n$ where the predictors are poly-time computable is $\mathcal{H}_n \subseteq TIME\big(poly(n)\big)$
  - $VCdim = O\big(poly(n)\big)$
  - CONS is NP-Complete:
    - If $P = NP$ we could actually learn this universal class!
    - If $RP \neq NP$, we can't efficiently **properly** learn it
  - But can we improperly learn it?
- How do we show that a class cannot be learned, no matter what hypothesis we are allowed to output?

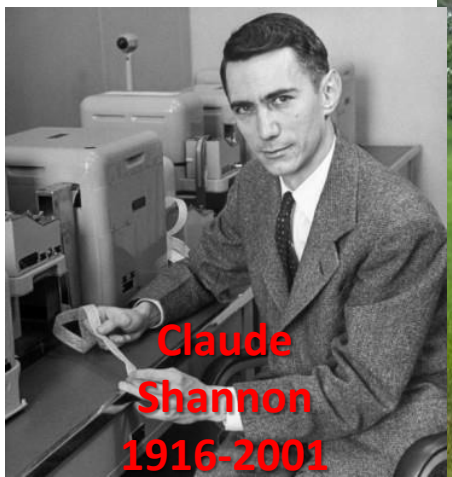
Learning and Crypto

- How to attack the cryptosystem:
  - Usually assume you know the form of f (but not K)

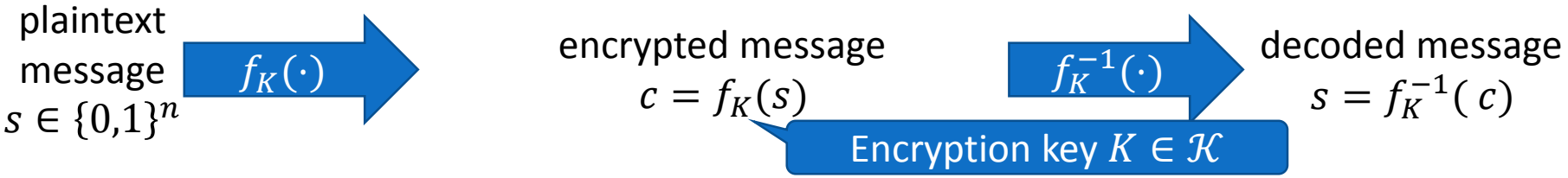
    e.g. you know target is using a GSM phone, or ssh, or file format specifies encryption, or you captured an Enigma machine
  - Collect messages for which you know the plaintext, i.e. pairs (s_i, c_i)

    … or have partial or statistical information on the plaintext s_i,

    e.g. s_i is an English word, and so $\Pr[s_i[1] = \text{"}t\text{"}] = 0.17$
  - Search for a key K s.t. $c_i = f_K(s_i)$
- Information Theoretically:
  - Need $m = \frac{\log|\mathcal{K}|}{n - H(s)}$ message pairs
  - Only way to be secure:
    One Time Pad, i.e. $\mathcal{K} = \{0,1\}^{m \cdot n}$



Learning and Crypto



- Reducing Code Breaking to Machine Learning:
  - Hypothesis class $\{ f_K^{-1} \mid K \in \mathcal{K} \}$ over domain $\mathcal{X} = \{\text{encrypted messages } c\}$ and targets $\{\text{plaintext messages } s\}$
  - Training set $\{(c_i, s_i)\}_{i=1..m}$, possibly noisy s_i
  - Learn decoder $c \mapsto s$

- Proper learning: find decoder of the form f_K^{-1}
- But really, finding any decoder that decodes most messages is fine
  ➔ Improper learning
- Statistically, enough to have $m \propto \log|\mathcal{H}| \leq \log|\mathcal{K}|$ messages
- So it better be computationally hard…

  Learning is computationally easy ➔ Breaking code is easy ➔ No crypto

  Crypto is possible ➔ learning is computationally hard

Discrete Cube Root Cryptosystem

- Choose primes p, q s.t. $3 \nmid (p-1)(q-1)$
- Public (encryption) key: $K = p \cdot q \approx 2^n$
- Private (decryption) key: $D = {}^1/_3 \mod (p-1)(q-1)$
- $f_K(s) = s^3 \bmod K$
  - easy to compute from K and s in time $O(n^2)$
- $f_K^{-1}(c) = c^D \bmod K$
  Proof: $3D = a(p-1)(q-1)+1$ for some integer a and so mod K we have $(c^D)^3 = c^{3D}$ $= (c^{p-1})^a (c^{q-1})^a c = 1^a 1^a c = c$
  - easy to compute from D and c in time $O(n^3)$
  - Hard to compute from K and c
- Hard in what sense?
  - Worst case: no poly time alg that works for every c
    maybe only hard on crazy c, but we can decrypt most messages
  - We would like: no poly time alg that works for any c
    impossible—always return 3487, you'll be right on some c
  - Enough: no poly time alg that works for random c
    can make c random by sending $f_K(s \oplus r), r$ for random r

Cryptographic Assumption

1. There exists a poly-time algorithm $\mathrm{ENC}(K, s) = f_K(s)$
2. For every $K \in \mathcal{K}$, there exists $D(K)$ and a poly-time algorithm $\mathrm{DEC}(D(K), c) = f_K^{-1}(c)$, i.e. s.t. $\forall_{s \in \mathcal{S}} DEC(D(K), f_K(s)) = s$
3. There is no poly-time (randomized) algorithm $BREAK(K, c)$ s.t. $\forall_{K \in \mathcal{K}}$

$$\Pr_{s \sim \mathrm{Unif}(\mathcal{S}), \text{randomness in } B}\left[BREAK\left(K, f_K(s)\right) = s\right] \geq \frac{1}{poly(n)}$$

$\mathcal{K}, \mathcal{S} \subseteq \{0,1\}^n$, i.e. size of inputs is $O(n)$

For discrete cube root:

- $\mathcal{K} = \{ K = (p-1)(q-1) \mid p, q \; prime, 2^{n-2} \leq K < 2^n \}$
- $\mathcal{S} = \{0,1\}^{n-2} = \{s \mid 0 \leq s < 2^{n-2}\}$

- For a useable public-key cryptosystem we also need a poly(n) algorithm for generating $\left(K, D(K)\right)$ -- we won't use this

Hardness of Learning Discrete Cube Root

$$\mathcal{H}_n = \{\, (c,i) \mapsto f_K^{-1}(c)[i] \mid K \in \mathcal{K}\,, i = 1..n-2 \,\}$$

- $\mathrm{VCdim}(\mathcal{H}_n) \leq \log|\mathcal{H}_n| \leq \log|\{0,1\}^n| = n$
- All predictors in $\mathcal{H}_n$ are computable in time $O(n^3)$
- **Claim**: Subject to discrete cube root assumption, $\mathcal{H}_n$ is not efficiently PAC learnable.
- Proof: Given poly-time learning algorithm A, construct $BREAK(K,c)$:

For i=1..n, $h_i \leftarrow A\left(\mathcal{D}_i, \epsilon = \frac{1}{4n}, \delta = \frac{1}{4n}\right)$

$D_i(x,y)$: $a \sim Unif(\{0,1\}^{n-2})$

$x = (f_K(a), i),\ y = a[i]$

Return $[h_1(c,1), h_2(c,2), \ldots, h_{n-2}(c,n-2)]$

W.p. $\geq 1 - \frac{n-2}{4n} > \frac{3}{4}$ (over randomness in $BREAK$) learning "worked" for all i, and in that case, the probability to err on some bit is $\leq \frac{n-2}{4n} < \frac{1}{4}$ (over the choice of c), and so overall w.p.>1/2, $BREAK(K,c) = f_K^{-1}(c)$.

Hardness of Learning: Conclusions

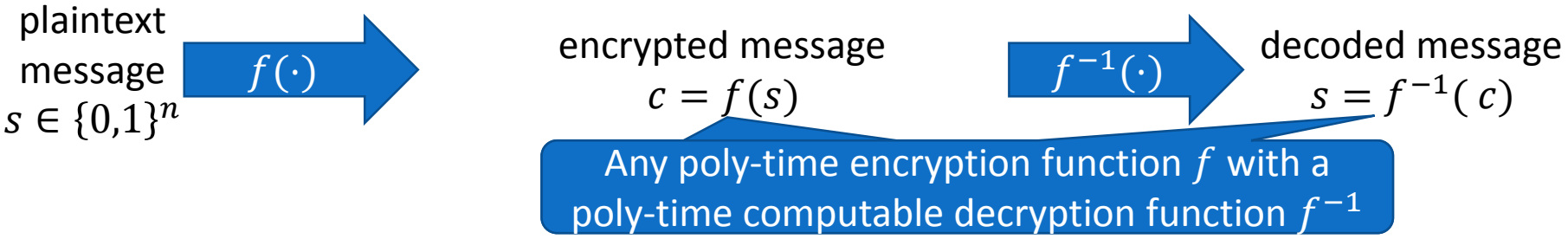
- We identified a family of functions that is provably **not** efficiently PAC-learnable
  - Despite having polynomial VC-dimension
  - Despite containing only easily computable functions
  - Subject to an assumption about security of a cryptosystem

- Since $\mathcal{H}_n \subseteq TIME\left(O(n^3)\right)$, this implies poly-time computable functions are not efficiently PAC-learnable

- In fact, if there is *any* secure public-key crypto-system, then poly-time computable functions are not efficiently PAC-learnable

- Even stronger: if we can efficiently learn poly-time computable functions then no cipher is secure to known-plaintext attacks

Learning and Crypto



- Reducing Breaking an unknown code to learning poly-time functions:
  - Hypothesis class $\{$ poly-time computable $h: \mathcal{X} \to \mathcal{Y}\}$ over domain $\mathcal{X} = \{$encrypted messages $c\}$ and targets $\{$plaintext messages $s\}$
  - Training set $\{(c_i, s_i)\}_{i=1..m}$, possibly noisy s_i
  - Learn decoder $c \mapsto s$

- Conclusion: if we can efficiently learn poly-time computable functions, we can break any cipher with a known-plaintext attack

➔ If there exists any secure cipher system, then poly-time functions are not efficiently PAC learnable

Hardness of Learning: Conclusions

- We identified a family of functions that is provably **not** efficiently PAC-learnable
  - Despite having polynomial VC-dimension
  - Despite containing only easily computable functions
  - Subject to an assumption about security of a cryptosystem

- Since $\mathcal{H}_n \subseteq TIME\left(O(n^3)\right)$, this implies poly-time computable functions are not efficiently PAC-learnable

- In fact, if there is *any* secure public-key crypto-system, then poly-time computable functions are not efficiently PAC-learnable

- Even stronger: if we can efficiently learn poly-time computable functions then no cipher is secure to known-plaintext attacks

- f_K^{-1} can be computed with circuits with $O(n^3)$ gates and depth $O(\log n)$
  - ➔ Can't efficiently PAC learn $O(\log n)$-depth circuits
  - ➔ Can't efficiently PAC lean $O(\log n)$-depth neural nets