COMP 585 **Spring 2024**
Probabilistic Toolkit for Learning and Computing *Instructor: Maryam Aliakbarpour*
March 29th, 2024 *Scriber: Erdong Hu*

Lecture 14

1 PAC Learning (Continued)

1.1 Recap

From the previous lecture, we defined *PAC-learnability* as the following:

Definition 1.1. *We say a class C is PAC-learnable if there is an algorithm A such that for all D, ϵ, δ, there is an m as a function of C, ϵ, δ such that with m i.i.d. samples $(x_i, y_i) \sim D$, A has probability $1 - \delta$, A outputs $\hat{c} \in C$ such that*

$$\begin{aligned} err(\hat{c}) &= Pr[\hat{c}(x) \neq y] \\ &\leq \min_{c \in C} err(c) + \epsilon \end{aligned}$$

where $\min_{c \in C} err(c) = 0$ in the realizable case.

1.2 Example: Boolean conjunctions

Let $X = \{0,1\}^n$ be the set of boolean vectors, and let x_i be literal variable on the *ith* boolean, $i \in [n]$ and $\bar{x}_i$ is the negation operation on that literal (Which we also consider as a literal). $\wedge$ denotes the conjunction ("and") operation between literals.

Let concept class C consist of functions $h : X \rightarrow \{0,1\}$ constructed from conjunction and negation operations (Set of conjunction functions).

Example: $n = 3$, $x = (x_1, x_2, x_3)$, $h(x) = x_1 \wedge \bar{x}_2$. Then $h(1,0,1) = 1$, $h(0,0,1) = 0$

Goal: We want to learn unknown function h from samples $(x^i, h(x^i)) \sim D, i \in [m]$

This problem is realizable because h is consistent (Due to being a function).

Algorithm:

1. Begin with $\hat{h}(x) = x_1 \wedge \bar{x}_1 \wedge \ldots x_n \wedge \bar{x}_n$.

2. For each sample x^i, if $h(x^i) = 1$, we remove inconsistent literals from $\hat{h}$. Otherwise, we ignore.

3. Output $\hat{h}$

Example: Given $\hat{h}(x) = x_2 \wedge x_3 \wedge \bar{x}_4$ and sample $\langle(1, 0, 1, 0), 1\rangle$, the algorithm would modify $\hat{h}$ to $\hat{h}(x) = x_3 \wedge \bar{x}_4$ as the x_2 term is inconsistent.

Goal: Given ϵ, δ we want to find sample size m such that

$$Pr[err(\hat{h}) \geq \epsilon] \leq \delta$$

As we only remove inconsistent literals, $\hat{h}$ from the algorithm will never have removed literals from the true realizable h. Hence if $h(x) = 0$ then $\hat{h}(x) = 0$ as well so that $\hat{h}$ always labels true 0 outputs correctly (Does not have false positives). Therefore $\hat{h}$ can only make mistake in the case that $\hat{h} = 0$ but $h(x) = 1$ (False negatives). This means that there is some literal z in $\hat{h}$ that is still inconsistent with h. Denote $h\big|_z(x)$ as the evaluation of the function only on the literal z. Then the aforementioned inconsistency can be denoted as $\hat{h}\big|_z(x) = 0$ but $h\big|_z(x) = 1$.

$$\begin{aligned}
err(\hat{h}) &= Pr_{x\sim D}[h(x) \neq \hat{h}(x)] \\
&= Pr_{x\sim D}[\exists \text{literal } z \in \hat{h} \text{ s.t. } hath\big|_z(x) = 0, \text{ but } h\big|_z(x) = 1] \\
&\leq \sum_{z\in\hat{h}} Pr_{x\sim D}[\hat{h}\big|_z(x) = 0, \text{ but } h\big|_z(x) = 1], \text{ by union bound} \\
&\leq \sum_{z\in\hat{h}} p(z)
\end{aligned}$$

Where we denote $p(z)$ as the probability of literal z being inconsistent. We consider z to be bad if $p(z) \geq \frac{\epsilon}{2n}$. If we have no bad literals, i.e. $p(z) < \frac{\epsilon}{2n}$ for each z

$$\begin{aligned}
err(\hat{h}) &\leq \sum_{z\in\hat{h}} p(z) \\
&< 2n(\frac{\epsilon}{2n}) = \epsilon
\end{aligned}$$

Which gives the desired ϵ-bound. For the δ-bound, in order for $err(\hat{h}) \geq \epsilon$, there must be at least one bad z, $p(z) \geq \frac{\epsilon}{2n}$. For such a bad literal to survive the algorithm, its inconsistency must not have arisen in any of the m samples. Since at each sample we have $p(z)$ probability of encountering the inconsistency of z, it has survival probability $1-p(z)$ per sample. Hence,

$$\begin{aligned}
Pr[err(\hat{h}) \geq \epsilon] &= Pr[\exists \text{bad literal } z] \\
&\leq 2nPr[\text{bad literal } z \text{ survives } m \text{ samples}], \text{ union bound} \\
&\leq 2n(1 - \frac{\epsilon}{2n})^m \\
&\leq 2ne^{-\frac{\epsilon m}{2n}} \leq \delta
\end{aligned}$$

Solving for m gives the following bound

$$\begin{aligned}
m &\geq \frac{2n}{\epsilon} \log{(\frac{2n}{\delta})} \\
m &= O(\frac{n}{\epsilon} \log{(\frac{n}{\delta})})
\end{aligned}$$

So the above algorithm shows this problem is PAC-learnable, i.e. algorithm output $\hat{h}$ has $1 - \delta$ probability of $err(\hat{h}) < \epsilon$.

2 Uniform Convergence and Empirical Risk Minimization (ERM)

2.1 ERM

In both previous examples of PAC-learning, our algorithm's output is made to be consistent with the samples in the training set. This approach is called *Empirical Risk Minimization*

Definition 2.1. *An algorithm is empirical risk minimization (ERM) if and only if for a training set $T \sim D^n$ it outputs a concept c such that $c = \arg\min_{c \in C} err_T(c)$*

2.2 Uniform Convergence

Definition 2.2. *We say for a class C has uniform convergence if and only if for each $\epsilon, \delta > 0$ $\exists m$ (A function of ϵ, δ) that such that for any distribution D*

$$Pr_{T \sim D^m}[\forall c \in C : |err(c) - err_T(\hat{c})| >= \epsilon] \leq \delta$$

where T is the training set

Uniform convergence implies via ERM agnostic PAC-learnability, in other words with probability $1 - \delta$ and optimal choice from ERM c^*

$$
\begin{aligned}
err_T(c^*) &\leq err_T(c) \\
&< err(c) + \epsilon \\
&< \min_{c \in C} err(c) + \epsilon
\end{aligned}
$$

The $\min_{c \in C} err(c)$ is called the *approximation error* that depends only on the hypothesis (concept) class C, while ϵ is the *estimation error*. A richer and more complex C decreases the approximation error but often increases the estimation error.