Lecture 5:
Computational Complexity of Learning
PAC-Bayes

• For any “prior” $P$ over hypothesis and any $\mathcal{D}$, $\forall_{S \sim \mathcal{D}^m}, \forall Q$

$$L_{\mathcal{D}}(h_Q) \leq L_S(h_Q) + \sqrt{\frac{KL(Q||P) + \log \frac{2m}{\delta}}{2(m-1)}}$$

• Minimizing the bound amounts to:

$$Q_\lambda = \arg\min_Q L_S(h_Q) + \lambda \cdot KL(Q||P)$$

• **Theorem:**

$$q_\lambda(h) \propto p(h)e^{-\beta L_S(h)}$$

for some “inverse temperature” $\beta$ that depends on $\lambda$, and also $P, \mathcal{D}$
PAC-Bayes vs Bayes

Bayesian approach:

- Assume $h \sim \mathcal{P}$,
- $x_1, \ldots, x_m$ iid from some $p(x)$
- $y_1, \ldots, y_m$ independent conditioned on $h$, with
  $$y_i | x_i, h = \begin{cases} h(x_i), & \text{w. p. } 1 - \nu \\ -h(x_i), & \text{w. p. } \nu \end{cases}$$

Bayesian posterior:

$$p(h|S) \propto p(h)p(S|h) \propto p(h) \prod_i \left( \frac{\nu}{1-\nu} \right)^{[h(x_i) \neq y_i]}$$

$$= p(h) e^{-\beta L_S(h)}$$

where $\beta = m \log \frac{1-\nu}{\nu}$
PAC-Bayes vs Bayes

PAC-Bayes

- $P$ encodes inductive bias, not assumption about reality

- SRM-type bound minimized by Gibbs distribution
  
  $q_\lambda(h) \propto p(h) e^{-\beta L_S(h)}$

- Post-hoc guarantee always valid ($\forall_\delta$), with no assumption about reality

\[
L_D(h_Q) \leq L_S(h_Q) + \sqrt{\frac{KL(Q||P) + \log \frac{2m}{\delta}}{2(m - 1)}}
\]

- If inductive bias very different from reality, bound will be high

- Bound valid for any $Q$

Bayesian Approach

- $P$ is prior over reality

- Posterior given by Gibbs distribution
  
  $q_\lambda(h) \propto p(h) e^{-\beta L_S(h)}$

- Risk analysis assuming prior
For any distribution $P$ over hypothesis and any source distribution $\mathcal{D}$, $\forall \delta \in \mathcal{D}^m$:

$$KL\left( L_S(h_Q) \parallel L_D(h_Q) \right) \leq \frac{KL(Q\parallel P) + \log \frac{2m}{\delta}}{m-1}$$

where $KL(\alpha \parallel \beta) = \alpha \log \frac{\alpha}{\beta} + (1-\alpha) \log \frac{1-\alpha}{1-\beta}$ for $\alpha, \beta \in [0,1]$.

\[
L_D(h_Q) \leq L_S(h_Q) + \sqrt{\frac{2L_S(h_Q)KL(QP)+\log \frac{2m}{\delta}}{m-1}} + \frac{2(KL(QP)+\log \frac{2m}{\delta})}{m-1}
\]

Can also be used as a tail bound instead of Hoeffding or Bernstein also with cardinality or VC-based guarantees. Arises naturally in PAC-Bayes.
Universal Learning using MDL/SRM
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


- For a description language $c$, let $|h|_c = \min_{c(p)=h} |p|$

- How different are $|h|_{\text{Python}}, |h|_{\text{Java}}, |h|_{\text{Turing Machine}}$?

  Length of Java interpreter written in Python

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

- For any computable $c: \mathcal{U} \rightarrow \mathcal{Y}^X$ 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. $|h|_{\text{Python}} \leq |h|_c + C$
Universal MDL 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 $(x_i, y_i) \in S$
Universal 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 $L$:
  - $h_L = \text{program } p: x \mapsto y \text{ of length } |p| \leq L$ 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 Learner

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

• Split training set to $S_{tr}, S_{ref}, S_{val}$
• 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

• “In Practice”: beats any learning method out there
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 (and relation to PAC-Bayes)

• After that:
  • Optimization, online learning, stability
• Question (Harvard, 1984): What is PAC-Learnable?

• Answer (Moscow, 1971): $\mathcal{H}$ learnable iff it has finite VC-dimension

• Valiant’s actual Question: What is efficiently PAC learnable?
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 $\forall \epsilon, \delta > 0$, $\exists m(\epsilon, \delta)$, $\forall D$ s.t. $L_D(h) = 0$ for some $h \in \mathcal{H}$, $\forall S \sim_D m(\epsilon, \delta)$,
  $$L_D(A(S)) \leq \epsilon$$

- 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 $\frac{1}{\epsilon}, \frac{1}{\delta}$:
  - Consider $\mathcal{H} = \{ \text{decision trees with } \leq 1000 \text{ nodes} \}$ or $\mathcal{H} = \{ \text{7 - layer Nueal Nets over } \mathbb{R}^d \text{ with } d \text{ units per layer} \}$
  - For each $d$, runtime is $poly(\frac{1}{\epsilon}, \log \frac{1}{\delta})$

- What we really want is “polynomial in the size of the problem”
Efficient PAC Learning

• We will study a family of hypothesis classes \( \{\mathcal{H}_n\}_{n=1}^{\infty} \), over \( \{\mathcal{X}_n\}_{n=1}^{\infty} \mathcal{H}_n \subseteq \mathcal{Y}^{\mathcal{X}_n} \).

• Usually \( \mathcal{X}_n = \{0,1\}^n \) (or perhaps \( \mathcal{X}_n = \mathbb{R}^n \)) and \( \mathcal{Y} = \{\pm 1\} \)

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

\[
L_{\mathcal{D}}(A(S)) \leq \epsilon
\]

and \( A \) can be computed in time \( \text{poly}(n, 1/\epsilon, \log 1/\delta) \)

• In particular, this implies \( m(n, \epsilon, \delta) \leq \text{poly}(n, 1/\epsilon, \log 1/\delta) \)

• 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. $L_D(h) = 0$ for some $h \in \mathcal{H}_n$, $\forall_{S \sim \mathcal{D}}^\delta m(n, \epsilon, \delta)$, $L_D(A(S)) \leq \epsilon$

and $A$ can be computed in time $\text{poly}(n, \frac{1}{\epsilon}, \log \frac{1}{\delta})$

- View 1: $A(\cdot)$ outputs a program (e.g. Turing Machine description) mapping $\mathcal{X}_n \rightarrow \mathcal{Y}$ that runs in time $\text{poly}(n, \frac{1}{\epsilon}, \log \frac{1}{\delta})$

- 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 $|w| \leq \text{poly}(n, \frac{1}{\epsilon}, \log \frac{1}{\delta})$
  - 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 over $\mathcal{X}_n = \mathbb{R}^n$
  $\mathcal{H}_n = \{\text{sign}(\langle w, x_i \rangle) | w \in \mathbb{R}^n\}$
  • Using an LP feasibility problem:
    Find $w$ s.t. $\forall_i y_i \langle w, \phi(x_i) \rangle > 1$

• Degree 7 polynomials over $\mathcal{X}_n = \mathbb{R}^n$
  $\mathcal{H}_n = \{\text{sign}(p(x)) | \text{poly } p \text{ of degree } \leq 7 \}$
  • As linear predictors over expanded feature space

• Axis Aligned Rectangles
  • By finding minimal enclosing rectangle of positive points

• Conjunctions of literals over $\{\pm 1\}^n$
  E.g. $h(x) = x[5] \land \overline{x[7]} \land x[12]$
Learning Conjunctions

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

- E.g. \( h(x) = x[5] \land x[7] \land x[12] \)

- \( \text{VCdim}(CONJ_n) \)

- 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 \( h = x[1] \land x[1] \land x[2] \land x[2] \land \cdots \land x[n] \land x[n] \)

For \( t = 1..m, \)

If \( y_t = 1, \)

For \( i = 1..n, \)

If \( x_t[i] = 1 \) remove \( x[i] \) 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 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
  • VCdim($\mathcal{H}_n$) ≤ poly($n$), and } required
  • There is a poly-time algorithm for 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

- $\mathcal{H}_n = 3 - \text{TERM} - \text{DNF}_n = \{T_1 \lor T_2 \lor T_3 \mid T_i \in \text{CONJ}_n\}$
  
  E.g. $h(x) = (x_1 \land \overline{x[7]} \land x[23] \land x[75]) \lor (\overline{x[2]} \land x[3]) \lor (x[5] \land x[6] \land x[7])$

- $\text{VCdim}$

- Learnable with $m(n, \epsilon, \delta) \leq \text{poly}(n, \frac{1}{\epsilon}, \log \frac{1}{\delta})$. Efficiently?

- $\text{FINDCONST}_{3-\text{TERM}-\text{DNF}}$ is NP-hard.
  
  $\Rightarrow$ Likely no efficient (poly-time) algorithm for $\text{FINDCONST}_{3-\text{TERM}-\text{DNF}}$

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 $h \in \mathcal{H}_n$, $\forall S \sim \mathcal{D}^{m(n,\epsilon,\delta)}$, $L_{\mathcal{D}}(A(S)) \leq \epsilon$

and $A$ can be computed in time $\text{poly}(n, \frac{1}{\epsilon}, \log \frac{1}{\delta})$

and $A$ always outputs a predictor in $\mathcal{H}_n$

- **View 3**: Output description of $h$. Formally:
  - Poly-time algorithm $B(w, x) \mapsto h_w(x)$, where $h_w \in \mathcal{H}_n$
  - Output $w \in \{0,1\}^*$ of length $|w| \leq \text{poly}(n, \frac{1}{\epsilon}, \log \frac{1}{\delta})$
  - $w$ is a description of a hypothesis $h_w \in \mathcal{H}_n$
Hardness of Proper Learning

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

Over $\mathcal{X}_n = \{0,1\}^n$ or $\mathcal{X}_n = \mathbb{R}^n$:

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

- Proof: given learning algorithm $A(\mathcal{D}, \epsilon, \delta)$ construct $C$ as follows—
  1. $h \leftarrow A\left(\text{uniform over } S, \frac{1}{2|S|}, \frac{1}{8}\right)$
  2. Check if $L_S(h) = 0$ by evaluating $h$ on each point in $S$
  3. If yes, output 1, otherwise output 0

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