Computational and Statistical Learning Theory

TTIC 31120

Prof. Nati Srebro

Lecture 5:

Computational Complexity of Learning

Plan

- Past two weeks:
  - learning binary predictors (classification)
  - Ignoring computational concerns
- This week:
  - understanding computational concerns
- Following weeks:
  - Beyond binary prediction: real-valued predictors (regression), multiclass, etc
  - Beyond the VC-dimension: scale sensitive, margin-based learning, regularization

PAC Learning

- **Definition**: A hypothesis class $\mathcal{H}$ is **PAC-Learnable** if there exists a learning rule A such that $\forall \epsilon, \delta > 0, \exists m(\epsilon, \delta), \forall \mathcal{D}$ s.t. $L_{\mathcal{D}}(h) = 0$ for some $h \in \mathcal{H}$, $\forall^{\delta}_{S \sim \mathcal{D}^{m(\epsilon,\delta)}}$,

$$L_{\mathcal{D}}\big(A(S)\big) \leq \epsilon$$

- **Theorem**: $\mathcal{H}$ is PAC-Learnable **if and only if** $\mathrm{VCdim}(\mathcal{H}) < \infty$
- E.g. $\mathcal{H} = \{ h \; computed \; by \; program \; w \mid |w| \leq 1000 \}$

- **Definition**: A hypothesis class $\mathcal{H}$ is **non uniformly learnable** if there exists a learning rule A such that $\forall \epsilon, \delta > 0 \forall_{h \in \mathcal{H}}, \exists m(\epsilon, \delta, h), \forall \mathcal{D}$ s.t. $L_{\mathcal{D}}(h) = 0$, $\forall^{\delta}_{S \sim \mathcal{D}^{m(\epsilon,\delta,h)}}$,

$$L_{\mathcal{D}}\big(A(S)\big) \leq \epsilon$$

- **Theorem**: $\mathcal{H}$ is non unfirmly learnable **if and only if** it is a countable union of finite VC-dim hypothesis classes
- E.g. all computable functions

Minimum Description Length Learner

- Task: predict y from x
- Input: labeled training set $S = \{(x_1, y_1), (x_2, y_2), \ldots\}$

- `Return shortest program` $p: x \mapsto y$ `s.t.` $p(x_i) = y_i$ `for all` $(x_i, y_i) \in S$

Choosing the Description Language

- Decision Trees? Polynomials? Python? Java?

- For a description language c, let $|h|_c = \min_{c(p)=h} |p|$
- How different are $|h|_{\text{Python}}, |h|_{\text{Java}}, |h|_{\text{TuringMachine}}$?

Length of Java interpreter written in Python

- Claim: $|h|_{\text{Python}} \leq |h|_{\text{Java}} + \overbrace{|\text{Java}|_{\text{Python}}}$

- For any computable $c: \mathcal{U} \to \mathcal{Y}^{\mathcal{X}}$ over a prefix-unambiguous $\mathcal{U} \subset \{0,1\}^*$ (i.e. s.t. $p, x \mapsto c(p)(x)$ is computable), there exists a constant C s.t. $|h|_{\text{Python}} \leq |h|_c + C$

MDL / SRM Learner

- Task: predict y from x
- Input: labeled training set $S = \{(x_1, y_1), (x_2, y_2), \ldots\}$

- Split training set to S_{tr}, S_{val}
- For each length L:
  - h_L = program $p: x \mapsto y$ of length $|p| \leq L$ with minimum error on S_{tr}
- Return h_L with minimum error on S_{val}

- Is this as good as any other computable learning rule?
- Can compete with "parametric" learning, outputting a compact predictor.
- But what about nearest neighbor?

SRM++ (Ultimate MDL) Learner

- Task: predict y from x
- Input: labeled training set $S = \{(x_1, y_1), (x_2, y_2), \dots\}$

- Split training set to S_{tr}, S_{ref}, S_{val}
- For each length L:
  - h_L =program $p: (S_{ref}, x) \mapsto y$ of length $|p| \leq L$ with minimum error on S_{tr} (when it can use S_{ref} as input)
- Return h_L with minimum error on S_{val}

- Theoretically: only a constant more training examples compared to any computable learning rule (homework)

- "In Practice": beats any learning method out there

Efficient PAC Learning

- **Definition (attempt)**: A hypothesis class $\mathcal{H}$ is **efficiently PAC-Learnable** if there exists a **poly-time computable** learning rule A such that $\forall \epsilon, \delta > 0$, $\exists m(\epsilon, \delta)$, $\forall \mathcal{D}$ s.t. $L_{\mathcal{D}}(h) = 0$ for some $h \in \mathcal{H}$, $\forall^{\delta}_{S \sim \mathcal{D}^{m(\epsilon,\delta)}}$,
$$L_{\mathcal{D}}\big(A(S)\big) \leq \epsilon$$
- Runtime polynomial in what?
- A(S) polynomial in S:
  - Can inflate m. If we really need only $m(\epsilon, \delta)$ samples, but computing $A(\cdot)$ is requires exponential runtime, instead ask for $m'(\epsilon, \delta) = 2^{m(\epsilon,\delta)}$ samples.
- Runtime polynomial in $1/\epsilon$, $1/\delta$:
  - Consider $\mathcal{H} = \{$ decision trees with ≤ 1000 nodes $\}$ or $\mathcal{H} = \{ 7 -$ layer Nueal Nets over $\mathbb{R}^d$ with d units per layer $\}$
  - For each d, runtime is $poly(1/\epsilon, \log 1/\delta)$
- What we really want is "polynomial in the size of the problem"

Efficient PAC Learning

- Consider $\mathcal{X}_n = \{0,1\}^n$ (or perhaps $\mathcal{X}_n = \mathbb{R}^n$) and $\mathcal{Y} = \{\pm 1\}$
- We will study a **family** of hypothesis classes $\{\mathcal{H}_n\}_{n=1}^{\infty}$, $\mathcal{H}_n \subseteq \mathcal{Y}^{\mathcal{X}_n}$.
- Definition: A family $\mathcal{H}_n$ of hypothesis classes is **efficiently 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 $\boldsymbol{A}$ **can be computed in time** $\boldsymbol{poly}(\boldsymbol{n}, {}^{\boldsymbol{1}}/_{\boldsymbol{\epsilon}}, \boldsymbol{log}\, {}^{\boldsymbol{1}}/_{\boldsymbol{\delta}})$
- In particular, this implies $m(n, \epsilon, \delta) \leq poly(n, {}^{1}/_{\epsilon}, \log {}^{1}/_{\delta})$
- Alternative view: instead of algorithm $A(S)$ taking S as input, algorithm $A(\mathcal{D}, \epsilon, \delta)$, that is allowed to sample from $\mathcal{D}$ in unit time.
- What does the algorithm output?

What Does the Algorithm Output?

- Definition: A family $\mathcal{H}_n$ of hypothesis classes is **efficiently 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 $\boldsymbol{A}$ **can be computed in time** $\boldsymbol{poly(n, {}^{1}/_{\epsilon}, log\, {}^{1}/_{\delta})}$

- 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 (e.g. a decision tree). Formally:
  - Output $w \in \{0,1\}^*$ of length $|w| \leq poly(n, {}^{1}/_{\epsilon}, log\, {}^{1}/_{\delta})$
  - There exists a poly-time algorithm $B(w, x) \mapsto h_w(x)$

What can we learn efficiently?

- For now—realizable setting (assuming there exists a consistent predictor in the hypothesis class)

- Linear Predictors
  - Using an LP feasibility problem:
    Find w s.t. $\forall_i \; y_i \langle w, \phi(x_i) \rangle > 1$
- Polynomials
  - As linear predictors over expanded feature space
- Axis Aligned Rectangles
  - By finding minimal enclosing rectangle of positive points
- Conjunctions of literals

Learning Conjunctions

$\mathcal{X}_n = \{0,1\}^n$ $\quad$ $\mathcal{H}_n = CONJ_n$ =conjunction of literals

- E.g. $h(x) = x[5] \wedge \overline{x[7]} \wedge x[12]$
- $VCdim(CONJ_n) \leq \log|CONJ_n| = \log(3^n + 1) = O(n)$
- Can learn with $m = O\left(\frac{n+\log\frac{1}{\delta}}{\epsilon}\right)$ samples. Efficiently?

Input: $S = \{(x_1, y_1), \ldots, (x_m, y_m)\}$
Initialize $h = x[1] \wedge \overline{x[1]} \wedge x[2] \wedge \overline{x[2]} \wedge \cdots \wedge x[n] \wedge \overline{x[n]}$
For $t = 1..m$,
    If $y_t = 1$,
        For $i = 1..n$,
            If $x_t[i] = 1$ remove $\overline{x[i]}$ from h
            If $x_t[i] = 0$ remove $x[i]$ from h

- Claim: If S is consistent with $CONJ_n$, returns h consistent with sample
- Proof sketch: when we remove a literal, it cannot be in any consistent conjunction, hence we recover maximal conjunction consistent with all positive examples. If there exists some consistent conjunction, this maximal conjunction must also be consistent with all negatives.

Learning Using FIND-CONS

- For any family of hypothesis classes consider the "find consistent" problem $\mathrm{FINDCONS}_{\mathcal{H}}$:
  - 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 no consistent $h \in \mathcal{H}_n$ exists.

- Claim: If
  - $\mathrm{VCdim}(\mathcal{H}_n) \leq poly(n)$, and } **required**
  - There is a poly-time algorithm for $\mathrm{FINDCONS}_{\mathcal{H}}$ } **???** (polynomial in size of input)

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

- Conclusion: CONJ_n is efficiently PAC learnable

- **Converse?**

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?

- $FINDCONST_{3-TERM-DNF}$ is NP-hard. What does this imply?

Proper 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 $\boldsymbol{A}$ **always outputs a predictor in** $\boldsymbol{\mathcal{H}_n}$

- 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 $B(w, x) \mapsto h_w(x)$

Hardness of Proper Learning

- For a family $\mathcal{H}_n$ consider the **decision** problem $CONS_{\mathcal{H}}(S) = 1$ iff $\exists_{h \in \mathcal{H}_n}$ s.t. $L_S(h) = 0$.

- Theorem: if $\mathcal{H}_n$ is efficiently properly PAC learnable, then $CONS_{\mathcal{H}} \in RP$
  - i.e. $\exists$ randomized algorithm C s.t. on any input S:
    - $\Pr[\, C(S) = 1 \mid CONS(S) = 1 \,] \geq 3/4$
    - $\Pr[\, C(S) = 0 \mid CONS(S) = 0 \,] = 1$

- Proof: given learning algorithm $A(\mathcal{D}, \epsilon, \delta)$ construct C as follows—

  1. $h \leftarrow A\left(uniform\ over\ S, \frac{1}{8}, \frac{1}{2|S|}\right)$
  2. Check if $L_S(h) = 0$ by evaluating h on each point in S
  3. If yes, output 1, otherwise output 0

  - If $CONS_{\mathcal{H}}(S) = 0$, we certainly output 0 (no $h \in \mathcal{H}_n$ will have $L_S(h) = 0$)
  - If $CONS_{\mathcal{H}}(S) = 1$, w.p. $\geq 7/8$, $L_{\mathcal{D}}(h) = L_S(h) \leq \frac{1}{2|S|}$, i.e. $L_S(h) = 0$
  - Runtime polynomial in $n|S|$