Chapter 4

Learning Theory

When we have built a classifier, one question people always ask is how good the classifier is. They want to evaluate the classifier. They want to see whether the classifier is able to predict what it is supposed to predict. Often times, the "gold standard" is to report the classification accuracy: Give me a testing dataset, and I will tell you how many times the classifier has done correctly. This is one way of evaluating the classifier. However, does this evaluation method really tells us how good the classifier is? Not clear. All it says is that for this classifier trained on a particular training dataset and tested on a particular testing dataset, the classifier performs such and such. Will it perform well if we train the classifier using another training set, maybe containing more data points? Will it perform well if we test it on a different testing dataset? It seems that we lack a way to quantify the generalization ability of our classifier.

There is another difficulty. When we train the classifier, we can only access the training data but not the testing data. This is like taking an exam. We can never see the exam questions when we study, for otherwise we will defeat the purpose of the exam! Since we only have the training set when we design our classifier, how do we tell whether we have trained a good classifier? Should we choose a more complex model? How many samples do we need? Remember, we cannot use any testing data and so all the evaluation has to be done internally using the training data. How to do that? Again, we are missing a way to quantify the performance of the classifier.

The objective of this chapter is to answer a few theoretical (and also practical) questions in learning: (i) Is learning feasible? (ii) How much can a classifier generalize? (iii) What is the relationship between number of training samples and the complexity of the classifier? (iv) How do we tell whether we have obtained a good classifier during the training?

4.1 Introduction

Let us start by recalling (refreshing) our notations. The vector $\boldsymbol{x} \in \mathbb{R}^d$ is called the input vector. There is an *unknown* target function $f : \mathcal{X} \to \mathcal{Y}$ which maps $\boldsymbol{x}$ to a label $y = f(\boldsymbol{x})$.

The set $\mathcal{X}$ contains all the input vectors, and we call $\mathcal{X}$ the input space. The set $\mathcal{Y}$ contains the corresponding labels, and we call $\mathcal{Y}$ the output space.

In any supervised learning scenario, there is always a training set $\mathcal{D}$. The training set contains N input-output pairs $(\boldsymbol{x}_1, y_1), \ldots, (\boldsymbol{x}_N, y_N)$, where $\boldsymbol{x}_n$ and y_n are related via $y_n = f(\boldsymbol{x}_n)$, for $n = 1, \ldots, N$. These input-output pairs are called the data points or samples. Since $\mathcal{D}$ is a finite collection of data points, there are many $\boldsymbol{x} \in \mathcal{X}$ that do not live in the training set $\mathcal{D}$. A data point $\boldsymbol{x}_n$ that is inside $\mathcal{D}$ is called an **in-sample**, and a data point $\boldsymbol{x}$ that is outside $\mathcal{D}$ is called an **out-sample**.

When we say that we use a machine learning algorithm to learn a classifier, we mean that we have an algorithmic procedure $\mathcal{A}$ which uses the training set $\mathcal{D}$ to select a hypothesis function $g : \mathcal{X} \to \mathcal{Y}$. The hypothesis function is again a mapping from $\mathcal{X}$ to $\mathcal{Y}$, because it tells what a sample $\boldsymbol{x}$ is being classified. However, a hypothesis function g is not the same as the target function f. We never know f because f is simply *unknown*. No matter how much we learn, the hypothesis function g is at best an approximation of f. The approximation error can be zero in some hand-craved toy examples, but in general $g \neq f$. All hypothesis functions are contained in the hypothesis set $\mathcal{H}$. If the hypothesis set is finite, then $\mathcal{H} = \{h_1, \ldots, h_M\}$, and g will be one of these h_m's. A hypothesis set can be infinite, for example we can perturb a perceptron decision boundary by an infinitesimal step to an infinite hypothesis set. An infinite hypothesis set is denoted by $\mathcal{H} = \{h_\sigma\}$, where σ denotes a continuous parameter.

The drawings in Figure **??** illustrate a few key concepts we just mentioned. On the left hand side there is an input space $\mathcal{X}$, which contains a small subset $\mathcal{D}$. The subset $\mathcal{D}$ is the training set, which includes a finite number of training samples or in-samples. There is an unknown target function f. The target function f maps an $\boldsymbol{x}_n$ to produce an output $y_n = f(\boldsymbol{x}_n)$, hence giving a colored dots in the middle of the figure. The objective of learning is to learn a classifier which can classify the red from the blue. The space containing all the possible hypothesis is the hypothesis set $\mathcal{H}$, which contains $h_1, \ldots, h_M$. The final hypothesis function returned by the learning algorithm is g.

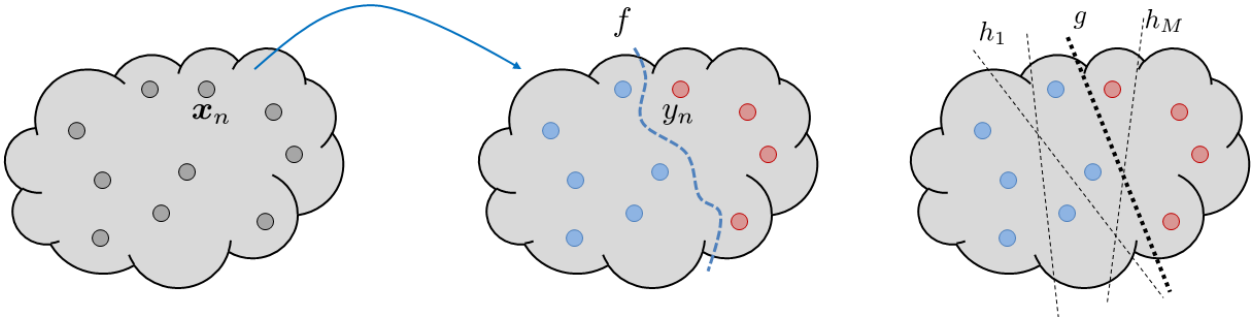

Figure 4.1: [Left] Treat the cloud as the entire input space $\mathcal{X}$ and correspondingly the output space $\mathcal{Y}$. The dots are the **in-samples** $\boldsymbol{x}_1, \ldots, \boldsymbol{x}_N$. The target function is a mapping f which takes $\boldsymbol{x}_n$ and send it to y_n. The red and blue colors indicate the class label. [Right] A learning algorithm picks a hypothesis function g from the hypothesis set $\mathcal{H} = \{h_1, \ldots, h_M\}$. Note that some hypotheses are good, and some are bad. A good learning algorithm will pick a good hypothesis, and a bad learning algorithm can pick a bad hypothesis.

Figure **??** illustrates what we called a **probabilistic** learning model. It is called a probabilistic learning model because there is an unknown distribution $p(\boldsymbol{x})$. The training samples $\{\boldsymbol{x}_1, \ldots, \boldsymbol{x}_N\}$ are generated according to $p(\boldsymbol{x})$. The same $p(\boldsymbol{x})$ also generates the testing samples $\boldsymbol{x}$. It is possible to lift the probabilistic assumption so that the training samples are drawn deterministically. In this case, the samples are simply fixed set of data points $\{\boldsymbol{x}_1, \ldots, \boldsymbol{x}_N\}$. The deterministic assumption will make learning infeasible, as we will see shortly. Therefore, we shall mainly focus on the probabilistic assumption.

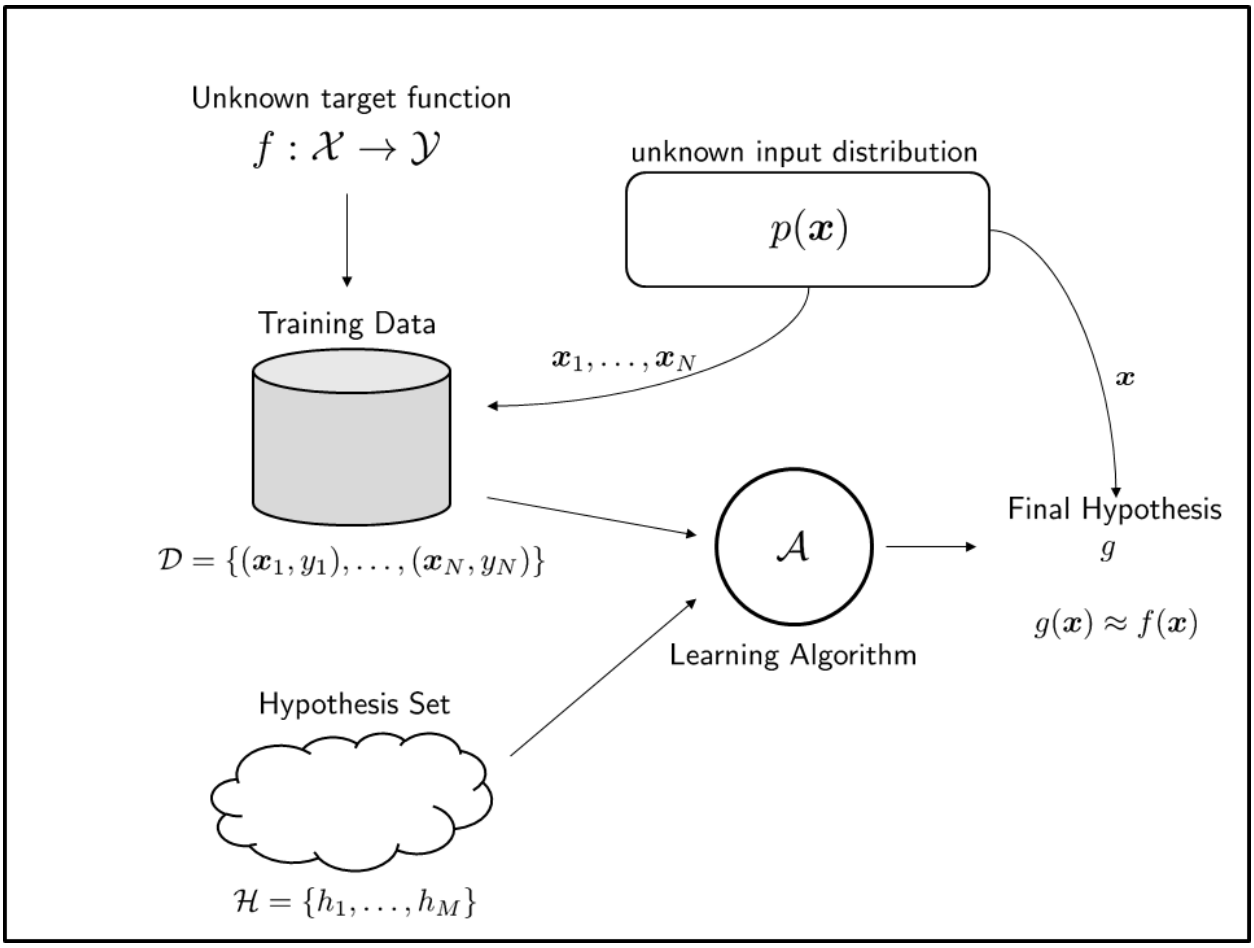

Figure 4.2: All essential components of a machine learning model.

4.2 Is Learning Feasible?

The first question we ask is: Suppose we have a training set $\mathcal{D}$, can we learn the target function f? If the answer is YES, then we are all in business, because it means that we will be able to predict the data we have not seen. If the answer is NO, then machine learning is a lair and we should all go home, because it means that we can only memorize what we have seen but we will not be able to predict what we have not seen.

Interestingly, the answer to this question depends on how we define the training samples $\boldsymbol{x}_n$'s. If $\boldsymbol{x}_n$'s are deterministically defined, then the answer is NO because $\mathcal{D}$ can contain no information about the out-samples. Thus, there is no way to learn outside $\mathcal{D}$. If $\boldsymbol{x}_n$'s are drawn from a probabilistic distribution, then the answer is YES because the distribution will tell us something about the out-samples. Let us look at these two situations one by one.

Learning outside $\mathcal{D}$

Let us look at the deterministic case. Consider a 3-dimensional input space $\mathcal{X} = \{0, 1\}^3$. Each vector $\boldsymbol{x} \in \mathcal{X}$ is a binary vector containing three elements, e.g., $\boldsymbol{x} = [0,\ 0,\ 1]^T$ or $\boldsymbol{x} = [1,\ 0,\ 1]^T$. Since there are 3 elements and each element take a binary state, there are totally $2^3 = 8$ input vectors in $\mathcal{X}$.

How about the number of possible target functions f can we have? Remember, a target function f is a mapping which converts an input vector $\boldsymbol{x}$ to a label y. For simplicity let us assume that f maps $\boldsymbol{x}$ to a binary output $y \in \{+1, -1\}$. Since there are 8 input vectors, we can think of f as a 8-bit vector, e.g., $f = [-1,\ +1,\ -1,\ -1,\ -1,\ +1,\ +1,\ +1]$, where each entry represents the output. If we do the calculation, we can show that there are totally $2^8 = 256$ possible target functions.

Here is the learning problem. Can we learn f from $\mathcal{D}$? To ensure that f is *unknown*, we will not disclose what f is. Instead, we assume that there is a training set $\mathcal{D}$ containing 6 training samples $\{\boldsymbol{x}_1, \ldots, \boldsymbol{x}_6\}$. Corresponding to each $\boldsymbol{x}_n$ is the label y_n. The relationship between $\boldsymbol{x}_n$ and y_n is shown in the table below. So our task is to pick a target function from the 256 possible choices.

$\boldsymbol{x}_n$			y_n
0	0	0	○
0	0	1	●
0	1	0	●
0	1	1	○
1	0	0	●
1	0	1	○

$\boldsymbol{x}_n$			y_n	g	f_1	f_2	f_3	f_4
0	0	0	○	○	○	○	○	○
0	0	1	●	●	●	●	●	●
0	1	0	●	●	●	●	●	●
0	1	1	○	○	○	○	○	○
1	0	0	●	●	●	●	●	●
1	0	1	○	○	○	○	○	○
1	1	0		○/●	○	●	○	●
1	1	1		○/●	○	○	●	●

Since we have seen 6 out of 8 input vectors in $\mathcal{D}$, there remains two input vectors we have not seen and need to predict. Thus, we can quickly reduce the number of possible target functions to $2^2 = 4$. Let us call these target functions f_1, f_2, f_3 and f_4. The boolean structure of these target functions are shown on the right hand side of the table above. Note that the first 6 entries of each f_i is fixed because they are already observed in $\mathcal{D}$.

In the table above we write down the final hypothesis function g. The last two entries of g is to be determined by the learning algorithm. If the learning algorithm decides "○", then we will have both "○". If the learning algorithm decides a "○" followed by a "●", then

we will have a "$\circ$" followed by a "$\bullet$". So the final hypothesis function g can be one of the 4 possible choices, same number of choices of the target functions.

Since we assert that f is *unknown*, by only observing the first 6 entries we will have 4 equally good hypothesis functions. They are equally good, because no matter which hypothesis function we choose, the last 2 entries will agree or disagree with the target depending on which one is the true target function. For example, on the left hand side of the table below, the true target function is f_1 and so our g is correct. But if the true target function is f_3, e.g., the right hand side of the table, then our g is wrong. We can repeat the experiment by choosing another g, and we can prove that not matter which g we choose, we only have 25% chance of picking the correct one. This is the same as drawing a lottery from 4 numbers. The information we learned from the training set $\mathcal{D}$ does not allow us to infer anything outside $\mathcal{D}$.

$\boldsymbol{x}_n$			y_n	g	f_1	f_2	f_3	f_4
0	0	0	$\circ$	$\circ$	$\circ$	$\circ$	$\circ$	$\circ$
0	0	1	$\bullet$	$\bullet$	$\bullet$	$\bullet$	$\bullet$	$\bullet$
0	1	0	$\bullet$	$\bullet$	$\bullet$	$\bullet$	$\bullet$	$\bullet$
0	1	1	$\circ$	$\circ$	$\circ$	$\circ$	$\circ$	$\circ$
1	0	0	$\bullet$	$\bullet$	$\bullet$	$\bullet$	$\bullet$	$\bullet$
1	0	1	$\circ$	$\circ$	$\circ$	$\circ$	$\circ$	$\circ$
1	1	0		$\circ$	$\circ$	$\bullet$	$\circ$	$\bullet$
1	1	1		$\circ$	$\circ$	$\circ$	$\bullet$	$\bullet$

$\boldsymbol{x}_n$			y_n	g	f_1	f_2	f_3	f_4
0	0	0	$\circ$	$\circ$	$\circ$	$\circ$	$\circ$	$\circ$
0	0	1	$\bullet$	$\bullet$	$\bullet$	$\bullet$	$\bullet$	$\bullet$
0	1	0	$\bullet$	$\bullet$	$\bullet$	$\bullet$	$\bullet$	$\bullet$
0	1	1	$\circ$	$\circ$	$\circ$	$\circ$	$\circ$	$\circ$
1	0	0	$\bullet$	$\bullet$	$\bullet$	$\bullet$	$\bullet$	$\bullet$
1	0	1	$\circ$	$\circ$	$\circ$	$\circ$	$\circ$	$\circ$
1	1	0		$\circ$	$\circ$	$\bullet$	$\circ$	$\bullet$
1	1	1		$\circ$	$\circ$	$\circ$	$\bullet$	$\bullet$

The above analysis shows that learning is infeasible if we have a deterministic generator generating the training samples. The argument holds regardless which learning algorithm $\mathcal{A}$ we use, and what hypothesis set $\mathcal{H}$ we choose. Whether $\mathcal{H}$ contains the correct hypothesis function, and whether $\mathcal{A}$ can pick the correct hypothesis, there is no difference in terms of predicting outside $\mathcal{D}$. We can also extend the analysis from binary function to general learning problem. As long as f remains unknown, it is impossible to predict outside $\mathcal{D}$.

Probabilistic Analysis

The deterministic analysis gives us a pessimistic result. Now, let us look at a probabilistic analysis. On top of the training set $\mathcal{D}$, we pose an assumption. We assume that all $\boldsymbol{x} \in \mathcal{X}$ is drawn from a distribution $p_{\boldsymbol{X}}(\boldsymbol{x})$. This includes all the in-samples $\boldsymbol{x}_n \in \mathcal{D}$ and the out-samples $\boldsymbol{x} \in \mathcal{X}$. At a first glance, putting a distributional assumption $p_{\boldsymbol{X}}(\boldsymbol{x})$ does not seem any different from the deterministic case: We still have a training set $\mathcal{D} = \{\boldsymbol{x}_1, \ldots, \boldsymbol{x}_N\}$, and f is still unknown. How can we learn the unknown f using just the training samples?

Suppose that we pick a hypothesis function h from the hypothesis set $\mathcal{H}$. For every in-sample $\boldsymbol{x}_n$, we check whether the output returned by h is the same as the output returned by f, i.e., $\{h(\boldsymbol{x}_n) = f(\boldsymbol{x}_n)\}$, for $n = 1, \ldots, N$. If $\{h(\boldsymbol{x}_n) = f(\boldsymbol{x}_n)\}$, then we say that the in-sample $\boldsymbol{x}_n$ is correctly classified in the training. If $\{h(\boldsymbol{x}_n) \neq f(\boldsymbol{x}_n)\}$, then we say that $\boldsymbol{x}_n$

is incorrectly classified. Averaging over all the N samples, we obtain a quantity called the **in-sample error**, or the training error.

Definition 1 (In-sample Error)**.** *Consider a training set $\mathcal{D} = \{\boldsymbol{x}_1, \ldots, \boldsymbol{x}_N\}$, and a target function f. The **in-sample error** (or the training error) of a hypothesis function $h \in \mathcal{H}$ is the empirical average of $\{h(\boldsymbol{x}_n) \neq f(\boldsymbol{x}_n)\}$:*

$$E_{\text{in}}(h) \stackrel{def}{=} \frac{1}{N}\sum_{n=1}^{N} [\![h(\boldsymbol{x}_n) \neq f(\boldsymbol{x}_n)]\!], \tag{4.1}$$

where $[\![\cdot]\!] = 1$ if the statement inside the bracket is true, and $= 0$ if the statement is false.

Training error is the amount of error we have during the training process. A good learning algorithm $\mathcal{A}$ should pick a hypothesis h that gives low training error. Training error is sometimes called the cost function (or the loss function) when we post the learning problem as an optimization. Thus, picking a good hypothesis is equivalent to minimizing the training error.

How about the out-samples? Since we assume that $\boldsymbol{x}$ is drawn from a distribution $p_{\boldsymbol{X}}(\boldsymbol{x})$, we can define the out-sample error as the probability that $\{h(\boldsymbol{x}) \neq f(\boldsymbol{x})\}$, for all $\boldsymbol{x} \sim p_{\boldsymbol{X}}(\boldsymbol{x})$.

Definition 2 (Out-sample Error)**.** *Consider an input space $\mathcal{X}$ containing elements $\boldsymbol{x}$ drawn from a distribution $p_{\boldsymbol{X}}(\boldsymbol{x})$, and a target function f. The **out-sample error** (or the testing error) of a hypothesis function $h \in \mathcal{H}$ is*

$$E_{\text{out}}(h) \stackrel{def}{=} \mathbb{P}[h(\boldsymbol{x}) \neq f(\boldsymbol{x})], \tag{4.2}$$

where $\mathbb{P}[\cdot]$ measures the probability of the statement based on the distribution $p_{\boldsymbol{X}}(\boldsymbol{x})$.

Since $[\![\cdot]\!]$ is a binary function, the out-sample error is the expected value of a sample being misclassified over the entire distribution:

$$\begin{aligned} E_{\text{out}}(h) &= \mathbb{P}[h(\boldsymbol{x}) \neq f(\boldsymbol{x})] \\ &= \underbrace{[\![h(\boldsymbol{x}_n) \neq f(\boldsymbol{x}_n)]\!]}_{=1}\mathbb{P}\Big\{h(\boldsymbol{x}_n) \neq f(\boldsymbol{x}_n)\Big\} \\ &\quad + \underbrace{[\![h(\boldsymbol{x}_n) = f(\boldsymbol{x}_n)]\!]}_{=0}\Big(1 - \mathbb{P}\Big\{h(\boldsymbol{x}_n) \neq f(\boldsymbol{x}_n)\Big\}\Big) \\ &= \mathbb{E}\Big\{[\![h(\boldsymbol{x}_n) \neq f(\boldsymbol{x}_n)]\!]\Big\}. \end{aligned} \tag{4.3}$$

Therefore, the relationship between the in-sample error $E_{\text{in}}(h)$ and out-sample error $E_{\text{out}}(h)$ is equivalent to the relationship between the empirical average and the population mean of a random variable. Figure **??** shows how an in-sample error is computed.

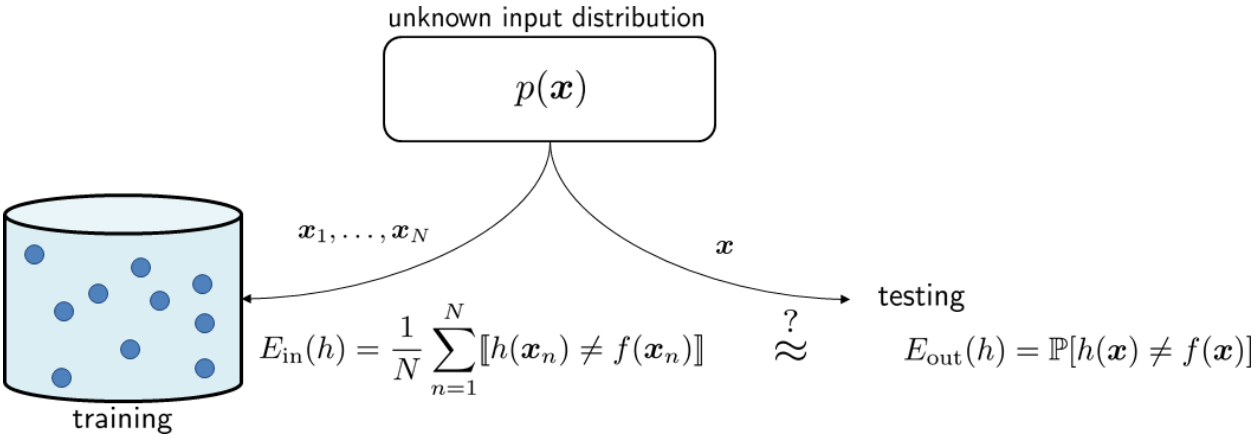

Figure 4.3: E_{in} is evaluated using the training data, whereas E_{out} is evaluated using the testing sample.

We now need a mathematical tool to analyze $E_{\text{in}}(h)$ and $E_{\text{out}}(h)$. The tool we use is reminiscent to the Weak Law of Large Number. However, instead of using the simple Chebyshev inequality, we use a much more powerful inequality called the Hoeffding inequality.

Theorem 1 (Hoeffding Inequality)**.** *Let $X_1, \ldots, X_N$ be a sequence of i.i.d. random variables such that $0 \leq X_n \leq 1$ and $\mathbb{E}[X_n] = \mu$. Then, for any $\epsilon > 0$,*

$$\mathbb{P}\left[\left|\frac{1}{N}\sum_{n=1}^{N} X_n - \mu\right| > \epsilon\right] \leq 2e^{-2\epsilon^2 N}. \tag{4.4}$$

The proof of the Hoeffding inequality can be found in many graduate probability textbooks. We refer the readers to `http://cs229.stanford.edu/extra-notes/hoeffding.pdf`. The essential implication of Hoeffding inequality is that the empirical average $\frac{1}{N}\sum_{n=1}^{N} X_n$ converges exponentially to the population mean μ as $N \to \infty$.

Let us take a quick comparison between the Hoeffding inequality and the Chebyshev inequality. Chebyshev inequality states that

$$\mathbb{P}\left[\left|\frac{1}{N}\sum_{n=1}^{N} X_n - \mu\right| > \epsilon\right] \leq \frac{\sigma^2}{\epsilon^2 N}. \tag{4.5}$$

If we let $2e^{-2\epsilon^2 N} \leq \delta$ for some δ in Hoeffding inequality, and $\frac{\sigma^2}{\epsilon^2 N}$ for some δ in Chebyshev inequality, we can easily see that the two inequalities imply

$$N \geq -\frac{1}{2\epsilon^2}\log\frac{\delta}{2}, \quad \text{and} \quad N \geq \frac{\sigma^2}{\epsilon^2\delta}.$$

For simplicity let us assume that $\sigma = 1$, $\epsilon = 0.1$ and $\delta = 0.01$. Then the above calculation will give $N \geq 265$ for Hoeffding whereas $N \geq 10000$ for Chebyshev. That means, Hoeffding

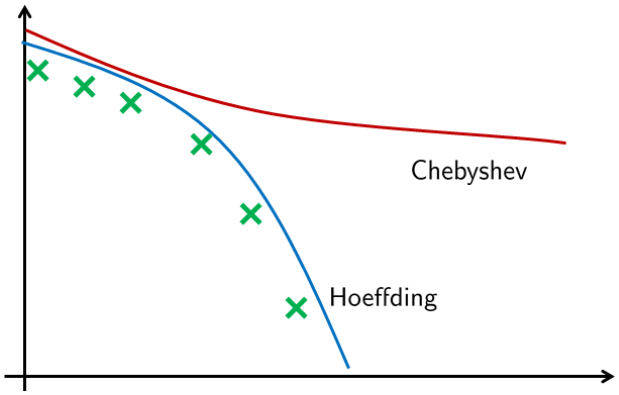

Figure 4.4: Comparing Hoeffding inequality and Chebyshev inequality to predict the actual probability bound.

inequality has a much lower prediction of how many samples we need to achieve an error of $\delta \leq 0.01$.

The Hoeffding inequality implies the following result in learning. Substituting the in-sample error $E_{\text{in}}(h)$ and out-sample error $E_{\text{out}}(h)$ into the inequality, we can show that

$$\mathbb{P}\left[|E_{\text{in}}(h) - E_{\text{out}}(h)| > \epsilon\right] \leq 2e^{-2\epsilon^2 N}. \tag{4.6}$$

As the number of training samples N grows, the in-sample error $E_{\text{in}}(h)$ (which is the **training error**) converges to the out-sample error $E_{\text{out}}(h)$ (which is the **testing error**). The in-sample error $E_{\text{in}}(h)$ is something we can compute numerically using the training set. The out-sample error is an unknown quantity because we do not know the target function f. Hoeffding inequality says even though we do not know $E_{\text{out}}(h)$, for large enough N the in-sample error $E_{\text{in}}(h)$ will be sufficiently close to $E_{\text{out}}(h)$. Therefore, we will be able to tell how good the hypothesis function is without accessing the unknown target function.

PAC Framework

The probabilistic analysis is called a **probably approximately correct** (PAC) framework. The word P-A-C comes from three principles of the Hoeffding inequality:

- **Probably**: We use the probability $\mathbb{P}\big[\,|E_{\text{in}}(h) - E_{\text{out}}(h)| > \epsilon\big] \leq 2e^{-2\epsilon^2 N}$ as a measure to quantify the error.
- **Approximately**: The in-sample error is an approximation of the out-sample error, as given by $\mathbb{P}\left[|E_{\text{in}}(h) - E_{\text{out}}(h)| > \epsilon\right] \leq 2e^{-2\epsilon^2 N}$. The approximation error is controlled by ϵ.
- **Correct**: The error is bounded by the right hand side of the Hoeffding inequality: $\mathbb{P}\left[|E_{\text{in}}(h) - E_{\text{out}}(h)| > \epsilon\right] \leq 2e^{-2\epsilon^2 N}$. The accuracy is controlled by N for a fixed ϵ.

Now, there is one last problem we need to resolve. The above Hoeffding inequality holds for a *fixed* hypothesis function h. This means that h is already chosen *before* we generate the dataset. If we allow h to change *after* we have generated the dataset, then the Hoeffding

inequality is no longer valid. What do we mean by after generating the dataset? In any learning scenario, we are given a training dataset $\mathcal{D}$. Based on this dataset, we have to choose a hypothesis function g from the hypothesis set $\mathcal{H}$. The hypothesis g we choose depends on what samples are inside $\mathcal{D}$ and which learning algorithm $\mathcal{A}$ we use. So g changes after the dataset is generated.

Why is Hoeffding inequality invalid if we use g instead of h? Suppose that $\mathcal{H}$ contains M hypothesis functions $h_1, \ldots, h_M$. The final hypothesis g is one of these potential hypotheses. To have $|E_{\text{in}}(g) - E_{\text{out}}(g)| > \epsilon$, we need to ensure that at least one of the M potential hypotheses can satisfy the inequality. This implies that

$$\begin{aligned} |E_{\text{in}}(g) - E_{\text{out}}(g)| > \epsilon \qquad \Longrightarrow \qquad & |E_{\text{in}}(h_1) - E_{\text{out}}(h_1)| > \epsilon \\ \textbf{or} \quad & |E_{\text{in}}(h_2) - E_{\text{out}}(h_2)| > \epsilon \\ & \cdots \\ \textbf{or} \quad & |E_{\text{in}}(h_M) - E_{\text{out}}(h_M)| > \epsilon. \end{aligned}$$

As a result, we can show that

$$\begin{aligned} \mathbb{P}\Big\{ |E_{\text{in}}(g) - E_{\text{out}}(g)| > \epsilon \Big\} &\overset{(a)}{\leq} \mathbb{P}\Big\{ \quad |E_{\text{in}}(h_1) - E_{\text{out}}(h_1)| > \epsilon \\ &\qquad \textbf{or} \quad |E_{\text{in}}(h_2) - E_{\text{out}}(h_2)| > \epsilon \\ &\qquad \cdots \\ &\qquad \textbf{or} \quad |E_{\text{in}}(h_M) - E_{\text{out}}(h_M)| > \epsilon \quad \Big\} \\ &\overset{(b)}{\leq} \sum_{m=1}^{M} \mathbb{P}\Big\{ \quad |E_{\text{in}}(h_m) - E_{\text{out}}(h_m)| > \epsilon \quad \Big\}, \end{aligned}$$

where (a) holds because $\mathbb{P}[A] \leq \mathbb{P}[B]$ if $A \Rightarrow B$, and (b) is the Union bound which says $\mathbb{P}[A \textbf{ or } B] \leq \mathbb{P}[A] + \mathbb{P}[B]$. Therefore, if we bound each h_m using the Hoeffding inequality

$$\mathbb{P}\Big\{ |E_{\text{in}}(h_m) - E_{\text{out}}(h_m)| > \epsilon \Big\} \leq 2e^{-2\epsilon^2 N},$$

then the overall bound on g is the sum of the M terms.

Theorem 2. *Consider a learning problem where we have a dataset $\mathcal{D} = \{\boldsymbol{x}_1, \ldots, \boldsymbol{x}_N\}$, and a hypothesis set $\mathcal{H} = \{h_1, \ldots, h_M\}$. Suppose g is the final hypothesis picked by the learning algorithm. Then, for any $\epsilon > 0$,*

$$\mathbb{P}\Big\{ |E_{\text{in}}(g) - E_{\text{out}}(g)| > \epsilon \Big\} \leq 2Me^{-2\epsilon^2 N}. \tag{4.7}$$

Feasibility from the Two View Points

The deterministic analysis shows that learning is infeasible, whereas the probabilistic analysis shows that learning is feasible. Are they contradictory? If we look at them closely, we realize that there is in fact no contradiction. Here are the reasons.

1. **Guarantee** and **Possibility**. If we want a deterministic answer, then the question we ask is "Can $\mathcal{D}$ tell us something *certain* about f outside $\mathcal{D}$?" In this case the answer is no because if we have not seen the example, there is always uncertainty about the true f. If we want a probabilistic answer, then the question we ask is "Can $\mathcal{D}$ tell us something *possibly* about f outside $\mathcal{D}$?" In this case the answer is yes.

2. Role of the **distribution**. There is one common distribution $p_{\boldsymbol{X}}(\boldsymbol{x})$ which generates both the in-samples and the out-samples. Thus, whatever $p_{\boldsymbol{X}}$ we use to generate $\mathcal{D}$, we must use it to generate the testing samples. The testing samples are not inside $\mathcal{D}$, but they come from the same distribution. Also, all samples are generated *independently*, so that we have i.i.d. when using the Hoeffding inequality.

3. **Learning goal**. The ultimate goal of learning is to make $E_{\text{out}}(g) \approx 0$. However, in order establish this result, we need two levels of approximation:

$$E_{\text{out}}(g) \quad \underset{\text{Hoeffding Inequality}}{\approx} \quad E_{\text{in}}(g) \quad \underset{\text{Training Error}}{\approx} \quad 0 \tag{4.8}$$

   The first approximation is made by the Hoeffding inequality, which ensures that for sufficiently large N, we can approximate the out-sample error by the examples in $\mathcal{D}$. The second approximation is to make the in-sample error, i.e., the training error, small. This requires a good hypothesis and a good learning algorithm.

The result in Equation (**??**) tells us something about the complexity of the hypothesis set $\mathcal{H}$ and the target function f.

- **More complex $\mathcal{H}$?** If $\mathcal{H}$ is complex with a large M, then the approximation by the Hoeffding inequality becomes loose. Remember, Hoeffing inequality states that

$$\mathbb{P}\Big\{ |E_{\text{in}}(g) - E_{\text{out}}(g)| > \epsilon \Big\} \leq 2Me^{2\epsilon^2 N}.$$

  As M grows, the upper bound on the right hand side becomes loose, and so we will run into risk where $E_{\text{in}}(g)$ can deviate from $E_{\text{out}}(g)$. However, if M is large, we have more candidate hypotheses to choose from and so the second approximation about the training error will go down. This gives the following relationship.

$$E_{\text{out}}(g) \quad \underset{\text{worse if } \mathcal{H} \text{ complex}}{\approx} \quad E_{\text{in}}(g) \quad \underset{\text{good if } \mathcal{H} \text{ complex}}{\approx} \quad 0$$

  Where is the optimal trade-off? This requires more investigation.

- **More complex f?** If the target function f is complex, we will suffer from being not able to push the training error down. This makes $E_{\text{in}}(g) \approx 0$ difficult. However, since the complexity of f has no influence to the Hoeffding inequality, the first approximation $E_{\text{in}}(g) \approx E_{\text{out}}(g)$ is unaffected. This gives us

$$E_{\text{out}}(g) \quad \underset{\text{no effect by } f}{\approx} \quad E_{\text{in}}(g) \quad \underset{\text{worse if } f \text{ complex}}{\approx} \quad 0$$

  Trying to improve the approximation $E_{\text{in}}(g) \approx 0$ by increasing the complexity of $\mathcal{H}$ needs to pay a price. If $\mathcal{H}$ becomes complex, then the approximation $E_{\text{in}}(g) \approx E_{\text{out}}(g)$ will be hurt.

4.3 VC Analysis

The objective of this section is go further into the analysis of the Hoeffding inequality to derive something called the **generalization bound**. There are two parts of our discussion. The first part is easy, which is to rewrite the Hoeffding inequality into a form of "confidence interval" or "error bar". This will allow us interpret the result better.

The second part is to replace the constant M in the Hoeffding inequality by something smaller. This will allow us derive something more meaningful. Why do we want to do that? What could go wrong with M? Remember that M is the number of hypotheses in $\mathcal{H}$. If $\mathcal{H}$ is a finite set, then everything is fine because the exponential decaying function of the Hoeffding inequality will override the constant M. However, for any practical $\mathcal{H}$, M is infinite. Think of a perceptron algorithm. If we slightly perturb the decision boundary by an infinitesimal translation, we will get an infinite number of hypotheses, although these hypotheses could be very similar to each other. If M is infinite, then the probability bound offered by the Hoeffding inequality can potentially be bigger than 1 which is valid but meaningless. To address this issue we need to learn a concept called the **VC dimension**.

Generalization Bound

Let us start with the Hoeffding inequality:

$$\mathbb{P}\Big\{ |E_{\text{in}}(g) - E_{\text{out}}(g)| > \epsilon \Big\} \leq 2Me^{-2\epsilon^2 N}.$$

Notice that this inequality is written in terms of ϵ. We want to rewrite the inequality. The inequality can be viewed as $\mathbb{P}[\mathcal{B}] \leq \delta$ for some event $\mathcal{B}$ (the $\mathcal{B}$ad event), which is equivalent to say that with probability $1-\delta$, the event $\mathcal{B}$ does not happen. Putting into our equation, we have that for probability $1-\delta$,

$$E_{\text{in}}(g) - \epsilon \leq E_{\text{out}}(g) \leq E_{\text{in}}(g) + \epsilon.$$

If we can express ϵ in terms of δ, then we will arrive our goal of rewriting the Hoeffding inequality. How about we substitute $\delta = 2Me^{-2\epsilon^2 N}$, which is the upper bound on the right hand side. By rearrange the terms, we can show that $\epsilon = \sqrt{\frac{1}{2N}\log\frac{2M}{\delta}}$. Therefore, we arrive at the following inequality.

Theorem 3. *Consider a learning problem where we have a dataset $\mathcal{D} = \{\boldsymbol{x}_1, \ldots, \boldsymbol{x}_N\}$, and a hypothesis set $\mathcal{H} = \{h_1, \ldots, h_M\}$. Suppose g is the final hypothesis picked by the learning algorithm. Then, with probability at least $1-\delta$,*

$$E_{\text{in}}(g) - \sqrt{\frac{1}{2N}\log\frac{2M}{\delta}} \leq E_{\text{out}}(g) \leq E_{\text{in}}(g) + \sqrt{\frac{1}{2N}\log\frac{2M}{\delta}}. \tag{4.9}$$

The inequality given by Equation (**??**) is called the **generalization bound**, which we can consider it as an "error bar". There are two sides of the generalization bound:

- $E_{\text{out}}(g) \leq E_{\text{in}}(g) + \epsilon$ (Upper Bound). The upper bound gives us a safe-guard of how worse $E_{\text{out}}(g)$ can be compared to $E_{\text{in}}(g)$. It says that the unknown quantity $E_{\text{out}}(g)$ will not be significantly higher than $E_{\text{in}}(g)$. The amount is specified by ϵ.
- $E_{\text{out}}(g) \geq E_{\text{in}}(g) + \epsilon$ (Lower Bound). The lower bound tells us what to expect. It says that the unknown quantity $E_{\text{out}}(g)$ cannot be better than $E_{\text{in}}(g) - \epsilon$.

To make sense of the generalization bound, we need to ensure that $\epsilon \to 0$ as $N \to \infty$. In doing so, we need to assume that M does not grow exponentially fast, for otherwise term $\log 2M$ will cancel out the effect of $1/N$. However, if $\mathcal{H}$ is an infinite set, then M is unavoidably infinite.

The Growth Function

To resolve the issue of having an infinite M, we realize that there is a serious slack caused by the union bound when deriving the Hoeffding inequality. If we look at the union bound, we notice that for every hypothesis $h \in \mathcal{H}$ there is an event $\mathcal{B} = \{|E_{\text{in}}(h) - E_{\text{out}}(h)| > \epsilon\}$. If we have M of these hypotheses, the union bound tells us that

$$\mathbb{P}[\mathcal{B}_1 \textbf{ or } \ldots \textbf{ or } \mathcal{B}_M] \leq \mathbb{P}[\mathcal{B}_1] + \ldots + \mathbb{P}[\mathcal{B}_M].$$

The union bound is tight ("$\leq$" is replaced by "$=$") when all the events $\mathcal{B}_1, \ldots, \mathcal{B}_M$ are not overlapping. But if the events $\mathcal{B}_1, \ldots, \mathcal{B}_M$ are overlapping, then the union bound is loose, in fact, very loose. Having a loose bound does not mean that the bound is wrong. The bound is still correct, but the right hand side of the inequality will be a severe overestimate of the left hand side. Will this happen in practice? Unfortunately many hypotheses are indeed very similar to each other and so the events $\mathcal{B}_1, \ldots, \mathcal{B}_M$ are overlapping. For example, if we move

the decision boundary returned by a perceptron algorithm by an infinitesimal step then we will have infinitely many hypotheses, and everyone is highly dependent on each other.

We need some tools to handle the overlapping situation. To do so we introduce two concepts. The first concept is called the **dichotomy**, and the second concept is called the **growth function**. Dichotomies will define a growth function, and the growth function will allow us replace M by a much smaller quantity that takes care of the overlapping issue.

Consider a dataset containing N data points $\boldsymbol{x}_1, \ldots, \boldsymbol{x}_N$. Pick a hypothesis h from the hypothesis set $\mathcal{H}$, and for simplicity assume that the hypothesis is binary: $\{+1, -1\}$. If we apply h to $(\boldsymbol{x}_1, \ldots, \boldsymbol{x}_N)$, we will get a N-tuple $(h(\boldsymbol{x}_1), \ldots, h(\boldsymbol{x}_N))$ of ± 1's. Each N-tuple is called a **dichotomy**. The collection of all possible N-tuples (by picking all $h \in \mathcal{H}$) is defined as $\mathcal{H}(\boldsymbol{x}_1, \ldots, \boldsymbol{x}_N)$. For example, if $\mathcal{H}$ contains two hypotheses h_α and h_β such that h_α turns all training samples $\boldsymbol{x}_n$ to $+1$ and h_β turns all training samples $\boldsymbol{x}_n$ to -1, then we have two dichotomies and $\mathcal{H}(\boldsymbol{x}_1, \ldots, \boldsymbol{x}_N)$ is defined as

$$\begin{aligned}\mathcal{H}(\boldsymbol{x}_1, \ldots, \boldsymbol{x}_N) &= \Big\{ \big(h_\alpha(\boldsymbol{x}_1), \ldots, h_\alpha(\boldsymbol{x}_N)\big), \big(h_\beta(\boldsymbol{x}_1), \ldots, h_\beta(\boldsymbol{x}_N)\big) \Big\} \\ &= \Big\{ \big(+1, \ldots, +1\big), \big(-1, \ldots, -1\big) \Big\}.\end{aligned}$$

More generally, the definition of $\mathcal{H}(\boldsymbol{x}_1, \ldots, \boldsymbol{x}_N)$ is as follows.

Definition 3. *Let $\boldsymbol{x}_1, \ldots, \boldsymbol{x}_N \in \mathcal{X}$. The dichotomies generated by $\mathcal{H}$ on these points are*

$$\mathcal{H}(\boldsymbol{x}_1, \ldots, \boldsymbol{x}_N) = \{(h(\boldsymbol{x}_1), \ldots, h(\boldsymbol{x}_N)) \mid h \in \mathcal{H}\}. \tag{4.10}$$

The above definition suggests that $\mathcal{H}(\boldsymbol{x}_1, \ldots, \boldsymbol{x}_N)$ is a function depending on the training samples $\boldsymbol{x}_1, \ldots, \boldsymbol{x}_N$. Therefore, a different set of $\{\boldsymbol{x}_1, \ldots, \boldsymbol{x}_N\}$ will give a different $\mathcal{H}(\boldsymbol{x}_1, \ldots, \boldsymbol{x}_N)$. However, since $\mathcal{H}(\boldsymbol{x}_1, \ldots, \boldsymbol{x}_N)$ is a binary N-tuple, there will be identical sequences of ± 1's in $\mathcal{H}(\boldsymbol{x}_1, \ldots, \boldsymbol{x}_N)$. Let us look at one example.

Suppose there are $N = 3$ data points in $\mathcal{X}$ so that we have $\boldsymbol{x}_1, \boldsymbol{x}_2, \boldsymbol{x}_3$. Use any method to build a linear classifier (could be a linear regression of a perceptron algorithm). Since there are infinitely many lines we can draw in the 2D plane, the hypothesis set $\mathcal{H}$ contains infinitely many hypotheses. Now, let us assume that the training data $\boldsymbol{x}_1, \boldsymbol{x}_2, \boldsymbol{x}_3$ are located at position A, B, C respectively, as illustrated in Figure **??**. These locations are fixed, and the 3 data points $\boldsymbol{x}_1, \boldsymbol{x}_2, \boldsymbol{x}_3$ must stay at these three locations. For this particular configuration of the locations, we can make as many as $2^3 = 8$ dichotomies. Notice that one dichotomy can still have infinitely many hypotheses. For example in the top left case of Figure **??**, we can move the yellow decision boundary up and low slightly, and we will still get the same dichotomy of $[-1, -1, -1]$. However, as we move the decision boundary away by changing the slope and intercept, we will eventually land on a different dichotomy, e.g., $[-1, +1, -1]$ as shown in the bottom left of Figure **??**. As we move around the decision boundary, we can construct at most 8 dichotomies for $\boldsymbol{x}_1, \boldsymbol{x}_2, \boldsymbol{x}_3$ located at A, B and C.

What if we move $\boldsymbol{x}_1, \boldsymbol{x}_2, \boldsymbol{x}_3$ to somewhere else, for example the locations specified by the red part of Figure **??**? In this case some dichotomies are not allowed, e.g., the cases of $[+1, -1, +1]$ and $[-1, +1, -1]$ are not allowed because our hypothesis set contains only linear models and a linear model is not able to cut through 3 data points of alternating classes with a straight line. We can still get the remaining six configurations, but the total will be less than 8. The total number of dichotomies here is 6.

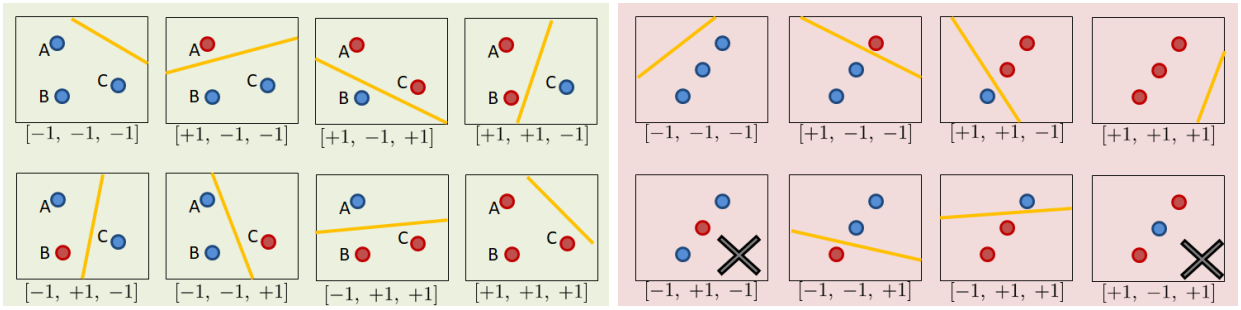

Figure 4.5: For a fixed configuration of $\boldsymbol{x}_1, \boldsymbol{x}_2, \boldsymbol{x}_3$, we can obtain different numbers of dichotomies. Suppose the hypothesis set contains linear models. [Left] There are 8 dichotomies for three data points located not on a line. [Right] When the three data points are located on a line, the number of dichotomies becomes 6.

Now we want to define a quantity that measures the number of dichotomies. This quantity should be universal for any configuration of $\boldsymbol{x}_1, \ldots, \boldsymbol{x}_N$, and should only be a function of $\mathcal{H}$ and N. If we can obtain such quantity, then we will have a way to make a better estimate than M. To eliminate the dependency on $\boldsymbol{x}_1, \ldots, \boldsymbol{x}_N$, we realize that among all the possible configurations of $\boldsymbol{x}_1, \ldots, \boldsymbol{x}_N$, there exists one that can maximize the size of $\mathcal{H}(\boldsymbol{x}_1, \ldots, \boldsymbol{x}_N)$. Define this maximum as the **growth function**.

Definition 4. *The growth function for a hypothesis set $\mathcal{H}$ is*

$$m_{\mathcal{H}}(N) = \max_{\boldsymbol{x}_1, \ldots, \boldsymbol{x}_N \in \mathcal{X}} |\mathcal{H}(\boldsymbol{x}_1, \ldots, \boldsymbol{x}_N)|, \tag{4.11}$$

where $|\cdot|$ denotes the cardinality of a set.

For example, $m_{\mathcal{H}}(3)$ of a linear model is 8, because if we configure $\boldsymbol{x}_1, \boldsymbol{x}_2, \boldsymbol{x}_3$ like the ones in the green part of Figure **??**, we will get 8 dichotomies. Of course, if we land on the red case we will get 6 dichotomies only, but the definition of $m_{\mathcal{H}}(3)$ asks for the maximum which is 8. How about $m_{\mathcal{H}}(N)$ when $N = 4$? It turns out that there are at most 14 dichotomies no matter where we put the four data points $\boldsymbol{x}_1, \boldsymbol{x}_2, \boldsymbol{x}_3, \boldsymbol{x}_4$.

So what is the difference between $m_{\mathcal{H}}(N)$ and M? Both are measures of the number of hypotheses. However, $m_{\mathcal{H}}(N)$ is measured from the N training samples in $\mathcal{X}$ whereas M is the number of hypotheses we have in $\mathcal{H}$. The latter could be infinite, the former is upper bounded (at most) 2^N. Why 2^N? Suppose we have N data points and the hypothesis is

binary. Then the set of all dichotomies $\mathcal{H}(\boldsymbol{x}_1, \ldots, \boldsymbol{x}_N)$ must be a subset in $\{+1, -1\}^N$, and hence there are at most 2^N dichotomies:

$$m_{\mathcal{H}}(N) \leq 2^N.$$

If a hypothesis set $\mathcal{H}$ is able to generate all 2^N dichotomies, then we say that $\mathcal{H}$ **shatter** $\boldsymbol{x}_1, \ldots, \boldsymbol{x}_N$. For example, a 2D perceptron algorithm is able to shatter 3 data points because $m_{\mathcal{H}}(3) = 2^3$. However, the same 2D perceptron algorithm is not able to shatter 4 data points because $m_{\mathcal{H}}(4) = 14 < 2^4$.

VC Dimension

We are now at the last step of our analysis. Let us start by looking at what we can do with the growth function. The most straight forward step is to replace M by $m_{\mathcal{H}}(N)$:

$$E_{\text{in}}(g) - \sqrt{\frac{1}{2N}\log\frac{2m_{\mathcal{H}}(N)}{\delta}} \leq E_{\text{out}}(g) \leq E_{\text{in}}(g) + \sqrt{\frac{1}{2N}\log\frac{2m_{\mathcal{H}}(N)}{\delta}}.$$

Since we know that $m_{\mathcal{H}}(N) \leq 2^N$, a natural attempt is to upper bound $m_{\mathcal{H}}(N)$ by 2^N. However, this will not help us because

$$\sqrt{\frac{1}{2N}\log\frac{2m_{\mathcal{H}}(N)}{\delta}} \leq \sqrt{\frac{1}{2N}\log\frac{2(2^N)}{\delta}} = \sqrt{\frac{1}{2N}\log\frac{2^{N+1}}{\delta}}.$$

For large N we can approximate $2^{N+1} \approx 2^N$, and so

$$\frac{1}{2N}\log\frac{2^N}{\delta} \approx \frac{N\log 2 - \log\delta}{2N} = \frac{\log 2}{2} - \frac{\log\delta}{2N} \to (\log 2)/2.$$

Therefore, as $N \to \infty$, the error bar will never approach zero but to a constant. This makes the generalization fail.

Can we find a better upper bound on $m_{\mathcal{H}}(N)$ so that we can send the error bar to zero as N grows? Here we introduce a parameter allows us to characterize the growth function.

Definition 5 (VC Dimension)**.** *The Vapnik-Chervonenkis dimension of a hypothesis set $\mathcal{H}$, denoted by d_{VC}, is the largest value of N for which $m_{\mathcal{H}}(N) = 2^N$.*

For example, consider the 2D perceptron algorithm. We can start with $N = 3$, and gradual increase N until we hit a critical point.

Suppose $N = 3$. Recall that $m_{\mathcal{H}}(3)$ is the maximum number of dichotomies that can be generated by a hypothesis set under $N = 3$ data points. As we have shown earlier, as long as the 3 data points are not on a straight line, it is possible to draw 8 different dichotomies. If the 3 data points are on a straight line, we can only generate 6 dichotomies. However,

since $m_{\mathcal{H}}(3)$ picks the maximum, we have that $m_{\mathcal{H}}(3) = 2^3$. Therefore, a 2D percetpron can shatter 3 data points.

Suppose $N = 4$. As we have discussed earlier, if we have $N = 4$ data points, there are always 2 dichotomies that cannot be generated by the perceptron algorithm. This implies that the growth function is $m_{\mathcal{H}}(4) = 14 < 2^4$. Since the perceptron algorithm can shatter $N = 3$ data points but not $N = 4$ data points, the VC dimension is $d_{\text{VC}} = 3$.

In genreal, if we have a high-dimensional perceptron algorithm, we can show this:

Theorem 4 (VC Dimension of Perceptron)**.** *Consider the input space $\mathcal{X} = \mathbb{R}^d \cup \{1\}$ ($\boldsymbol{x} = [x_1, \ldots, x_d, 1]^T$). The VC dimension of the perceptron algorithm is*

$$d_{\text{VC}} = d + 1. \tag{4.12}$$

Proof. We shall prove that $d_{\text{VC}} \geq d + 1$ and $d_{\text{VC}} \leq d + 1$. To prove $d_{\text{VC}} \geq d + 1$, we ask: Can we shatter $d + 1$ data points by a d-dimensional perceptron algorithm? Note that here we are only trying to show that it is possible to shatter $d + 1$ data points. Begin "possible" means that we can guaranteed to be able to shatter up to $d + 1$ data points. Since d_{VC} is the next number that the algorithm cannot shatter, by proving this result we can claim that d_{VC} is at least $d + 1$. Whether it can go to $d + 2$ is to be determined.

Since our goal is to show $d_{\text{VC}} \geq d+1$, we just need to pick a configuration (i.e., $d+1$ data points) such that the perceptron algorithm can shatter. To this end we choose $\boldsymbol{x}_1, \ldots, \boldsymbol{x}_{d+1}$ by defining $\boldsymbol{x}_n = [1, 0, \ldots, 1, \ldots, 0]^T$, i.e., 1 on the first entry and a standard basis vector on the rest. Geometrically, if we ignore the bias term (i.e., the first entry of each vector $\boldsymbol{x}_n$), then the data points live on the vertices of a d-dimensional cube. We want to show that this configuration can be shattered by the perceptron algorithm.

Recall that a perceptron algorithm makes a decision by checking

$$\text{sign}(\boldsymbol{x}_n^T \boldsymbol{w}) \stackrel{?}{=} y_n,$$

where $y_n \in \{+1, -1\}$ is a binary decision. Our question can be formulated as: Is it possible to find a vector $\boldsymbol{w}$ such that

$$\begin{cases} \text{sign}(\boldsymbol{x}_1^T \boldsymbol{w}) & = y_1 \\ & \vdots \\ \text{sign}(\boldsymbol{x}_{d+1}^T \boldsymbol{w}) & = y_{d+1} \end{cases}$$

If we can find $\boldsymbol{w}$, then that means we can shatter the $d + 1$ data points.

The first thing we realize is that the sign operator does not matter as far as finding *a* $\boldsymbol{w}$. If we solve the above system of equations without the sign, and if we obtain a $\boldsymbol{w}$ that flips the sign of one of the rows, then the $\boldsymbol{w}$ we found is not able to shatter. But if the $\boldsymbol{w}$ we found can still fit all the $d + 1$ equations, then we can safely remove the sign and prove that

the $d+1$ data points are shattered. Therefore, to this end, we consider a simpler problem.

$$\begin{bmatrix} -\boldsymbol{x}_1^T- \\ -\boldsymbol{x}_2^T- \\ \vdots \\ -\boldsymbol{x}_{d+1}^T- \end{bmatrix} \begin{bmatrix} w_0 \\ w_1 \\ \vdots \\ w_d \end{bmatrix} = \begin{bmatrix} 1 & 0 & 0 & \dots & 0 \\ 1 & 1 & 0 & \dots & 0 \\ 1 & 0 & 1 & & 0 \\ & & & \ddots & 0 \\ 1 & 0 & \dots & 0 & 1 \end{bmatrix} \begin{bmatrix} w_0 \\ w_1 \\ \vdots \\ w_d \end{bmatrix} = \begin{bmatrix} y_1 \\ y_2 \\ \vdots \\ y_{d+1} \end{bmatrix} = \begin{bmatrix} \pm 1 \\ \pm 1 \\ \vdots \\ \pm 1 \end{bmatrix}$$

We ask: Is this system of linear equations solvable? This in turns asks: Is the $(d+1)\times(d+1)$ matrix invertible? And clearly the matrix is invertible. Therefore, we will be able to find $\boldsymbol{w}$ for any given $\boldsymbol{y}$, and hence we prove that $d_{\text{VC}} \geq d+1$.

For the other direction to show that $d_{\text{VC}} \leq d+1$, we need to show that we are not able to shatter any set of $d+2$ data points. Suppose we have $d+2$ data points $\boldsymbol{x}_1, \dots, \boldsymbol{x}_{d+1}, \boldsymbol{x}_{d+2}$, we can write

$$\boldsymbol{x}_{d+2} = \sum_{j=1}^{d+1} \alpha_j \boldsymbol{x}_j.$$

Construct a dichotomy with labels

$$y_i = \begin{cases} \text{sign}(\alpha_i), & i = 1, \dots, d+1, \\ -1, & i = d+2. \end{cases}$$

Assume that all labels correct so that $\text{sign}(\alpha_i) = \text{sign}(\boldsymbol{w}^T\boldsymbol{x}_i)$ for $i = 1, \dots, d+1$. This implies that $\alpha_i^T \boldsymbol{w}^T \boldsymbol{x}_i > 0$ for all $i = 1, \dots, d+1$. Because of the linear combination, we also have that $\boldsymbol{w}^T\boldsymbol{x}_{d+2} > 0$. But $y_{d+2} = \text{sign}(\boldsymbol{w}^T\boldsymbol{x}_{d+2}) = -1$. So there is a contradiction. Therefore, we cannot shatter $d+2$ data points using a d-dimensional perceptron algorithm. This completes the proof. □

Bounding the Growth Function

Now that we have the VC dimension, we can bound the growth function. The following theorem show that $m_{\mathcal{H}}(N)$ is indeed upper bounded by a polynomial of order no greater than d_{VC}.

Theorem 5. *Let d_{VC} be the VC dimension of a hypothesis set $\mathcal{H}$, then*

$$m_{\mathcal{H}}(N) \leq \sum_{i=0}^{d_{\text{VC}}} \binom{N}{i}. \tag{4.13}$$

We shall skip the proof which can be found in Theorem 2.4 of AML's Learning from Data textbook. The polynomial bound comes from the following exercise.

Exercise. Prove by induction that

$$\sum_{i=0}^{d} \binom{N}{i} \leq N^d + 1.$$

Using this result, we can show that

$$m_{\mathcal{H}}(N) \leq N^{d_{\text{VC}}} + 1.$$

If we substitute $m_{\mathcal{H}}(N)$ by this upper bound $N^{d_{\text{VC}}}+1$, then the generalization bound becomes

$$\epsilon = \sqrt{\frac{1}{2N} \log \frac{2m_{\mathcal{H}}(N)}{\delta}} \leq \sqrt{\frac{1}{2N} \log \frac{2(N^{d_{\text{VC}}} + 1)}{\delta}}. \tag{4.14}$$

How do we interpret the VC dimension? The VC dimension can be informally viewed as the **effective number of parameters** of a model. Higher VC dimension is means a more complex model, and hence a more diverse hypothesis set $\mathcal{H}$. As a result, the growth function $m_{\mathcal{H}}(N)$ will be big. (Think about the number of dichotomies that can be generated by a complex model versus a simple model, and hence the overlap we encounter in the union bound.) There are two scenarios of the VC dimension.

- $d_{\text{VC}} < \infty$. This implies that the generalization error will go to zero as N grows:

  $$\epsilon = \sqrt{\frac{1}{2N} \log \frac{2(N^{d_{\text{VC}}} + 1)}{\delta}} \to 0,$$

  as $N \to \infty$ because $(\log N)/N \to 0$. If this is the case, then the final hypothesis $g \in \mathcal{H}$ will generalize. Such generalization result holds independent of the learning algorithm $\mathcal{A}$, independent of the input distribution $p_{\boldsymbol{X}}$ and independent of the target function f. It only depends on the hypothesis set $\mathcal{H}$ and the training examples $\boldsymbol{x}_1, \ldots, \boldsymbol{x}_N$.
- $d_{\text{VC}} = \infty$. This means that the hypothesis set $\mathcal{H}$is as diverse as it can be, and it is not possible to generalize. The generalization error will never go to zero.

Are we all set about the generalization bound? It turns out that we need some additional technical modifications to ensure the validity of the generalization bound. We shall not go into the details but just state the result.

Theorem 6 (Generalization Bound)**.** *For any tolerance* $\delta > 0$

$$E_{\text{out}}(g) \leq E_{\text{in}}(g) + \sqrt{\frac{8}{N} \log \frac{4m_{\mathcal{H}}(2N)}{\delta}}, \tag{4.15}$$

with probability at least $1 - \delta$.

Interpreting the Generalization Bound

The VC generalization bound in Equation (**??**) is universal in the sense that it applies to all hypothesis set $\mathcal{H}$, learning algorithm $\mathcal{A}$, input space $\mathcal{X}$, distribution p, and binary target function f. So can we use the VC generalization bound to predict the exact generalization error for any learning scenario? Unfortunately the answer is no. The VC generalization bound we derived is a valid upper bound but also a very loose upper bound. The loose-ness nature of the generalization bound comes from the following reasons (among others):

- The Hoeffding inequality has a slack. The inequality works for all values of E_{out}. However, the behavior of E_{out} could be very different at different values, e.g., at 0 or at 0.5. Using one bound to capture both cases will result in some slack.
- The growth function $m_{\mathcal{H}}(N)$ gives the **worst case** scenario of how many dichotomies are there. If we draw the N data points at random, it is unlikely that we will land on the worst case, and hence the typical value of $m_{\mathcal{H}}(N)$ could be far fewer than 2^N even if $m_{\mathcal{H}}(N) = 2^N$.
- Bounding $m_{\mathcal{H}}(N)$ by a polynomial introduces further slack.

Therefore, the VC generalization bound can only be used a rough guideline of understanding how well the learning algorithm generalize.

Sample Complexity

Sample complexity concerns about the number of training samples N we need to achieve the generalization performance. Recall from the generalization bound:

$$E_{\text{out}}(g) \leq E_{\text{in}}(g) + \sqrt{\frac{8}{N}\log\frac{4m_{\mathcal{H}}(2N)}{\delta}}.$$

Fix a $\delta > 0$, if we want the generalization error to be at most ϵ, we can enforce that

$$\sqrt{\frac{8}{N}\log\frac{4m_{\mathcal{H}}(2N)}{\delta}} \leq \epsilon.$$

Rearranging the terms yields $N \geq \frac{8}{\epsilon^2}\log\left(\frac{4m_{\mathcal{H}}(2N)}{\delta}\right)$. If we replace $m_{\mathcal{H}}(2N)$ by the VC dimension, then we obtain a similar bound

$$N \geq \frac{8}{\epsilon^2}\log\left(\frac{4(2N)^{d_{\text{VC}}}+1}{\delta}\right).$$

Example. Suppose $d_{\text{VC}} = 3$, $\epsilon = 0.1$ and $\delta = 0.1$ (90% confidence). The number of samples we need satisfies the equation

$$N \geq \frac{8}{0.1^2}\log\left(\frac{4(2N)^3+4}{0.1}\right).$$

If we plug in $N = 1000$ to the right hand side, we will obtain

$$N \geq \frac{8}{0.1^2} \log \left(\frac{4(2 \times 1000)^3 + 4}{0.1} \right) \approx 21,193.$$

If we repeat the calculation by plugging in $N = 21,193$, obtain a new N, and iterate, we will eventually obtain $N \approx 30,000$. If $d_{\text{VC}} = 4$, we obtain $N \approx 40,000$ samples. This means that every value of d_{VC} corresponds to 10,000 samples. In practice, we may require significantly less number of samples. A typical number of samples is approximately $10 \times d_{\text{VC}}$.

Model Complexity

The other piece of information that can be obtained from the generalization bound is how complex the model could be. If we look at the generalization bound, we realize that the error ϵ is a function of N, $\mathcal{H}$ and δ:

$$E_{\text{out}}(g) \leq E_{\text{in}}(g) + \underbrace{\sqrt{\frac{8}{N} \log \frac{4m_{\mathcal{H}}(2N)}{\delta}}}_{=\epsilon(N,\mathcal{H},\delta)}$$

If we replace $m_{\mathcal{H}}(2N)$ by $(2N)^{d_{\text{VC}}} + 1$, then we can write $\epsilon(N, \mathcal{H}, \delta)$ as

$$\epsilon(N, d_{\text{VC}}, \delta) = \sqrt{\frac{8}{N} \log \left(\frac{4\left((2N)^{d_{\text{VC}}} + 1\right)}{\delta} \right)}$$

The three factors N, d_{VC} and δ have different influence on the error ϵ:

- d_{VC}: The VC dimension controls the complexity of the model. As d_{VC} grows, the in-sample error E_{in} drops because large d_{VC} implies that we have a more complex model to fit the training data. However, ϵ grows as d_{VC} grows. If we have a very complex model, then it would be more difficult to generalize to the out-samples. The trade-off between model complexity and generalization is shown in Figure **??**. The blue curve represents the in-sample error E_{in} which drops as d_{VC} increases. The red curve represents the model complexity which increases as d_{VC} increases. The black curve is the out-sample error E_{out}. There exists an optimal model complexity so that E_{out} is minimized.

- N: A large number of training samples always helps the generalization bound, as reflected by the fact that $\epsilon(N, \mathcal{H}, \delta) \to 0$ as $N \to \infty$.

- δ: The confidence level tells us how harsh we want the generalization to be. If we want a very high confidence interval, e.g., 99.99%, then we need a very small $\delta = 0.0001$. This will in turn affect the number of training samples N required to achieve the confidence level and the desired error bound.

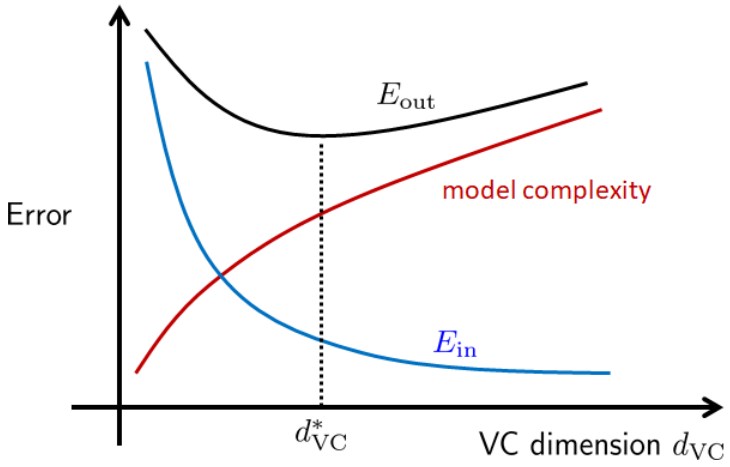

Figure 4.6: The VC generalization bound suggests a trade-off between model complexity and generalization. If we use a more complex model, the in-sample error drops but the out-sample error increases. The optimal model complexity is determined when the out-sample error is minimized.

Testing Data

The VC analysis provides us a good guideline to train a model. However, the estimate provided by the VC analysis is often too loose to provide any accurate prediction of E_{out}. In practice, no one really uses VC analysis to inform a training process. What is more often used is a testing dataset. The testing dataset

$$\mathcal{D}_{\text{test}} = \{\boldsymbol{x}_1, \ldots, \boldsymbol{x}_L\}$$

contains L samples drawn from the distribution $p_{\boldsymbol{X}}(\boldsymbol{x})$. No testing data $\boldsymbol{x}_m$ can be in the training dataset $\mathcal{D}_{\text{training}}$.

Since in the testing phase the final hypothesis g is already determined, we will not run into the same trouble in the training phase where we need to use the Union bound to account for the M candidate hypotheses in $\mathcal{H}$. As a result, the Hoeffding inequality simplifies to

$$\mathbb{P}\Big\{ |E_{\text{in}}(g) - E_{\text{out}}(g)| > \epsilon \Big\} \leq 2e^{-2\epsilon^2 L},$$

and the generalization bound becomes

$$E_{\text{out}}(g) \leq E_{\text{in}}(g) + \sqrt{\frac{1}{2L}\log\frac{2}{\delta}}.$$

Therefore, as the number of testing samples increases, we can certify the out-sample error by evaluating $E_{\in}$ using the testing samples.

There are a few reminders about using the testing data:

- The common notion of **testing accuracy** is $E_{\text{in}}(g)$, calculated based on the L testing samples. Therefore, having $E_{\text{in}}(g)$ does not imply that we will generalize well. If we change another testing dataset, $E_{\text{in}}(g)$ will change because it is a numerical value based on empirical sum. What is guaranteed by the generalization bound is that as long as L is sufficiently large, $E_{\text{out}}(g)$ will stay close to $E_{\text{in}}(g)$ no matter which particular testing dataset we use. There is a variance associated with $E_{\text{in}}(g)$, and this variance is reflected by $\sqrt{\frac{1}{2L}\log\frac{2}{\delta}}$.

- The testing data has to be used *after* the hypothesis is determined. If we ever use the testing data as a feedback to re-select the hypothesis, then it is cheating. For example, we cannot train a SVM, submit to a competition website, and mark the misclassified samples to re-design the SVM.

- In principle the generalization bound is improved when we have more testing samples. However, most practical datasets only have training data points and no testing data points. We can partition the training set into training and validation. The proportion of training and validation needs to be carefully chosen. If we allocate too many samples for validation purpose, then we will loose our ability to training a good classifier.

4.4 Bias and Variance Analysis

The bias-variance analysis is an alternative way of analyzing the out-sample error. Instead of defining the out-sample error as the probability $E_{\text{out}}(g) = \mathbb{P}[g(\boldsymbol{x}) \neq f(\boldsymbol{x})]$, bias-variance analysis defines the out-sample error using the squared error:

$$E_{\text{out}}(g^{(\mathcal{D})}) = \mathbb{E}_{\boldsymbol{x}}\Big[(g^{(\mathcal{D})}(\boldsymbol{x}) - f(\boldsymbol{x}))^2\Big]. \tag{4.16}$$

One thing to note is that we make the dependency on the training dataset $\mathcal{D}$ explicit. The reason will be clear later. What this means is that if we use a different training set $\mathcal{D}$, we will get a different $E_{\text{out}}(g^{(\mathcal{D})})$. This will give us many $E_{\text{out}}(g^{(\mathcal{D})})$, depending on how the training sets $\mathcal{D}$'s are generated. To account for all the possible $\mathcal{D}$'s, we can compute the expectation and define the expected out-sample error:

$$\begin{aligned}
\mathbb{E}_{\mathcal{D}}\left[\mathbb{E}_{\text{out}}(g^{(\mathcal{D})})\right] &= \mathbb{E}_{\mathcal{D}}\left[\mathbb{E}_{\boldsymbol{x}}\Big[(g^{(\mathcal{D})}(\boldsymbol{x}) - f(\boldsymbol{x}))^2\Big]\right] \\
&= \mathbb{E}_{\boldsymbol{x}}\left[\mathbb{E}_{\mathcal{D}}\Big[(g^{(\mathcal{D})}(\boldsymbol{x}) - f(\boldsymbol{x}))^2\Big]\right] \\
&= \mathbb{E}_{\boldsymbol{x}}\left[\mathbb{E}_{\mathcal{D}}\Big[g^{(\mathcal{D})}(\boldsymbol{x})^2\Big] - 2\underbrace{\mathbb{E}_{\mathcal{D}}[g^{(\mathcal{D})}(\boldsymbol{x})]}_{\bar{g}(\boldsymbol{x})}f(\boldsymbol{x}) + f(\boldsymbol{x})^2\right].
\end{aligned}$$

Here, we define $\overline{g}(\boldsymbol{x}) = \mathbb{E}_{\mathcal{D}}[g^{(\mathcal{D})}(\boldsymbol{x})]$, which can be considered as the asymptotic limit of the estimate $\overline{g}(\boldsymbol{x}) \approx \frac{1}{K}\sum_{k=1}^{K} g_k(\boldsymbol{x})$ as $K \to \infty$. The hypotheses $g_1, \ldots, g_K$ are the final hypothesis returned by using the training sets $\mathcal{D}_1, \ldots, \mathcal{D}_K$. Therefore, for any fixed $\boldsymbol{x}$, $g_k(\boldsymbol{x})$ is a random variable over the training set $\mathcal{D}_k$. However, one should be careful that even if $g_1, \ldots, g_K$ are inside the hypothesis set, the mean $\overline{g}$ is *not* necessarily inside too.

Let us do some additional calculation:

$$\begin{aligned}
& \mathbb{E}_{\mathcal{D}}\left[\mathbb{E}_{\text{out}}(g^{(\mathcal{D})})\right] \\
= \; & \mathbb{E}_{\boldsymbol{x}}\left[\mathbb{E}_{\mathcal{D}}\left[g^{(\mathcal{D})}(\boldsymbol{x})^2\right] - 2\mathbb{E}_{\mathcal{D}}[g^{(\mathcal{D})}(\boldsymbol{x})]f(\boldsymbol{x}) + f(\boldsymbol{x})^2\right] \\
= \; & \mathbb{E}_{\boldsymbol{x}}\Big[\underbrace{\mathbb{E}_{\mathcal{D}}\left[g^{(\mathcal{D})}(\boldsymbol{x})^2\right] - \overline{g}(\boldsymbol{x})^2}_{\mathbb{E}_{\mathcal{D}}[(g^{(\mathcal{D})}(\boldsymbol{x})-\overline{g}(\boldsymbol{x}))^2]} + \underbrace{\overline{g}(\boldsymbol{x})^2 - 2\mathbb{E}_{\mathcal{D}}[g^{(\mathcal{D})}(\boldsymbol{x})]f(\boldsymbol{x}) + f(\boldsymbol{x})^2}_{(\overline{g}(\boldsymbol{x})-f(\boldsymbol{x}))^2}\Big].
\end{aligned}$$

Based on this decomposition, we can define two terms:

$$\begin{aligned}
\text{bias}(\boldsymbol{x}) &\stackrel{\text{def}}{=} (\overline{g}(\boldsymbol{x}) - f(\boldsymbol{x}))^2, \\
\text{var}(\boldsymbol{x}) &\stackrel{\text{def}}{=} \mathbb{E}_{\mathcal{D}}[(g^{(\mathcal{D})}(\boldsymbol{x}) - \overline{g}(\boldsymbol{x}))^2].
\end{aligned}$$

The first term is called the **bias**, as it measures the deviation between the average function $\overline{g}(\boldsymbol{x})$ and the target function $f(\boldsymbol{x})$. Thus, regardless of how we pick the particular training set, there is an intrinsic gap between the what we would expect ($\overline{g}(\boldsymbol{x})$) and the ideal target $f(\boldsymbol{x})$. The second term is called the **variance**. It measures the variance of the random variable $g^{(\mathcal{D})}(\boldsymbol{x})$ with respect to its mean $\overline{g}(\boldsymbol{x})$. Using the bias and variance decomposition, we can show that

$$\begin{aligned}
\mathbb{E}_{\mathcal{D}}\left[\mathbb{E}_{\text{out}}(g^{(\mathcal{D})})\right] &= \mathbb{E}_{\boldsymbol{x}}[\text{bias}(\boldsymbol{x}) + \text{var}(\boldsymbol{x})] \\
&= \text{bias} + \text{var},
\end{aligned}$$

where $\text{bias} = \mathbb{E}_{\boldsymbol{x}}[\text{bias}(\boldsymbol{x})]$ is the average bias over the distribution $p(\boldsymbol{x})$, and $\text{var} = \mathbb{E}_{\boldsymbol{x}}[\text{var}(\boldsymbol{x})]$ is the average variance over $p(\boldsymbol{x})$.

What can we say about the bias-variance decomposition when analyzing the model complexity? We can consider two extreme cases. In the first case, we have a very simple model and so $\mathcal{H}$ is small. Since there are not many choices of the hypothesis, the deviation between the target f and the average of these hypotheses $\overline{g}$ is large. Thus, the bias is large. On the other hand, the variance is limited because we only have very few hypotheses in $\mathcal{H}$.

The second case is when we have a complex model. By selecting different training sets $\mathcal{D}$'s, we will be able to select hypothesis functions $g_1, \ldots, g_K$ that agree with f. In this case, the deviation between the target f and the average of these hypotheses $\overline{g}$ is very small. The bias is thus $\text{bias} \approx 0$. The variance, however, is large because there are many training sets under consideration.

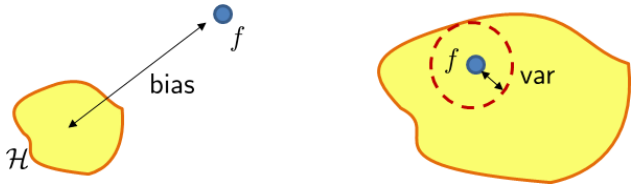

Figure 4.7: [Left] Large bias but small variance. [Right] Small bias but large variance.

Demonstration

Consider a target function $f(x) = \sin(\pi x)$ and a dataset of size $N = 2$. We sample uniformly in the interval $[-1, 1]$ to generate a data set containing two data points (x_1, y_1) and (x_2, y_2). We want to use these two data points to determine which of the following two models are better:

- $\mathcal{M}_0$ = Set of all lines of the form $h(x) = b$;
- $\mathcal{M}_1$ = Set of all lines of the form $h(x) = ax + b$.

Figure **??** illustrates an example of how the models would yield the lines. Given two data points, $\mathcal{M}_0$ seeks a horizontal line $h(x) = b$ that matches the two data points. This line must be the one that passes through the mid-point of the two data points. The model $\mathcal{M}_1$ is allowed to find an arbitrary straight line that matches the two data points. Since there are only two data points, the best straight must be the one that passes through both of them. More specifically, the line returned by $\mathcal{M}_0$ is

$$h(x) = \frac{y_1 + y_2}{2},$$

and the line returned by $\mathcal{M}_1$ is

$$h(x) = \left(\frac{y_2 - y_1}{x_2 - x_1}\right) x + (y_1 x_2 - y_2 x_1)\,.$$

As we change (x_1, y_1) and (x_2, y_2), we will obtain different straight lines.

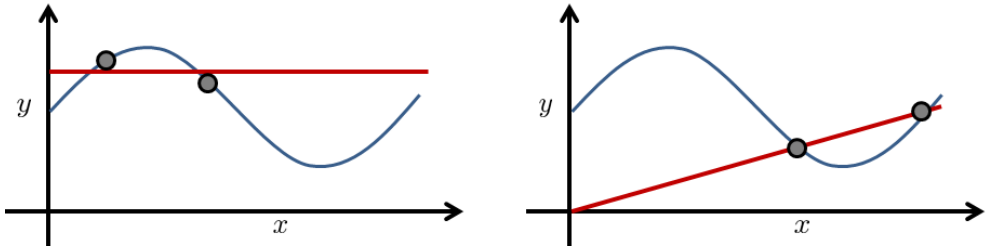

Figure 4.8: [Left] Fitting two data points using $\mathcal{M}_0$. [Right] Fitting two data points using $\mathcal{M}_1$.

If we keep drawing two random samples from the sine function, we will eventually get a set of straight lines for both cases. However, since $\mathcal{M}_0$ restricts ourselves to horizontal lines, the set of straight lines are all horizontal. In contrast, the set of straight lines for $\mathcal{M}_1$ contains lines of different slopes and y-intercepts.

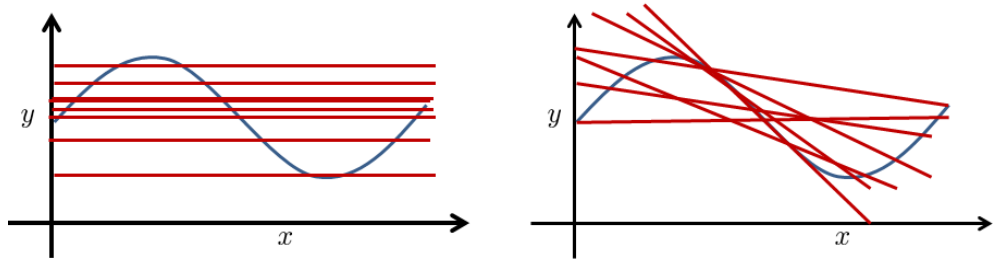

Figure 4.9: [Left] Possible lines generated by $\mathcal{M}_0$. [Right] Possible lines generated by $\mathcal{M}_1$.

As we increase the number of experiments, the set of straight lines will form a distribution of the model. Since now we have a distribution, we can determine is mean, which is a function, as $\overline{g}$. Similarly, we can determine the variance of the function $\text{var}(x)$. For example, in Figure **??** we draw the possible lines that are within one standard deviation from the mean function, i.e., $\overline{g} \pm \sqrt{\text{var}(x)}$.

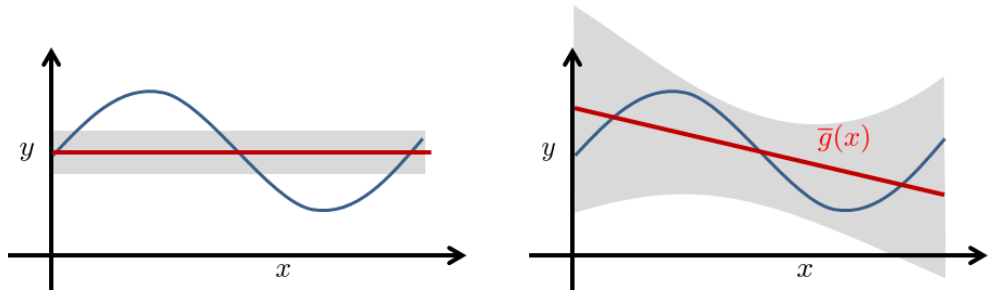

Figure 4.10: Average hypothesis function $\overline{g}(x)$ and the variance $\text{var}(x)$.

So which model is better in terms of bias-variance? If we compute the bias and variance, we can show that

$$\begin{aligned} \text{bias}_{\mathcal{M}_0} &= 0.5, & \text{bias}_{\mathcal{M}_1} &= 0.21, \\ \text{var}_{\mathcal{M}_0} &= 0.25, & \text{var}_{\mathcal{M}_1} &= 1.69. \end{aligned}$$

Therefore, as far as generalization is concerned, a simple model using a horizontal line is actually more preferred in the bias-variance sense. This is counter-intuitive because how can a horizontal line with only one degree of freedom be better than a line with two degrees of freedom when approximating the sine function? However, the objective here is not to use a line to approximate a sine function because we are not supposed to observe the entire sine function. Remember, we are only allowed to see two data points and our goal is to

construct a line based on these two data points. The approximation error in the usual sense is captured by the bias, as $\overline{g}$ is the best possible line within the class. The generalization, however, should also take into account of the variance. While $\mathcal{M}_1$ has a lower bias, its variance is actually much larger than that of $\mathcal{M}_0$. The implication is that while on average $\mathcal{M}_1$ performs well, chances are we pick a bad line in $\mathcal{M}_1$ that end up causing very undesirable out-sample performance.

One thing to pay attention to is that the above analysis is based on $N = 2$ data points. If we increase the number of data points, the variance of $\mathcal{M}_1$ will drop. As $N \to \infty$, the variance of both $\mathcal{M}_0$ and $\mathcal{M}_1$ will eventually drop to zero and so only the bias term matters. Therefore, if we have infinitely many training data, a complex model will of course provide a better generalization.

Bias-Variance and VC Dimension

There is a subtle but important difference between the bias-variance analysis and the VC analysis. Bias-variance depends on the learning algorithm $\mathcal{A}$ whereas the VC analysis is independent of $\mathcal{A}$. With the same hypothesis set $\mathcal{H}$, VC will always return the same generalization bound. This is a uniform performance guarantee over all possible choices of dataset $\mathcal{D}$. For bias-variance, the same $\mathcal{H}$ can lead to different $g^{(\mathcal{D})}$, depending of which $\mathcal{D}$ is being used. This is reflected in the bias and variance term $E_{\text{out}}(g^{(\mathcal{D})})$. Of course, the overall bias-variance is independent of $\mathcal{D}$ because we take expectation $\mathbb{E}_{\mathcal{D}}\left[E_{\text{out}}(g^{(\mathcal{D})})\right]$. VC analysis does not have this issue.

In practice, bias and variance cannot be computed because we never have the target function. (If we know the target function there is nothing to learn!) Therefore, bias-variance can only be served as a conceptual tool to guide the design of a learning algorithm. For example, one can try to reduce the bias but maintaining the variance (e.g., via regularization and prior), or reduce the variance but maintaining the bias.

Learning Curve

Both bias-variance and VC analysis provide a trade-off between model complexity and sample complexity. Figure **??** shows a typical scenario. Suppose that we have learned an final hypothesis $g^{(\mathcal{D})}$ using dataset $\mathcal{D}$ of size N. This final hypothesis will give us an in-sample error $E_{\text{in}}(g^{(\mathcal{D})})$ and out-sample error $E_{\text{out}}(g^{(\mathcal{D})})$. These two errors are functions of the dataset $\mathcal{D}$. If we take the expectation over $\mathcal{D}$, we will obtain the expected error $\mathbb{E}_{\mathcal{D}}\left[E_{\text{in}}(g^{(\mathcal{D})})\right]$ and $\mathbb{E}_{\mathcal{D}}\left[E_{\text{out}}(g^{(\mathcal{D})})\right]$. These expected error will give us two curves, as shown in Figure **??**.

If we have a simple model, the in-sample error $\mathbb{E}_{\mathcal{D}}\left[E_{\text{in}}(g^{(\mathcal{D})})\right]$ is a good approximate of the out-sample error $\mathbb{E}_{\mathcal{D}}\left[E_{\text{out}}(g^{(\mathcal{D})})\right]$. This implies a small gap between the two. However, the overall expected error could still be large because our model is simple. This is reflected in the high off-set in the learning curve.

If we have a complex model. the in-sample error $\mathbb{E}_{\mathcal{D}}\left[E_{\text{in}}(g^{(\mathcal{D})})\right]$ would be small because we are able to fit the training data. However, the out-sample error is large $\mathbb{E}_{\mathcal{D}}\left[E_{\text{out}}(g^{(\mathcal{D})})\right]$ because the generalization using a complex model is difficult. The two curves will eventually meet as N grows, since the variance of the out-sample will drop. The convergence rate is slower than a simple model, because it take many more samples for a complex model to generalize well.

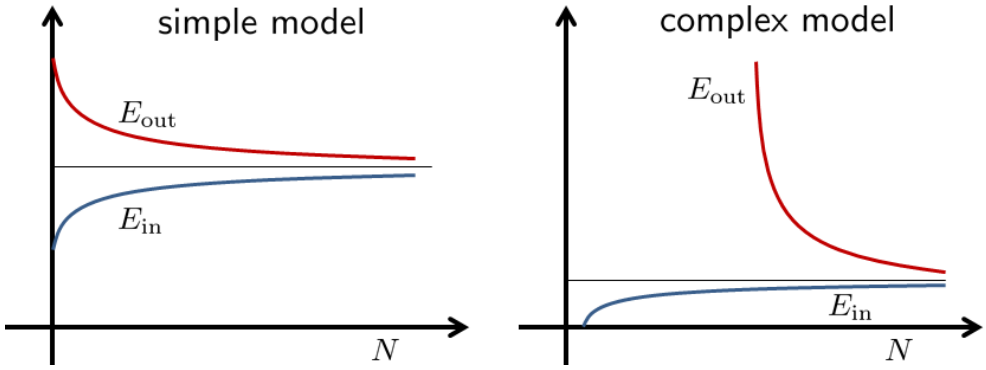

Figure 4.11: Learning curves of a simple and a complex model.

The VC analysis and the bias-variance analysis provide two different view of decomposing the error. VC analysis decompose E_{out} as the in-sample error $E_{\text{in}}(g)$ and the generalization error ϵ. This ϵ is the gap between E_{in} and E_{out}. The bias-variance analysis decompose E_{out} as bias and variance. The bias is the residue caused by the average hypothesis $\bar{g}$. The bias is a fixed quantity and does not change over N. The gap between E_{out} and the bias is the variance. The variance drops as N increases.

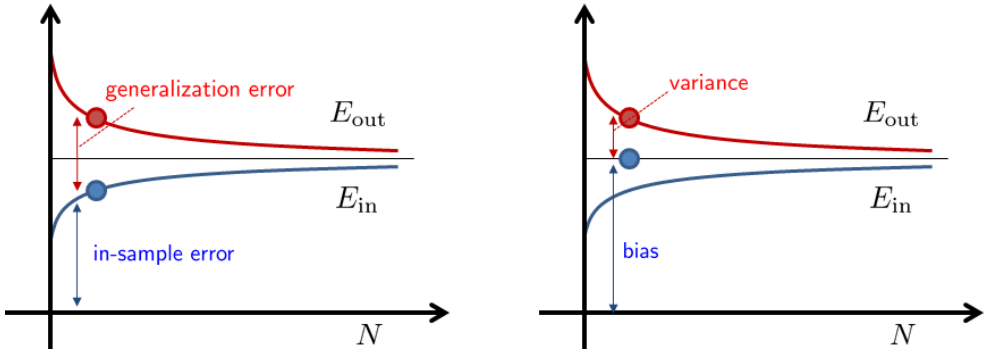

Figure 4.12: [Left] VC analysis. [Right] Bias-variance analysis

4.5 Validation

4.6 Practical Considerations

