CS272 - Foundations of Learning, Decisions, and Games

Lecture 2: Statistical learning I

September 2, 2025

Lecturer: Nika Haghtalab *Readings: Chp 2, UML*
Scribe: Nika Haghtalab

As a motivating example, consider going to the farmer's market. You want to identify delicious apples from the bad ones. Let's make a simplifying assumption that only three features of an apple affect its deliciousness: color (green/red), firmness (soft/crunchy), and size (small/medium/large). In the past, you have had a several apples from the market and you have diligently recorded the features of every apple you ate and their level of tastiness. Your goal is to use this historical record to learn to identify tasty apples from the non-tasty ones. That is, given a new apple that you haven't yet tasted, predict whether this apple is tasty.

For today's lecture, we make a simplifying assumption that there is an unknown mapping from apples to labels, denoted by $c : \{green, red\} \times \{soft, crunchy\} \times \{small, medium, large\} \to \{\text{tasty}, \text{not tasty}\}$, that perfectly determines the deliciousness of an apple. Such a mapping is called a *concept*. A collection of concepts is called a *concept class*. We assume that the concept c belongs to some known concept class $\mathcal{C}$ that is pre-determined. Our goal is to learn c or a close approximation of it, so that we can near-perfectly identify all delicious apples in the market.

How do you learn c? This is what we discuss in this lecture.

1 Formal Model

Let us formally define notations that will be used in this lecture and many of the following lectures.

- **Domain (Instance space):** An arbitrary set $\mathcal{X}$ that includes all possible instances, e.g., apples, that the learner may wish to label. An instance is typically described by a vector of values, representing the relevant feature values. For example, and apple can be described by a feature vector $(green, crunchy, medium)$. In this case, the domain can be the set of all possible feature vectors, i.e., $\mathcal{X} := \{green, red\} \times \{soft, crunchy\} \times \{small, medium, large\}$. An $x \in \mathcal{X}$ is called an *instance*.

- **Labels:** A set $\mathcal{Y}$ that includes all possible labels or predictions for a single instance. In the apple example, we have $\mathcal{Y} = \{\text{tasty}, \text{not tasty}\}$. For simplicity, this course works with 2-element label sets, which we usually refer to as $\{0, 1\}$, $\{-1, 1\}$, $\{false, true\}$, etc. For ease of presentation, in the apple example we refer to tasty, 1, $true$ interchangeably.

- **Labeled instance:** An instance-label pair $(x, y) \in \mathcal{X} \times \mathcal{Y}$ is called a labeled instance.

- **Concept:** A *concept* (later on will also be called a classifier or predictor or hypothesis) is a function $c : \mathcal{X} \to \mathcal{Y}$. For example, the concept $(color = red) \vee (size \neq small)$ assigns to any apple that is red or is not small, the label $true$.

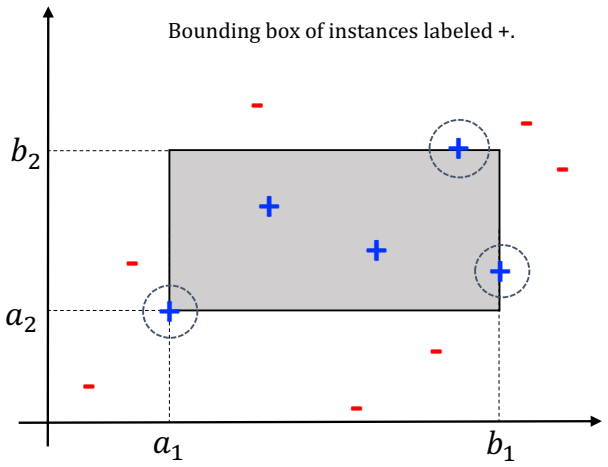

Figure 1: An axis-aligned rectangle in two dimensions.

- **Concept class:** A concept class $\mathcal{C}$ is a pre-determined set of concepts.

2 The Consistency Model

We start our study of *learnability* with the*consistency* model. While this may not be a very realistic model of learning, it's a great place for demonstrating ideas that will come up again later.

We say that a concept $c \in \mathcal{C}$ is *consistent* with a set of samples $\{(x_1, y_1), \ldots, (x_m, y_m)\}$, if for all $i \in [m]$, $c(x_i) = y_i$. We say that a concept class $\mathcal{C}$ is *learnable* in the *consistency model* if there is an algorithm $\mathcal{A}$ such that, for any set of labeled instances $S = \{(x_1, y_1), \ldots, (x_m, y_m)\}$, $\mathcal{A}(S) = c$ for some $c \in \mathcal{C}$ that is consistent with the examples, or $\mathcal{A}(S) =$ "no such concept exists" if no such concept $c \in \mathcal{C}$ exists.

We are especially interested in algorithms that are *computationally efficient* and can learn in the consistency model. Let's consider a few examples of such algorithms.

2.1 Geometrical Examples

Axis-aligned rectangles. In this example, we consider $\mathcal{X} = \mathbb{R}^2$ and $\mathcal{Y} = \{+, -\}$. An axis-aligned rectangle is a concept that assigns $+$ to instance that are within some rectangle and $-$ to those outside. More formally, each concept $c \in \mathcal{C}$ is defined by four parameters $a_1, b_1, a_2, b_2 \in \mathbb{R}$ and

$$c(\mathbf{x}) = \begin{cases} + & \text{if for } i \in \{1, 2\}, a_i \leq x_i \leq b_i \\ - & \text{otherwise} \end{cases}$$

How would you design an algorithms $\mathcal{A}$ that runs efficiently, in the size of the input set S and learns $\mathcal{C}$ in the consistency model? A simple solution is to find the minimum and maximum instances

labeled $+$ along each of the axes. Then, consider the axis-aligned rectangle whose boundaries are defined by these examples (See Figure 1). Note that, this is the most *conservative* concept in $\mathcal{C}$ that is consistent with all $(\mathbf{x}_i, +) \in S$. That is, the positive region of any other axis-aligned rectangle $c' \in \mathcal{C}$ that is also consistent with all $(\mathbf{x}_i, +) \in S$ includes the positive region of c. All that is left is to check if c is also consistent with all $(\mathbf{x}_i, -) \in S$. If it is consistent then $\mathcal{A}(S) = c$. Otherwise, no other concept can be consistent with the data in which case $\mathcal{A}(S)$ states that no consistent axis-aligned rectangle exists.

Note that such an algorithm take $O(|S|)$ to find the minimum and maximum instances labeled $+$ along each axis and to form the bonding box. It takes an additional $O(|S|)$ runtime to check that the concept defined by the bounding box is consistent with the rest of the data.

2.2 Consistency Model and Generalization

At a high level, learning in the consistency model is really about optimization on observed labeled instances. But it is not necessary clear whether the concept that is learned in the consistency model is a good predictor for instances that the algorithm has not encountered yet. As a thought exercise and while ignoring the need for computationally efficient algorithms, consider the setting where $\mathcal{C}$ includes all boolean functions on n bit. Then one can learn (inefficiently though) in the consistency model, by having $\mathcal{A}(S)$ say no consistent concept exists if S includes an $(\mathbf{x}, y)$ and $(\mathbf{x}, \bar{y})$, and otherwise memorize the instances and their respective labels through a disjunctive normal form. While this algorithm does learn in the consistency model, the learning seems ineffective in a way. For example, any instance that hasn't appeared in S will be labeled as negative. This makes the concept class especially brittle on unseen instances.

3 The PAC Model

We now define additional notations for capturing accuracy and generalization.

- **Data-Generating distribution:** We consider a probability distribution $\mathcal{D}$ over $\mathcal{X} \times \mathcal{Y}$. We assume that instances we received are independent and identically distributed (i.i.d) according to an unknown $\mathcal{D}$. I.I.D. means that samples are all distributed according to the same distribution and are independent of each other. Throughout today's lecture, we assume that there is an unknown concept c that determines the true label of instances. That is, there is an unknown $c \in \mathcal{C}$ such that $\mathcal{D}$ is only supported on instances (x, y) where $y = c(x)$. This is called the "consistency assumption".

- **True Error:** Consider a data-generating distribution $\mathcal{D}$ and the true labeling concept c. The *true error* of a concept h with respect to $\mathcal{D}$ is the probability that h makes a mistake, i.e., disagrees with c, on a freshly drawn sample from $\mathcal{D}$. That is,

$$\mathrm{err}_{\mathcal{D}}(h) = \Pr_{(x,y)\sim\mathcal{D}}[h(x) \neq y].$$

- **Empirical Error:** Given a sample set $S = \{(x_1, y_1), \ldots, (x_m, y_m)\}$, the *empirical error* of a concept h with respect to S is the fraction of instances in S that are incorrectly labeled by h. That is,

$$\mathrm{err}_S(h) = \frac{1}{m} \sum_{i=1}^{m} 1(h(x_i) \neq y_i)).$$

The basic idea of the Probably Approximately Correct (PAC) learning model is to assume that labeled instances are coming from a fixed but unknown distribution $\mathcal{D}$ and the goal is to use the sample set S to learn a concept h that has a small *true error* on $\mathcal{D}$.

Definition 3.1 (PAC Learning)**.** An algorithm $\mathcal{A}$ PAC-learns concept class $\mathcal{C}$ if there is a function $m_{\mathcal{C}}(\epsilon, \delta) : (0,1) \times (0,1) \to \mathbb{N}$ such that the following is true: For any $c^* \in \mathcal{C}$ and any distribution $\mathcal{D}$ labeled according to c^*, any $\epsilon > 0$ and $\delta > 0$, there is an algorithm $\mathcal{A}$ that takes an i.i.d. sample set S of size $m \geq m_{\mathcal{C}}(\epsilon, \delta)$, and with probability $1 - \delta$ returns a function $h : \mathcal{X} \to \mathcal{Y}$ such that $\mathrm{err}_{\mathcal{D}}(h) \leq \epsilon$.

Algorithm $\mathcal{A}$ is computationally efficient if the number of samples and runtime is polynomial time in $\frac{1}{\epsilon}, \frac{1}{\delta}, m$ and a natural representation of the concept class $\mathcal{C}$.

In PAC learning, ϵ and δ represent two types of bad events. Here, δ is the probability that a "total disaster" could happen and $\mathcal{A}(S)$ returns a hypothesis h that is completely wrong, i.e., $\mathrm{err}_{\mathcal{D}}(h)$ is very large. So, typical range of δ is 0.01-0.001 or less. On the other hand, ϵ describes how close h and c^* are, when we avoid that total disaster. ϵ is typically much larger than δ, for example $\epsilon = 0.05$-0.1 is a quite reasonable. PAC learning refers to the fact that our hypothesis is "probably" (with probability $1 - \delta$) "approximately" (up to an error of ϵ) correct!

Remark 1 There are different versions of PAC learning based on whether the returned function $h \in \mathcal{C}$ or not. When the algorithm is guaranteed to return a function $h \in \mathcal{C}$ this is called *proper PAC learning*. Otherwise, it's called *improper PAC learning*. To emphasize the fact that instances in $\mathcal{D}$ are generated by a concept c^*, sometimes the community refers to PAC learning as *realizable PAC learning.*

Remark 2 When the assumption that instances in $\mathcal{D}$ are labeled according to a concept c^* that is in the range of outcomes of the algorithm is removed, i.e., *lack of realizibility*, it might be impossible to have a concept h that has $\mathrm{err}_{\mathcal{D}}(h) \leq \epsilon$ at all. This latter model, where no assumption on the existence of a good concept is made, is called *agnostic learning*. We will come back to this in a few lectures.

4 Consistency versus PAC

In this section we show how one can relate learnability in the consistency model and the PAC model.

Theorem 4.1 (PAC Learnability of Finite Concept Classes)**.** *Let $\mathcal{A}$ be an algorithm that learns a concept class $\mathcal{C}$ in the consistency model (that is, it returns $h \in \mathcal{C}$ whenever a consistent concept w.r.t. S exists). Then, $\mathcal{A}$ learns the concept class $\mathcal{C}$ (by the hypothesis class $\mathcal{H} = \mathcal{C}$) in the PAC learning model using*

$$m_{\mathcal{C}}(\epsilon, \delta) = \frac{1}{\epsilon}\left(\ln(|\mathcal{C}|) + \ln(\frac{1}{\delta})\right).$$

Let us review two useful fact before we prove this theorem.

Fact 4.2. *For any $\alpha \in [0, 1]$, $1 - \alpha \leq \exp(-\alpha)$.*

Fact 4.3 (Union Bound)**.** *Let $E_1, \ldots, E_k$ be probabilistic events. Then,* $\Pr\left[\bigcup_{i\in[k]} E_i\right] \leq \sum_{i=1}^{k} \Pr[E_i]$.

Proof of Theorem 4.1. Since we are in the (realizable) PAC setting, we know that there is a concept $c^* \in \mathcal{C}$ that is consistent with the sample set S. Therefore, $\mathcal{A}(S)$ will return a hypothesis $h_S \in \mathcal{C}$ that is also consistent with S. So it is sufficient to show that with probability $1 - \delta$, h_S has true error of at most ϵ. That is, it is sufficient to bound the probability of the following bad event.

$$B : \exists h \in \mathcal{C} \text{ such that } h \text{ is consistent with } S \text{ and } \mathrm{err}_{\mathcal{D}}(h) > \epsilon.$$

Fix one hypothesis $h \in \mathcal{C}$ that has $\mathrm{err}_{\mathcal{D}}(h) > \epsilon$. What is the probability that this hypothesis is consistent with the sample set S? Note that for a freshly sampled $(x, c^*(x)) \sim \mathcal{D}$, the probability that h makes a mistake on x is exactly its true error. That is,

$$\Pr_{(x,c^*(x))\sim\mathcal{D}}[h(x) \neq c^*(x)] = \mathrm{err}_{\mathcal{D}}(h).$$

So, the probability that such an h does not make a mistake on any of the m samples in S is

$$\Pr[h \text{ consistent with } S] = (1 - \mathrm{err}_{\mathcal{D}}(h))^m < (1 - \epsilon)^m \leq \exp(-m\epsilon).$$

Applying this to all possible hypothesis in $\mathcal{C}$, we have that

$$\Pr[B] = \Pr\left[\bigcup_{h\in\mathcal{C}} h \text{ is consistent with } S \text{ and } \mathrm{err}_{\mathcal{D}}(h) > \epsilon\right] \leq |\mathcal{C}| \exp(-m\epsilon).$$

Therefore, when $m \geq \frac{1}{\epsilon}\left(\ln(|\mathcal{C}|) + \ln(\frac{1}{\delta})\right)$, we have that $\Pr[B] \leq \delta$. This proves the claim. □

4.1 Beyond Finite Concept Classes

In the next lecture, we see how you can get a variant of Theorem 4.1 even for some infinite concept classes.