CSC 588: Machine learning theory **Spring 2021**

Lecture 2: PAC learning

Lecturer: Chicheng Zhang *Scribe: Chicheng Zhang*

1 The supervised learning pipeline

In the first part of this course, we will study statistical learning, where we will use techniques from probability and statistics to establish guarantees of machine learning algorithms. Specifically, we focus on the so-called supervised learning pipeline, which we illustrate below using a simple example.

Suppose we would like to build an image classifier, such that it can automatically tell cats from dogs (so that in the future, when you take new pictures of pets, it can automatically categorize them for your convenience). To this end, we download many images of cats and dogs from the internet, and associate every image with a label, either 'cat' or 'dog'. We then apply a *learning algorithm* on these labeled images as input, and output a prediction rule (or *classifier*). This classifier can be used for *prediction* or *classification*: upon seeing a new image, it can output a label (either 'cat' or 'dog'), that hopefully successfully tells the species of the pet in the image.

The general training-test pipeline of supervised learning can be summarized by the following figure:

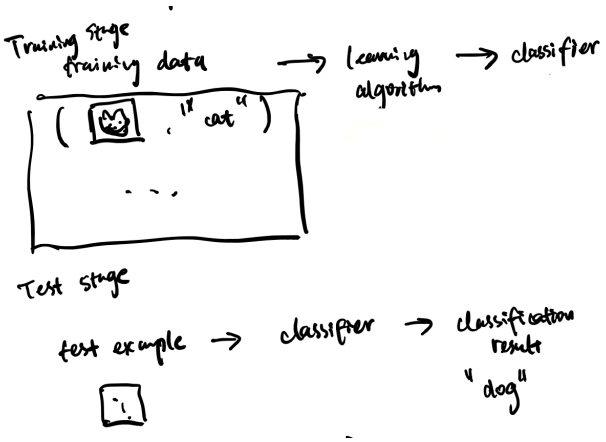

Figure 1: The training-test pipeline of supervised learning

How do we measure the success the the above pipeline? Specifically, what do we mean by having generated a "good" classifier, and what do we mean by having a "good" learning algorithm? To answer these questions, we now take another view of the test stage, as shown in Figure 2.

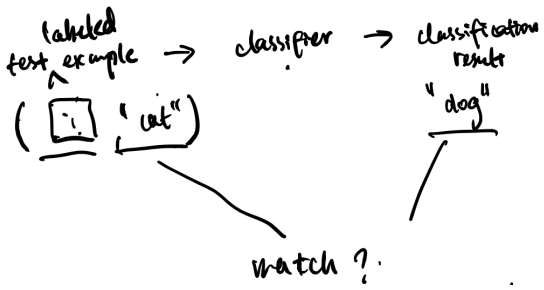

Figure 2: An alternative view of the test stage

Specifically, we can image a test example associated with a label hidden to us. Hypothetically, if we have access to this label, we can measure the performance of a classifier on this example by checking if the classifier's prediction on this example matches the label; loosely speaking, it performs well if the prediction matches the label. Moreover, we assume that there is a distribution over the (unlabeled example, label) pairs where test examples are drawn; a natural notion of the performance of a classifier can be its *error rate*, that is, when a random example is drawn, the chance that a mismatch (misclassification, prediction error) happens.

Intuitively, why can a learning algorithm generate a good classification model? We need to assume the training and test data satisfy certain similarity condition; specifically, here, we assume that training and test data are drawn from the same distribution.

Given this pipeline, we can ask the following questions:

1. How do we formalize the process mathematically?
2. Can we formally define what properties a good predictor / a good learning algorithm should have?

The PAC learning model in the next section provides a solid answer to these questions.

2 The PAC learning model

PAC stands for "Probably Approximately Correct" [1], which is a celebrated theoretical model for supervised machine learning (in particular, classification).

Let us start with defining some basic terminologies used in the PAC learning model:

1. Instance domain $\mathcal{X}$ - the space where the images lie in. For example, each image is grayscale, of resolution 640×480, and is represented by its pixel intensity. Then we can take $\mathcal{X} = \mathbb{R}^{640\times480}$.
2. Label space $\mathcal{Y} = \{-1, +1\}$ - the space where labels lie in. For example, we can use $+1$ to denote class 'cat', and -1 to denote class 'dog'.
3. Data distribution D: the distribution where examples (x, y)'s are drawn from.
4. Training set: a set S drawn iid from D.
5. Classifier h (hypothesis): a mapping from $\mathcal{X}$ to $\mathcal{Y}$ - given a image as input, the classifier outputs a label in $\{-1, +1\}$, indicating whether the image is a cat or a dog.
6. Hypothesis class $\mathcal{H}$: a (structured) collection of classifiers. For example,

$$\mathcal{H} = \{\text{all neural networks with ResNet-18 architecture}\},^{1}$$

where each different combination of weightings corresponds to a different classifier in $\mathcal{H}$.

[1] With an additional linear threshold layer for the output.

7. Error: given a classifier h, define its generalization error as $\text{err}(h, D) = \mathbb{P}_{(x,y)\sim D}(h(x) \neq y)$. This is the performance measure of a classifier. Define h's training error with respect to the training set S of size n as $\text{err}(h, S) = \frac{1}{n}\sum_{(x,y)\in S} \mathbf{1}(h(x) \neq y)$, also abbreviated as $\mathbb{P}_{(x,y)\sim S}(h(x) \neq y)$.

8. Realizable vs. agnostic settings:

   (a) Realizable: we are given the promise that there exists a classifier $h^\star$ in $\mathcal{H}$, such that $\text{err}(h^\star, D) = 0$.

   (b) Agnostic: there may or may not exists a classifier in $\mathcal{H}$ that has zero generalization error.

Definition 1. *We call an algorithm $\mathcal{A}$ PAC learns hypothesis class $\mathcal{H}$ with sample complexity function $f : (0,1) \times (0,1) \to \mathbb{N}$, if for any distribution D realizable with respect to $\mathcal{H}$, $\epsilon > 0$, $\delta > 0$, when $\mathcal{A}$ receives $m \geq f(\epsilon, \delta)$ iid training examples from D as input, it outputs a classifier $\hat{h}$, such that with probability $1 - \delta$,*

$$\text{err}(\hat{h}, D) \leq \epsilon.$$

Remarks:

1. ϵ is called the *target error rate*, and δ is called the *confidence parameter*. This may remind you the notion of $(1-\alpha)$-confidence interval in statistics, where the goal is to construct an interval such that it contains the underlying parameter with probability $1-\alpha$. Here α is essentially δ, the failure probability. "Probably" means that the algorithm succeed with high probability $(1 - \delta)$, and "Approximately Correct" means that the classifier returned has small error rate (ϵ).

2. The original definition in [1] requires that the learning algorithm runs in polynomial time. Here we relax the original definition and do not require computational efficiency.

3. The sample complexity is hypothesis-class dependent but *distribution independent*. In practice, one might want a sample complexity definition that "exploits the easiness of the data", i.e. if distribution D is easy to learn, then the learning algorithm may draw less samples from D to learn a good classifier. We will return to this point when we study model selection.

4. If $\mathcal{A}$ always return a h in $\mathcal{H}$, then it is called a *proper* learning algorithm. Otherwise, it is called an improper learning algorithm. Interestingly, improper learning can sometimes provide more power in achieving learnability and saving sample complexity.

2.1 Sample complexity of finite hypothesis classes

Let us consider the setting where the hypothesis class $\mathcal{H}$ is of finite size. In fact, it can be argued that all hypothesis classes considered in computer-based applications are finite! To see this, consider for example the set of all ResNet-18 networks, with each weight taking values in 64-bit floating-point numbers. Any such network can be represented using $(64 \times \#\text{weights})$ bits.

We will now show a foundational result: any finite hypothesis class $\mathcal{H}$ is learnable, and has a PAC sample complexity of order $f(\epsilon, \delta) = \frac{1}{\epsilon}(\ln|\mathcal{H}| + \ln\frac{1}{\delta})$. To see this, let us consider the following simple learning algorithm.

Algorithm 1 The consistency algorithm

Require: Training samples S of size m iid from D, hypothesis class $\mathcal{H}$

1: **return** $\hat{h}$ in $\mathcal{H}$ such that $\hat{h}$ agrees with all examples in S, that is, for all (x, y) in S, $\hat{h}(x) = y$.

This algorithm makes intuitively sense because it tries to return a classifier that can "explain" the training examples well; moreover, it will always have a valid output, because under the realizability assumption, with probability 1, $h^\star$ agrees with all training examples.

Theorem 2. *Suppose $\mathcal{H}$ is finite. If the consistency algorithm (Algorithm 1) is given m iid examples from D realizable with respect to $\mathcal{H}$, then with probability $1-\delta$, its output $\hat{h}$ is such that*

$$\mathrm{err}(\hat{h}, D) \leq \epsilon_R(m, \delta, \ln|\mathcal{H}|) \triangleq \frac{\ln|\mathcal{H}| + \ln\frac{1}{\delta}}{m}. \tag{1}$$

In other words, it PAC learns hypothesis class $\mathcal{H}$ with sample complexity $m(\epsilon, \delta) = \frac{1}{\epsilon}(\ln|\mathcal{H}| + \ln\frac{1}{\delta})$.

We will make a few remarks on the theorem, and prove this theorem in the next lecture.

The high-level idea of the proof is as follows. Denote by $\mu = \frac{\ln|\mathcal{H}|+\ln\frac{1}{\delta}}{m}$. We decompose the hypothesis class to two parts: $\mathcal{H}_\mu = \{h \in \mathcal{H} : \mathrm{err}(h, D) \leq \mu\}$ and $\mathcal{H}_\mu^C = \mathcal{H} \backslash \mathcal{H}_\mu = \{h \in \mathcal{H} : \mathrm{err}(h, D) > \mu\}$. See Figure 3.

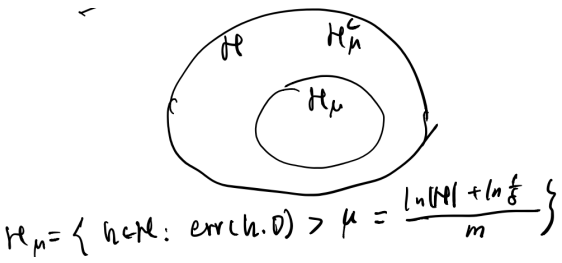

Figure 3: A decomposition of hypothesis class $\mathcal{H}$

We will show that, there exists an event E, such that E happens with probability $1-\delta$, in which all classifiers in $\mathcal{H}_\mu$ will make mistakes on some example in S, that is, they won't be selected by the consistency algorithm. Therefore, the consistency algorithm will only return elements from $\mathcal{H}_\mu^C$, which has a generalization error rate at most μ.

References

[1] L. Valiant, A theory of the learnable. J. ACM, 1984.