CS6781 - Theoretical Foundations of Machine Learning

Lecture 3: Probably Approximately Correct Learning

January 28, 2020

Lecturer: Nika Haghtalab *Readings: Chp 2.2-3.1, UML*

As we mentioned in the previous lecture, 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. Let us highlight this problem with an example of learning disjunctive normal forms in the consistency model. Let $\mathcal{X} = \{0,1\}^n$ and $\mathcal{C}$ be the class of boolean concepts that can be expressed as disjunctive normal form (DNFs) with n literals and as many clauses as one likes, such as $(x_1 \wedge x_3) \vee (\overline{x_1} \wedge x_5 \wedge x_8)$.

Is the class of DNFs efficiently learnable in the consistency model? It is not hard to see that it is. Consider an algorithm $\mathcal{A}$ that for any set of labeled instances S,

- For every $(\mathbf{x}, +) \in S$, it creates a conjunction where the ith variable appears as x_i or $\overline{x_i}$, based on the value of ith coordinate of $\mathbf{x}$, i.e., $(1,0,1)$ corresponds to the clause $x_1 \wedge \overline{x_2} \wedge x_3$.
- If there is an $\mathbf{x}$ such that $(\mathbf{x}, +) \in S$ and $(\mathbf{x}, -) \in S$, then $\mathcal{A}(S)$ returns "no consistent concept exists." Otherwise, $\mathcal{A}(S)$ returns the disjunction of clauses made in step 1.

This algorithm learns DNFs in the consistency model, because it only assigns a positive label to positive instances in the sample set and negative label otherwise. Now consider an example $(\mathbf{x}, +)$ that never appeared in S. The concept returned by $\mathcal{A}$ labels any such instance as negative, therefore, making a mistake on any positive instance that had never been observed by the algorithm. This means that algorithms in the consistency model are good at memorizing the history, but they may not generalize to unseen examples.

In this lecture, we look at an alternative model of learning that emphasizes the generalization ability of the learned concepts.

1 The PAC Model

We use the definitions of domain $\mathcal{X}$, label set $\mathcal{Y}$, concept classes $\mathcal{C}$, etc. from the previous lecture. We will also define additional notations that will be used in this lecture and many of the following lectures.

- **Data-Generating distribution:** We consider a probability distribution $\mathcal{D}$ over $\mathcal{X}$. 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 $x_1, \ldots, x_m$ 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, the set

of labeled instances $S = \{(x_1, y_1), \ldots, (x_m, y_m)\}$ is generated by taking $x_i \sim \mathcal{D}$ i.i.d and having $y_i = c(x_i)$.

- **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,

$$\mathrm{err}_{\mathcal{D}}(h) = \mathbf{Pr}_{x\sim\mathcal{D}}[h(x) \neq c(x)].$$

- **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 $\mathcal{D}$. We assume that there is an unknown concept $c \in \mathcal{C}$ that truly labels instances in distribution $\mathcal{D}$. We also assume that we have access to another set of concepts $\mathcal{H}$ from which we have to choose the concept. For ease of representation, we often call $\mathcal{H}$ the class of hypotheses.

Definition 1.1 (PAC Learning)**.** An algorithm $\mathcal{A}$ PAC-learns concept class $\mathcal{C}$ by hypothesis class $\mathcal{H}$ if there is a functions $m_{\mathcal{C}}(\epsilon, \delta) : (0, 1) \times (0, 1) \to \mathbb{N}$ 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 hypothesis $h \in \mathcal{H}$ 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 what $\mathcal{H}$ and $\mathcal{C}$ represent. We typically consider $\mathcal{H} \supseteq \mathcal{C}$, to ensure that the target concept c^* remains a legitimate outcome of the algorithm. When $\mathcal{C} = \mathcal{H}$, we call this *proper PAC learning*. If there is a possibility of learning $h \in \mathcal{H} \setminus \mathcal{C}$, this is called *improper PAC learning*. Typically, when you are asked about *PAC learnability* of a concept class $\mathcal{C}$ the choice and representation of $\mathcal{H}$ does not matter and can include any function that can be evaluated in polynomial time. To emphasize the fact that instances in $\mathcal{D}$ are generated by a concept c^* and the representation of $\mathcal{H}$ does not matter, sometimes we refer 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.

2 Consistency versus PAC

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

Theorem 2.1 (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 2.2. *For any $\alpha \in [0, 1]$, $1 - \alpha \leq \exp(-\alpha)$.*

Fact 2.3 (Union Bound)**.** *Let $E_1, \ldots, E_k$ be probabilistic events. Then, $\mathbf{Pr}\left[\bigcup_{i\in[k]} E_i\right] \leq \sum_{i=1}^{k} \mathbf{Pr}\left[E_i\right]$.*

Proof of Theorem 2.1. 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 $x \sim \mathcal{D}$, the probability that h makes a mistake on x is exactly its true error. That is,

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

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

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

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

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

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

3 Examples

Monotone Conjunctions Are monotone conjunctions learnable in the PAC model (by the class of monotone conjunctions)? Yes! Last lecture we saw that we can efficiently learn monotone conjunctions in the consistency model. Note that there are at most 2^n monotone conjunctions. So, using Theorem 2.1, we can PAC learn monotone conjunctions with $m(\epsilon, \delta) = O\left(\epsilon^{-1}(n + \ln(1/\delta)\right)$.

Decision Lists Here, $\mathcal{C}$ is a class of all boolean functions that can be represented as a list of if-then rules of the form

$$\text{If } \ell_1 \text{ then } b_1, \text{ else if } \ell_2 \text{ then } b_2, \ldots, \text{ else } b_k,$$

where each ℓ_i is a literal that is a variable x_j or its negation $\overline{x_j}$ and $b_i \in \{+, -\}$ is the assigned label. For example, a concept $c(\mathbf{x}) =$ "If x_2 then $+$, else if $\overline{x_1}$ then $-$, else $+$" is a decision list.

Is the class of decision lists learnable in the PAC model (by itself)? Let's first see how large $|\mathcal{C}|$ is for the class of all decision lists.

Note that there is a total of $4n$ "if-then" rules, because each variable x_i can appear as one of $\{x_i, \overline{x_i}\}$ and each $b_i \in \{+, 0\}$. Once a rule is applied, there is no point applying it again, since it will not be satisfied a second time. So, the number of decision lists is at most the number of different permutations of these rules. That is $|\mathcal{C}| \leq (4n)!$.[1] This results in $\ln(|\mathcal{C}|) = O(n \ln n)$.

Now, all that remain is to see if we can learn the class of decision lists in the consistency model. Consider the following algorithm that enumerates all $4n$ possible rules in the set L and

1. Find some rule in L that is consistent with the current set of labeled examples, and the "if" clause is satisfied by at least one example. If no such rule exists, then return "No consistent classifier exists."

2. Add that rule to the bottom the decision list and remove any $(x, y) \in S$ from S that satisfied its "if" clause.

3. Repeat until no instances left in S.

This algorithm runs in time polynomial in n and $|S|$, since in any round there are at most $4n$ rules to evaluate on at most $|S|$ samples. Can you see why this algorithm learns in the consistency model? Suppose that the decision list c is consistent with the data. And assume that we are in Step 1 of the above algorithm with at least one instance left in the set S. Each of these instances is classified by some rule in c. Take the top rule. Note that adding this rule does not cause any inconsistencies, because that's the first rule in c to also apply to these examples. So, if it causes inconsistencies in our current decision list, it would have also caused an inconsistency in c. So, the algorithm will always have a suitable rule in Step 1.

Using this in Theorem 2.1, we see that we can efficiently PAC learn decision lists with $m(\epsilon, \delta) = O\left(\epsilon^{-1}(n \ln(n) + \ln(1/\delta)\right)$.

[1] Can you think of a tighter analysis that shows that the number of decision rules is $4^n n!$? To show this, start by showing that no variable appears twice in a decision list.

Linear thresholds Are linear thresholds learnable in the PAC model? Last lecture we saw that we can efficiently learn linear thresholds in the consistency model. But the set of all linear thresholds is infinitely large. Therefore, we cannot directly use Theorem 2.1 to show that linear thresholds are PAC learnable. In the next lecture, we explore how we could reason about PAC learnability of infinite hypothesis classes by understanding their structure.