Lecture 24: PAC Learning

Handwritten Notes - Converted to LaTeX

November 21, 2025

PAC Learning

How do we generalize machine learning? Broadly speaking, **Supervised Learning** is about coming up with an algorithm or suite of algorithms that, given some initial training (labeled data), outputs a "prediction algorithm" that does well on future data that is unlabeled. The most basic but still very useful and important problem is **binary classification** (Yes or No).

Set Up

1. **Data** is formalized as vectors $\mathbf{x} \in X$.
2. **Labeled data**: pairs $(\mathbf{x}_1, y_1), \ldots, (\mathbf{x}_n, y_n)$ where y_i is the correct label for data $\mathbf{x}_i$.
3. Want to obtain a **prediction algorithm** (could be randomized) $A(\mathbf{x})$ that, given a future data point $\mathbf{x}$, outputs a label.

Since arbitrary algorithms are complicated and hard to reason about and interpret and optimize over, traditionally the set of prediction algorithms (or **hypotheses** $\mathcal{H}$) were simple. The term **concept class** $\mathcal{C}$ is used to indicate that we are only interested in finding a "prediction" algorithm from this class.

The PAC Model

The PAC (Probably Approximately Correct) model, introduced by Valiant, is a nice theoretical model that addresses the issues of generalization, sample complexity, and efficiency of learning.

4. In the full consistency model we will assume that $\exists$ some $c^* : X \to \{0, 1\}$ which gives the true label for each example. This c is called a **concept**.
5. The examples are drawn from a **distribution** D.

Given a hypothesis $h \in \mathcal{H}$, we define its **error** with respect to c^* and D as:

$$\mathrm{err}_D(h) = \Pr_{\mathbf{x} \sim D}[h(\mathbf{x}) \neq c^*(\mathbf{x})]$$

PAC Learnability Definition

Definition: A pair $(\mathcal{C}, \mathcal{H})$ is **PAC-Learnable** if $\exists$ a learning algorithm that, given $m = \mathrm{poly}(1/\epsilon, 1/\delta)$ random samples from D, outputs (efficiently) a hypothesis h such that $\Pr_{\mathcal{S}}[\mathrm{err}_D(h) > \epsilon] \leq \delta$.

(i) The algorithm is allowed to output an imperfect hypothesis even when there exists a perfect concept.

(ii) The algorithm is allowed to fail with some small probability δ.

Consistency and Generalization Bounds for Finite $\mathcal{H}$

Consistency

Definition: Given a set of correctly labeled examples $S = \{(\mathbf{x}_1, y_1), \ldots, (\mathbf{x}_n, y_n)\}$ (where $y_i = c^*(\mathbf{x}_i)$), a hypothesis $h : X \to \{0, 1\}$ is **consistent** if $h(\mathbf{x}_i) = y_i$ for all i.

Theorem for Finite Hypothesis Classes (Consistent Case)

Theorem: Let $\mathcal{H}$ be a finite hypothesis class and let $\epsilon, \delta \in (0,1)$. Suppose the sample size is:

$$n \geq \frac{1}{\epsilon}\left(\ln|\mathcal{H}| + \ln\frac{1}{\delta}\right)$$

Then, if h is consistent with S, then $\mathrm{err}_D(h) \leq \epsilon$ with probability $1-\delta$.

Proof Sketch: Say hypothesis h is **bad** if $\mathrm{err}_D(h) > \epsilon$. Let $h_1, h_2, \ldots, h_l$ be the bad hypotheses. The probability that $\bar{h}_i$ is consistent with S is at most $(1-\epsilon)^n$. By the Union Bound, if $l(1-\epsilon)^n < \delta$, then no bad hypothesis will be consistent, so the output of the algorithm will be a good hypothesis. This condition is satisfied by the required n since $l \leq |\mathcal{H}|$.

Theorem for Finite Hypothesis Classes (Agnostic Case)

We may only have a "good" hypothesis $h^* \in \mathcal{H}$ that makes a small error $\mathrm{err}_D(h^*)$. We find a hypothesis h that makes the sample error $\mathrm{err}_S(h)$ as small as possible.

Theorem: Let $\mathcal{H}$ be a finite hypothesis class and let $\epsilon, \delta \in (0,1)$. Suppose the sample size is:

$$n \geq \frac{1}{2\epsilon^2}\left(\ln|\mathcal{H}| + \ln\frac{2}{\delta}\right)$$

Then, with probability $1-\delta$, for all $h \in \mathcal{H}$:

$$|\mathrm{err}_S(h) - \mathrm{err}_D(h)| \leq \epsilon$$

Proof: Needs additive Chernoff bound.

VC-Dimension Based Bounds

Consider the setting where $X \subseteq \mathbb{R}^d$ and $\mathcal{H}$ is the class of half-spaces in $\mathbb{R}^d$. Here, $|\mathcal{H}|$ is infinite.

Theorem: Let $\mathcal{H}$ be a family of hypotheses with **VC-dimension** d_{VC}. Let $\epsilon, \delta \in (0,1)$.

1. If $n \geq \frac{c}{\epsilon}\left(d_{VC}\ln\frac{d_{VC}}{\epsilon} + \ln\frac{1}{\delta}\right)$ for some constant c, then, with probability $1-\delta$, any consistent hypothesis h has $\mathrm{err}_D(h) \leq \epsilon$.

2. If $n \geq \frac{1}{\epsilon^2}\left(d_{VC}\ln\frac{d_{VC}}{\epsilon} + \ln\frac{1}{\delta}\right)$, then $\forall h \in \mathcal{H}, |\mathrm{err}_D(h) - \mathrm{err}_S(h)| \leq \epsilon$.

Examples

Disjunctions

Let $X = \{0,1\}^n$. $\mathcal{C}$ is the class of **disjunctions** of Boolean literals (e.g., $z_1 + z_5 + \overline{z_7}$). The size of the concept class is $|\mathcal{C}| = 3^n$.

Claim: Given a set of examples $S = \{(\mathbf{x}_i, y_i)\}$, there is an **efficient algorithm** that outputs a disjunction $c \in \mathcal{C}$ that is consistent with S if one exists.

Algorithm Outline:

1. Start with $c = (z_1 + \overline{z_1}) + \cdots + (z_n + \overline{z_n})$.

2. Let S_0 be examples with $y_i = 0$ and S_1 be examples with $y_i = 1$.

3. **Prune based on S_0 (Negative Examples):** For each $i \in S_0$, remove literals from c that are satisfied by $\mathbf{x}_i$.

4. **Check S_1 (Positive Examples):** For each $i \in S_1$, check if at least one literal in c is satisfied by $\mathbf{x}_i$. If not, output "no consistent hypothesis".

Figure 1: Example Data Table

Example	**Data (x)**	**Label (y)**
$\mathbf{x}_1$	$(0, 0, 1, 0, 1)$	$y_1 = 0$
$\mathbf{x}_2$	$(1, 0, 1, 0, 1)$	$y_2 = 0$
$\mathbf{x}_3$	$(1, 1, 0, 0, 0)$	$y_3 = 0$
$\mathbf{x}_4$	$(1, 1, 0, 0, 0)$	$y_4 = 1$
$\mathbf{x}_5$	$(0, 0, 0, 0, 0)$	$y_5 = 1$

Half-Spaces in $\mathbb{R}^d$

$\mathcal{C}$ is the class of half-spaces in $\mathbb{R}^d$. Finding a consistent hypothesis is equivalent to finding a hyperplane $\mathbf{a} \cdot \mathbf{x} - b = 0$ that separates positive examples (S_1) from negative examples (S_0).

This can be formulated as a **Linear Program (LP)** to find the coefficients $a_1, \ldots, a_d, b$:

- Constraints for $i \in S_1$ (positive examples):

$$\sum_{j=1}^{d} a_j x_{i,j} - b > 0$$

- Constraints for $i \in S_0$ (negative examples):

$$\sum_{j=1}^{d} a_j x_{i,j} - b \leq 0$$

The LP is feasible if and only if a separating half-space exists.