COMP 585 **Spring 2025**

Probabilistic Toolkit for Learning and Computing *Instructor: Maryam Aliakbarpour*

March 13, 2025 *Scriber: Vincent Wang & Maryam Aliakbarpour*

Lecture 15

The PAC Learning Framework

In machine learning, we often give a machine a set of labeled examples and ask it to learn a function that can predict the labels of new, unseen examples. A key question is: how many examples are needed to learn such a function with high accuracy? The framework of *Probably Approximately Correct (PAC)* learning provides a formal way to answer this question. It connects the number of training samples to the accuracy and confidence of the learned function.

A Motivational Example: The Runner Problem

Suppose you are observing a runner and want to understand the conditions under which they decide to go jogging. Let's assume the weather is described by a pair of features: temperature (x_1) and precipitation (x_2). Based on a set of past observations, our goal is to find a function that can predict whether the runner will jog or stay home. We will make one key assumption: the runner's behavior is perfectly described by a rectangular region of weather conditions. They jog if the weather is inside this rectangle and stay home otherwise.

We have access to a dataset of m past days, where each day is represented by a weather condition and a label ($+1$ for jogging, -1 for staying home). We can view these labels as being generated by a "true" rectangle r^*. Our task is to use the samples to find a hypothesis rectangle $\hat{r}$ that is a good approximation of r^*.

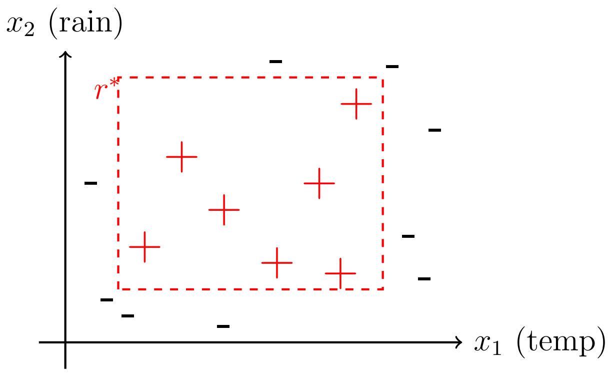

Figure 1: Observed examples from an unknown distribution, labeld by the true concept r^*.

A Learning Algorithm for Our Example

A simple and intuitive algorithm is to find the smallest axis-aligned rectangle that encloses all the positive examples we have seen. We can call this the “tightest-fitting” rectangle.

1. Draw m samples $\{(x^{(i)}, y^{(i)})\}_{i=1}^{m}$ i.i.d. from a distribution D.
2. Identify the set of all positive samples, $S^+ = \{x^{(i)} | y^{(i)} = +1\}$.
3. Output the hypothesis rectangle $\hat{r}$ defined by the minimum and maximum coordinates of the points in S^+.

The figure below shows how our algorithm finds the hypothesis rectangle $\hat{r}$ (the blue rectangle) based on the positive samples (+). The true rectangle r^* is shown in red. The region in between, unshaded red, is where our *learned* hypothesis will make a mistake.

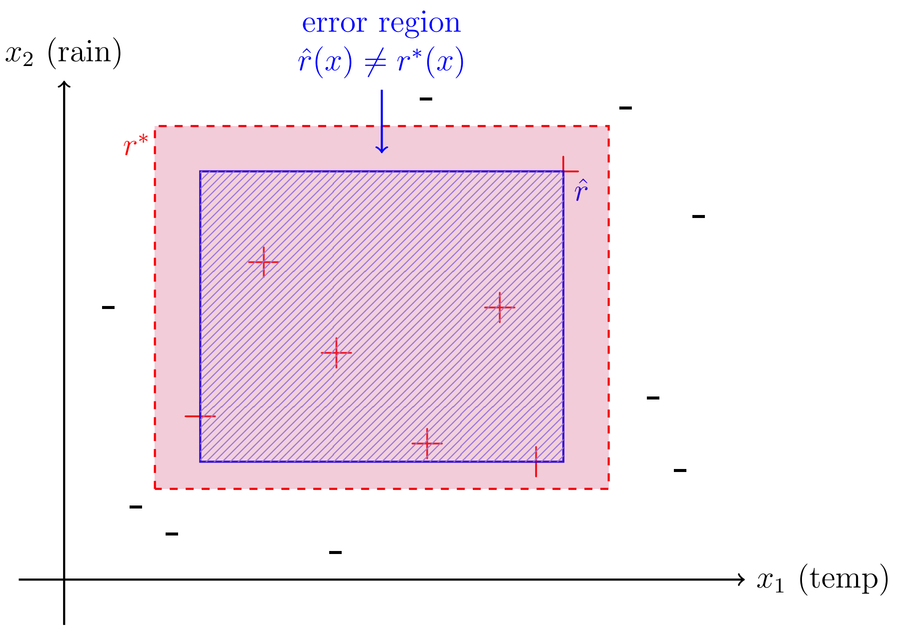

Figure 2: The optimal hypothesis r^*, and the hypothesis found by the learning algorithm $\hat{r}$

Core Definitions

The runner problem provides a clear example for defining the components of the PAC learning framework.

- *Instance Space (X)*: The set of all possible objects or instances that can be labeled. In our example, this is the set of all possible weather conditions, which we model as $\mathbb{R}^2$.

- *Concept (c)*: A function $c : X \to \{-1, +1\}$ that assigns a label to every element in the instance space[cite: 10]. In our example, a concept is a function that takes a weather condition and outputs whether the runner jogs or not.
- *Concept Class ($\mathcal{C}$)*: A collection of concepts, representing the set of all possible functions that the learning algorithm is allowed to choose from. For the runner problem, our concept class is the set of all axis-aligned rectangles in $\mathbb{R}^2$.
- *Target Concept (c^*)*: The true, correct function that we are trying to learn. In our example, it is the specific rectangle r^* that perfectly describes the runner's behavior. Note that target concept may *not* always exist. If such a c^* is in our concept class C, we refer to the setting as the *realizable case.*
- *Data Distribution (D)*: A probability distribution over the instance space from which samples are drawn. In our framework, we assume the distribution is arbitrary but fixed, which is called the *distribution-free* setting. In our example, this corresponds to the likelihood of different weather conditions occurring.
- *Training Set*: A set of m samples $\{(x_i, c^*(x_i))\}_{i=1}^m$ drawn i.i.d. from the data distribution. In our example, this is the collection of labeled weather conditions from past days.

In this context, the terms "hypothesis" and "concept" are sometimes used interchangeably.

Defining PAC Learnability

In PAC learning framework, we aim to design an algorithm that receives the training set and outputs a *good enough* concept $\hat{c}$. We hope that this concept is *approximately correct* with high *probability.*

The quality of our concept $\hat{c}$ is measured by its error, which is the probability of it mislabeling a new, unseen sample drawn from the distribution D. Crucially, the error should be defined over the same distribution from which the examples are drawn:

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

For our example, the error is the probability that a new point lands in the red unshaded region shown in Figure 2. In this reign, the true label (determined by r^*) is $+$, while the

Definition 1. *A concept class is* PAC-learnable *if there exists an efficient algorithm $\mathcal{A}$ that, for any target concept, any distribution, and any desired accuracy ε and confidence δ, receives $m = m(\epsilon, \delta)$ labeled samples from D and produces a hypothesis with an error of at most ε, with probability at least $1 - \delta$.*

In this definition, we assume algorithm $\mathcal{A}$ receives ϵ and δ, uses them to determine a sample size m, and then receives a training set of m examples from a distribution D. While the concept class $\mathcal{C}$ is not an explicit input to the algorithm, its designer has knowledge of it. This approach allows us to ignore the details of how the concept class $\mathcal{C}$ is represented.

Analyzing Our Learning Algorithm

The final step is to prove that our simple algorithm actually works according to the definition of PAC learning. In particular, our goal is to determine how many samples we need with respect to two given parameter ϵ and δ.

Earlier, we defined the error of a concept as its probability of mislabeling. The error of our hypothesis $\hat{r}$ is the probability mass of the region where it disagrees with r^*, which is $r^* \setminus \hat{r}$.

$$\mathrm{err}(\hat{r}) = \mathbf{Pr}_{x \sim D}[x \in r^* \text{ and } x \notin \hat{r}] = D(r^* \setminus \hat{r})$$

Our algorithm could fail if our randomly drawn samples are unrepresentative of the underlying distribution. Specifically, the algorithm fails if, for a given ε, it finds a hypothesis $\hat{r}$ with $\mathrm{err}(\hat{r}) > \varepsilon$. This happens only if none of our samples landed in the region with probability mass greater than ε.

Let's bound the probability of this "bad event."

1. Assume the true error of our chosen $\hat{r}$ is greater than ε. That is, $D(r^* \setminus \hat{r}) > \varepsilon$.
2. For a single random draw $x^{(i)}$ from D, the probability that it falls into this error region is $D(r^* \setminus \hat{r}) > \varepsilon$.
3. Therefore, the probability that a single sample $x^{(i)}$ *misses* this error region is $1 - D(r^* \setminus \hat{r}) < 1 - \varepsilon$.
4. Since our m samples are drawn i.i.d., the probability that *all* m of them miss the error region is:

$$\mathbf{Pr}[\text{no sample lands in } r^* \setminus \hat{r}] = (1 - D(r^* \setminus \hat{r}))^m < (1 - \varepsilon)^m$$

5. This is the probability of failure. We want this probability to be at most δ. Thus, we require:

$$(1 - \varepsilon)^m \leq \delta \, .$$

To find the required number of samples m, we solve the inequality. Using the useful bound $1 - x \leq e^{-x}$ for $x \in \mathbb{R}$, we get:

$$e^{-\varepsilon m} \geq (1 - \varepsilon)^m \, .$$

It is sufficient to require $e^{-\varepsilon m} \leq \delta$. Solving for m:

$$-\varepsilon m \leq \ln(\delta)$$

$$\varepsilon m \geq -\ln(\delta) = \ln(1/\delta)$$

$$m \geq \frac{1}{\varepsilon} \ln\left(\frac{1}{\delta}\right)$$

This result gives us the *sample complexity* for learning axis-aligned rectangles. Since m is a function that grows polynomially with $1/\varepsilon$ and $1/\delta$, and our algorithm is computationally efficient, we can conclude that the class of axis-aligned rectangles is efficiently PAC-learnable.

Bibliographic Note

PAC learning was introduced by Valiant in 1984 [Val84]. This work established a rigorous mathematical foundation for the field of machine learning. The content of this lecture was derived from Section 2-3 in [SSBD14].

References

[SSBD14] Shai Shalev-Shwartz and Shai Ben-David. *Understanding Machine Learning: From Theory to Algorithms*. Cambridge university press, 2014.

[Val84] Leslie G Valiant. A theory of the learnable. *Communications of the ACM*, 27(11):1134–1142, 1984.