ECE 50024 Machine Learning
Stanley Chan
stanchan@purdue.edu

PURDUE
UNIVERSITY

Part 4 Learning Theory

April 20, 2023

Contents

1 Introduction

1.1 Motivation

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:

How do people think about evaluating a classifier?

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.

What are the objectives of this chapter?

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?

1.2 Notations

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

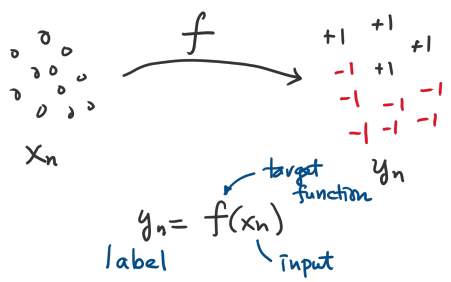


Figure 1: Input and output in a learning model.

In any supervised learning scenario, there is always a **training set** $\mathcal{D}$. The training set contains N input-output pairs

$$(\mathbf{x}_1, y_1), \ldots, (\mathbf{x}_N, y_N),$$

where $\mathbf{x}_n$ and y_n are related via $y_n = f(\mathbf{x}_n)$, 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 $\mathbf{x} \in \mathcal{X}$ that do not live in the training set $\mathcal{D}$. A data point $\mathbf{x}_n$ that is inside $\mathcal{D}$ is called an **in-sample**, and a data point $\mathbf{x}$ that is outside $\mathcal{D}$ is called an **out-sample**.

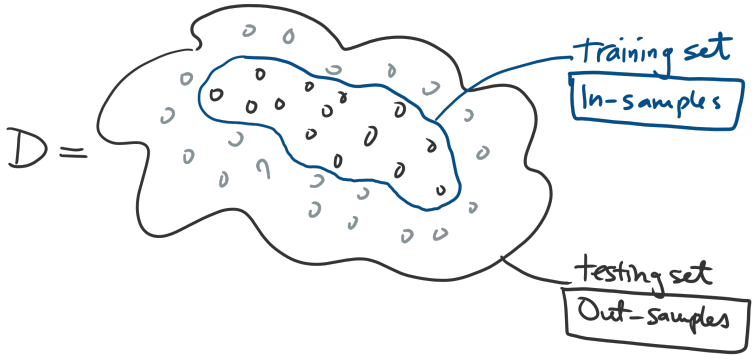


Figure 2: In samples and out samples.

What are in-samples and out-samples?

- In-samples are data points $\mathbf{x}_n$ that live inside the training set $\mathcal{D}$. So, in-sample = training data points.
- Out-samples are data points $\mathbf{x}_n$ that are not present in the training set $\mathcal{D}$. So out-sample = testing data points.

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 $\mathbf{x}$ 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.

What are hypothesis functions?

- They are the candidate classifiers you want evaluate. They are denoted as $h_1, \ldots, h_M$.
- The collection of these hypothesis functions is the set $\mathcal{H} = \{h_1, \ldots, h_M\}$.
- The job of a learning algorithm $\mathcal{A}$ is to pick the "best" hypothesis function from $\mathcal{H}$. "Best" depends on how you define.

The drawings in Figure 3 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 $\mathbf{x}_n$ to produce an output $y_n = f(\mathbf{x}_n)$, 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.

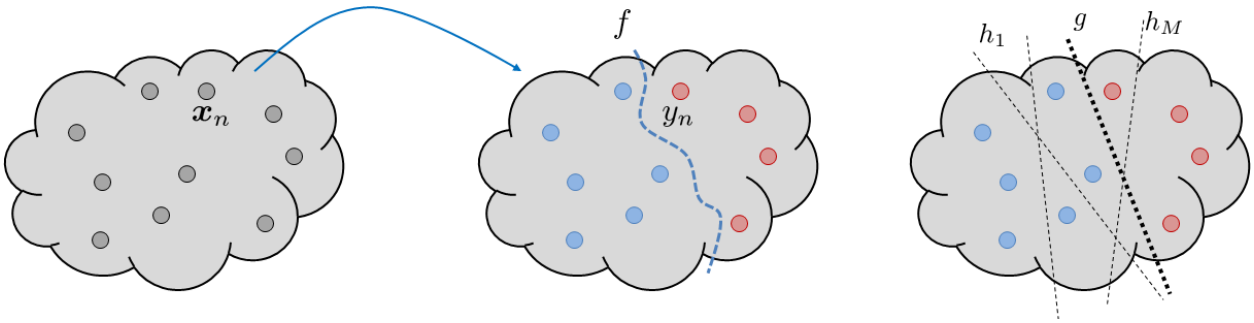


Figure 3: [Left] Treat the cloud as the entire input space $\mathcal{X}$ and correspondingly the output space $\mathcal{Y}$. The dots are the **in-samples** $\mathbf{x}_1, \ldots, \mathbf{x}_N$. The target function is a mapping f which takes $\mathbf{x}_n$ and send it to y_n. The red and blue colors 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 4 illustrates what we called a **probabilistic** learning model. It is called a probabilistic learning model because there is an unknown distribution $p(\mathbf{x})$. The training samples $\{\mathbf{x}_1, \ldots, \mathbf{x}_N\}$ are generated according to $p(\mathbf{x})$. The same $p(\mathbf{x})$ also generates the testing samples $\mathbf{x}$. 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 $\{\mathbf{x}_1, \ldots, \mathbf{x}_N\}$. The deterministic assumption will make learning infeasible, as we will see shortly. Therefore, we shall mainly focus on the probabilistic assumption.

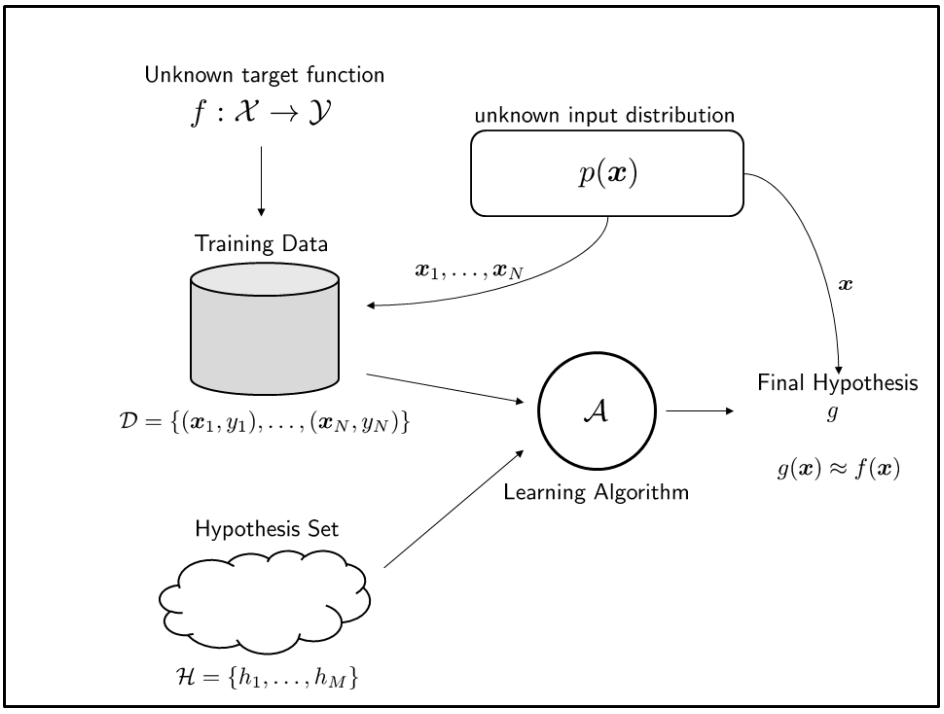


Figure 4: All essential components of a machine learning model.

Why need a probability distribution?

- To link the in-samples and the out-samples.
- We assume that all sample $\mathbf{x}_1, \ldots, \mathbf{x}_N$ are drawn from the distribution $p(\mathbf{x})$.
- If the in-samples are out-samples do not come from the same distribution, there is no way you can learn.

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 $\mathbf{x}_n$'s. If $\mathbf{x}_n$'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 $\mathbf{x}_n$'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.

2.1 Learning outside the training set $\mathcal{D}$

Let us look at the deterministic case. Consider a 3-dimensional input space $\mathcal{X} = \{0,1\}^3$. Each vector $\mathbf{x} \in \mathcal{X}$ is a binary vector containing three elements, e.g., $\mathbf{x} = [0,\ 0,\ 1]^T$ or $\mathbf{x} = [1,\ 0,\ 1]^T$.

Number of input vectors. Since there are 3 elements and each element take a binary state, there are totally $2^3 = 8$ input vectors in $\mathcal{X}$.

Number of target functions. 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 $\mathbf{x}$ to a label y. For simplicity let us assume that f maps $\mathbf{x}$ 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.

What is the learning problem? 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 $\{\mathbf{x}_1, \ldots, \mathbf{x}_6\}$. Corresponding to each $\mathbf{x}_n$ is the label y_n. The relationship between $\mathbf{x}_n$ and y_n is shown in the table below. So our task is to pick a target function from the 256 possible choices that can best predict the two unknown cases.

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

Figure 5: The training set $\mathcal{D}$ for our example. We have 6 training samples $(\mathbf{x}_1, y_1), \ldots, (\mathbf{x}_6, y_6)$. Our goal is to pick the best hypothesis function that can explain these known observations.

Possibility 1. The first possible choice of the final hypothesis function g is shown in the Figure below. Since one 1's gives us $\bullet$ and anything with more than one 1's gives us $\circ$, the remaining two cases must be $\circ$ and $\circ$, respectively, because $[1,1,0]$ has more than one 1's, and $[1,1,1]$ also has more than one 1's.

$\boldsymbol{x}_n$			y_n
0	0	0	$\circ$
0	0	1	$\bullet$
0	1	0	$\bullet$
0	1	1	$\circ$
1	0	0	$\bullet$
1	0	1	$\circ$
1	1	0	$\circ$
1	1	1	$\circ$

Figure 6: Possibility 1: We check whether the number of 1's is more than one. If no, we predict $\bullet$. If yes, we predict $\circ$.

Possibility 2. The second possible choice of the final hypothesis function g is shown in the Figure below. Since an odd number of 1's gives us $\bullet$ and an even number of 1's gives us $\circ$, the remaining two cases must be $\circ$ and $\bullet$, respectively, because $[1,1,0]$ is even, and $[1,1,1]$ is odd.

$\boldsymbol{x}_n$			y_n
0	0	0	$\circ$
0	0	1	$\bullet$
0	1	0	$\bullet$
0	1	1	$\circ$
1	0	0	$\bullet$
1	0	1	$\circ$
1	1	0	$\circ$
1	1	1	$\bullet$

Figure 7: Possibility 2: We check whether the number of 1's odd or even. If even, we predict $\bullet$. If odd, we predict $\circ$.

General case. 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 below 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 "$\circ$", then we will have both "$\circ$". If the learning algorithm decides a "$\circ$" followed by a "$\bullet$", 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%

$\boldsymbol{x}_n$			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		○/●	○	○	●	●

Figure 8: All possible choices

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 $\mathcal{D}$.

$\mathbf{x}_n$			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		○	○	○	●	●

$\mathbf{x}_n$			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		○	○	○	●	●

Table 1: Suppose we say that the final hypothesis g is the one shown above. If the true function is f_1, then we are correct. But if the true function is f_3, we are wrong. Since there is no additional information regarding how the training samples are related to the testing samples, there is no way we can differentiate the better or worse of f_1 and f_3. So we end up with a lottery draw, and this defeats the purpose of learning.

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 $\mathcal{D}$.

What have we learned from this example?

- If the training samples and the testing samples are drawn deterministically from an unknown process, there is no way a machine learning can learn anything meaningful.
- For a machine learning algorithm to work, you need the training samples and the testing samples to be correlated via some probability distributions.

2.2 In-sample and out-sample error

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 $\mathbf{x} \in \mathcal{X}$ is drawn from a distribution $p_{\mathbf{X}}(\mathbf{x})$. This includes all the in-samples $\mathbf{x}_n \in \mathcal{D}$ and the out-samples $\mathbf{x} \in \mathcal{X}$. At a first glance, putting a distributional assumption $p_{\mathbf{X}}(\mathbf{x})$ does not seem any different from the deterministic case: We still have a

training set $\mathcal{D} = \{\mathbf{x}_1, \ldots, \mathbf{x}_N\}$, 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 $\mathbf{x}_n$, we check whether the output returned by h is the same as the output returned by f, i.e., $\{h(\mathbf{x}_n) = f(\mathbf{x}_n)\}$, for $n = 1, \ldots, N$. If $\{h(\mathbf{x}_n) = f(\mathbf{x}_n)\}$, then we say that the in-sample $\mathbf{x}_n$ is correctly classified in the training. If $\{h(\mathbf{x}_n) \neq f(\mathbf{x}_n)\}$, then we say that $\mathbf{x}_n$ 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 $\mathcal{D} = \{\mathbf{x}_1, \ldots, \mathbf{x}_N\}$, 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 $\{h(\mathbf{x}_n) \neq f(\mathbf{x}_n)\}$:*

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

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.

What is "in-sample" error?

- In-sample error is the training error.
- It is a finite average of the error accumulated in the training dataset.
- In-sample error can be computed from the training dataset.

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

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

$$E_{\text{out}}(h) \stackrel{def}{=} \mathbb{P}[h(\mathbf{x}) \neq f(\mathbf{x})], \tag{2}$$

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

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

$$\begin{aligned} E_{\text{out}}(h) &= \mathbb{P}[h(\mathbf{x}) \neq f(\mathbf{x})] \\ &= \underbrace{[\![h(\mathbf{x}_n) \neq f(\mathbf{x}_n)]\!]}_{=1} \mathbb{P}\Big\{h(\mathbf{x}_n) \neq f(\mathbf{x}_n)\Big\} \\ &\quad + \underbrace{[\![h(\mathbf{x}_n) = f(\mathbf{x}_n)]\!]}_{=0} \Big(1 - \mathbb{P}\Big\{h(\mathbf{x}_n) \neq f(\mathbf{x}_n)\Big\}\Big) \\ &= \mathbb{E}\Big\{[\![h(\mathbf{x}_n) \neq f(\mathbf{x}_n)]\!]\Big\}. \end{aligned} \tag{3}$$

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 9 shows how an in-sample error is computed.

What is "out-sample" error?

- Out-sample error is the testing error. It is *the* error that tells us how well our machine learning algorithm is doing.
- It is the statistical expectation of the error of the population set.
- Out-sample error cannot be computed from the training set. It can only be estimated, if the training and testing data are drawn from the same distribution.

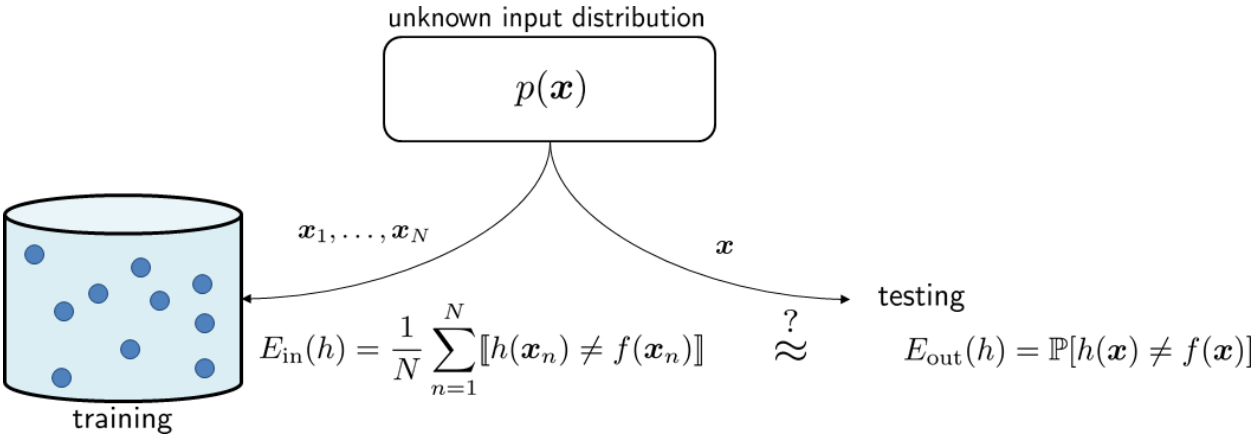


Figure 9: E_{in} is evaluated using the training data, whereas E_{out} is evaluated using the testing sample.

2.3 Basic probability inequality

We now need a mathematical tool to analyze $E_{\text{in}}(h)$ and $E_{\text{out}}(h)$. But to elaborate on our tools, we need to review a few basic results in probability inequalities.

Our first inequality is the **Markov's inequality**. It is an elementary inequality that links probability and expectation.

Theorem 1 (**Markov's inequality**). *Let $X \geq 0$ be a non-negative random variable. Then, for any $\varepsilon > 0$, we have*

$$\mathbb{P}[X \geq \varepsilon] \leq \frac{\mathbb{E}[X]}{\varepsilon}. \tag{4}$$

Markov's inequality concerns the **tail** of the random variable. As illustrated in Figure **??**, $\mathbb{P}[X \geq \varepsilon]$ measures the probability that the random variable takes a value greater than ε. Markov's inequality asserts that this probability $\mathbb{P}[X \geq \varepsilon]$ is upper-bounded by the ratio $\mathbb{E}[X]/\varepsilon$. This result is useful because it relates the probability and the expectation. In many problems the probability $\mathbb{P}[X \geq \varepsilon]$ could be difficult to evaluate if the PDF is complicated. The expectation, on the other hand, is usually easier to evaluate.

Proof. Consider $\varepsilon\mathbb{P}[X \geq \varepsilon]$. It follows that

$$\varepsilon\mathbb{P}[X \geq \varepsilon] = \int_{\varepsilon}^{\infty} \varepsilon\, f_X(x)\, dx \leq \int_{\varepsilon}^{\infty} x f_X(x)\, dx,$$

where the inequality is valid because for any $x \geq \varepsilon$ the integrand (which is non-negative) will always increase (or at least not decrease). It then follows that

$$\int_{\varepsilon}^{\infty} x f_X(x)\, dx \leq \int_{0}^{\infty} x f_X(x)\, dx = \mathbb{E}[X]. \qquad \square$$

How tight is Markov's inequality? It is possible to create a random variable such that the equality is met. However, in general, the estimate provided by the upper bound is not tight. Here is an example.

Example 1. Let $X \sim \text{Uniform}(0, 4)$. Verify Markov's inequality for $\mathbb{P}[X \geq 2]$, $\mathbb{P}[X \geq 3]$ and $\mathbb{P}[X \geq 4]$.

Solution. First, we observe that $\mathbb{E}[X] = 2$. Then

$$\begin{aligned} \mathbb{P}[X \geq 2] = 0.5, \qquad & \frac{\mathbb{E}[X]}{2} = 1, \\ \mathbb{P}[X \geq 3] = 0.25, \qquad & \frac{\mathbb{E}[X]}{3} = 0.67, \\ \mathbb{P}[X \geq 4] = 0, \qquad & \frac{\mathbb{E}[X]}{4} = 0.5. \end{aligned}$$

Therefore, although the upper bounds are all valid, they are very loose.

The next inequality is a simple extension of Markov's inequality. The result is known as Chebyshev's inequality. Chebyshev's inequality states that this tail probability is upper-bounded by $\text{Var}[X]/\varepsilon^2$.

Theorem 2 (**Chebyshev's inequality**). *Let X be a random variable with mean μ. Then for any $\varepsilon > 0$ we have*

$$\mathbb{P}[|X - \mu| \geq \varepsilon] \leq \frac{\text{Var}[X]}{\varepsilon^2}. \tag{5}$$

Proof. We apply Markov's inequality to show that

$$\mathbb{P}[|X - \mu| \geq \varepsilon] = \mathbb{P}[(X - \mu)^2 \geq \varepsilon^2] \leq \frac{\mathbb{E}[(X - \mu)^2]}{\varepsilon^2} = \frac{\text{Var}[X]}{\varepsilon^2}.$$

Example 2. Let $X \sim \text{Uniform}(0, 2\sqrt{2})$. Find the bound of Chebyshev's inequality for the probability $\mathbb{P}[|X - \mu| \geq 1]$.

Solution. Note that $\mathbb{E}[X] = 2$ and $\sigma^2 = (2\sqrt{2})^2/12 = 2/3$. Therefore, we have

$$\mathbb{P}[|X - \mu| \geq 1] \leq \frac{\sigma^2}{\varepsilon^2} = \frac{2}{3},$$

which is a valid upper bound, but not very useful.

Example 3. Let $X \sim \text{Exponential}(1)$. Find the bound of Chebyshev's inequality for the probability $\mathbb{P}[X \geq \varepsilon]$.

Solution. Note that $\mathbb{E}[X] = 1$ and $\sigma^2 = 1$. Thus we have

$$\mathbb{P}[X \geq \varepsilon] = \mathbb{P}[X - \mu \geq \varepsilon - \mu] \leq \mathbb{P}[|X - \mu| \geq \varepsilon - \mu] \leq \frac{\sigma^2}{(\varepsilon - \mu)^2} = \frac{1}{(\varepsilon - 1)^2}.$$

We can compare this with the exact probability, which is

$$\mathbb{P}[X \geq \varepsilon] = 1 - F_X(\varepsilon) = e^{-\varepsilon}.$$

Again, the estimate given by Chebyshev's inequality is acceptable but too conservative.

Corollary 1. *Let $X_1, \ldots, X_N$ be i.i.d. random variables with mean $\mathbb{E}[X_n] = \mu$ and variance $\text{Var}[X_n] = \sigma^2$. Let $\overline{X}_N = \frac{1}{N}\sum_{n=1}^{N} X_n$ be the sample mean. Then*

$$\mathbb{P}\left[\left|\overline{X}_N - \mu\right| > \epsilon\right] \leq \frac{\sigma^2}{N\epsilon^2}. \tag{6}$$

Proof. We can first show that $\mathbb{E}[\overline{X}_N] = \mu$ and $\text{Var}[\overline{X}_N]$ satisfies

$$\text{Var}[\overline{X}_N] = \frac{1}{N^2}\sum_{n=1}^{N}\text{Var}[X_n] = \frac{\sigma^2}{N}.$$

Then by Chebyshev's inequality,

$$\mathbb{P}\left[\left|\overline{X}_N - \mu\right| > \epsilon\right] \leq \frac{\text{Var}[\overline{X}_N]}{\epsilon^2} = \frac{\sigma^2}{N\epsilon^2}.$$

□

The consequence of this corollary is that the upper bound $\sigma^2 N/\epsilon^2$ will converge to zero as $N \to \infty$. Therefore, the probability of getting the event $\{\left|\overline{X}_N - \mu\right| > \epsilon\}$ is vanishing. It means that the sample average $\overline{X}_N$ is converging to the true population mean μ, in the sense that the probability of failing is shrinking.

2.4 Chernoff bound

We now introduce a powerful inequality or a set of general procedures that gives us some highly useful inequalities. The idea is named for Herman Chernoff, although it was actually due to his colleague Herman Rubin.

Theorem 3 (**Chernoff's bound**). *Let X be a random variable. Then, for any $\varepsilon \geq 0$, we have that*

$$\mathbb{P}[X \geq \varepsilon] \leq e^{-\varphi(\varepsilon)}, \tag{7}$$

where[a]

$$\varphi(\varepsilon) = \max_{s>0}\left\{s\varepsilon - \log M_X(s)\right\}, \tag{8}$$

and $M_X(s)$ is the moment-generating function.

[a] $\varphi(\varepsilon)$ is called the Fenchel-Legendre dual function of $\log M_X$. See references [6-14].

Proof. There are two tricks in the proof of Chernoff's bound. The first trick is a nonlinear transformation. Since e^{sx} is an increasing function for any $s > 0$ and x, we have that

$$\begin{aligned}
\mathbb{P}[X \geq \varepsilon] &= \mathbb{P}[e^{sX} \geq e^{s\varepsilon}] \\
&\overset{(a)}{\leq} \frac{\mathbb{E}[e^{sX}]}{e^{s\varepsilon}} \\
&\overset{(b)}{=} e^{-s\varepsilon}M_X(s) \\
&= e^{-s\varepsilon+\log M_X(s)},
\end{aligned}$$

where the inequality (a) is due to Markov's inequality. Step (b) just uses the definition of MGF that $\mathbb{E}[e^{sX}] = M_X(s)$.

Now for the second trick. Note that the above result holds for all s. That means it must also hold for the s that minimizes $e^{-s\varepsilon+\log M_X(s)}$. This implies that

$$\mathbb{P}[X \geq \varepsilon] \leq \min_{s>0}\left\{e^{-s\varepsilon+\log M_X(s)}\right\}.$$

Again, since e^x is increasing, the minimizer of the above probability is also the maximizer of this function:

$$\varphi(\varepsilon) = \max_{s>0} \left\{ s\varepsilon - \log M_X(s) \right\}.$$

Thus, we conclude that $\mathbb{P}[X \geq \varepsilon] \leq e^{-\varphi(\varepsilon)}$.

□

Let's consider an example of how Chernoff's bound can be useful. Suppose that we have a random variable $X \sim \text{Gaussian}(0, \sigma^2/N)$. The number N can be regarded as the number of samples. For example, if $Y_1, \ldots, Y_N$ are N Gaussian random variables with mean 0 and variance σ^2, then the average $X = \frac{1}{N}\sum_{n=1}^{N} Y_n$ will have mean 0 and variance σ^2/N. Therefore, as N grows, the variance of X will become smaller and smaller.

First, since the random variable is Gaussian, we can show the following:

Lemma 1. *Let $X \sim$ Gaussian$(0, \frac{\sigma^2}{N})$ be a Gaussian random variable. Then, for any $\varepsilon > 0$,*

$$\mathbb{P}[X \geq \varepsilon] = 1 - \Phi\left(\frac{\sqrt{N}\varepsilon}{\sigma}\right), \tag{9}$$

where Φ is the standard Gaussian's CDF.

Note that this is the **exact** result: If you tell me ε, N, and σ, then the probability $\mathbb{P}[X \geq \varepsilon]$ is exactly the one shown on the right-hand side. No approximation, no randomness.

Proof. Since X is Gaussian, the probability is

$$\begin{aligned}
\mathbb{P}[X \geq \varepsilon] &= \int_{\varepsilon}^{\infty} \frac{1}{\sqrt{2\pi(\sigma^2/N)}} \exp\left\{-\frac{x^2}{2(\sigma^2/N)}\right\} dx \\
&= 1 - \int_{-\infty}^{\varepsilon} \frac{1}{\sqrt{2\pi(\sigma^2/N)}} \exp\left\{-\frac{x^2}{2(\sigma^2/N)}\right\} dx \\
&= 1 - \int_{-\infty}^{\frac{\varepsilon}{\sqrt{\sigma^2/N}}} \frac{1}{\sqrt{2\pi}} \exp\left\{-\frac{x^2}{2}\right\} dx \\
&= 1 - \Phi\left(\frac{\varepsilon}{\sqrt{\sigma^2/N}}\right) = 1 - \Phi\left(\frac{\sqrt{N}\varepsilon}{\sigma}\right).
\end{aligned}$$

□

Let us compute the bound given by Chebyshev's inequality.

Lemma 2. *Let $X \sim$ Gaussian$(0, \frac{\sigma^2}{N})$ be a Gaussian random variable. Then, for any $\varepsilon > 0$, Chebyshev's inequality implies that*

$$\mathbb{P}[X \geq \varepsilon] \leq \frac{\sigma^2}{N\varepsilon^2}. \tag{10}$$

Proof. We apply Chebyshev's inequality by assuming that $\mu = 0$:

$$\begin{aligned}
\mathbb{P}[X \geq \varepsilon] = \mathbb{P}[X - \mu \geq \varepsilon - \mu] &\leq \mathbb{P}[|X - \mu| \geq \varepsilon - \mu] \\
&\leq \frac{\mathbb{E}[(X-\mu)^2]}{(\varepsilon - \mu)^2} = \frac{\sigma^2}{N\varepsilon^2}.
\end{aligned}$$

□

We now compute Chernoff's bound.

Theorem 4. *Let $X \sim$ Gaussian$(0, \frac{\sigma^2}{N})$ be a Gaussian random variable. Then, for any $\varepsilon > 0$, Chernoff's bound implies that*

$$\mathbb{P}[X \geq \varepsilon] \leq \exp\left\{-\frac{\varepsilon^2 N}{2\sigma^2}\right\}. \tag{11}$$

Proof. The MGF of a zero-mean Gaussian random variable with variance σ^2/N is $M_X(s) = \exp\left\{\frac{\sigma^2 s^2}{2N}\right\}$. Therefore, the function φ can be written as

$$\begin{aligned}\varphi(\varepsilon) &= \max_{s>0}\left\{s\varepsilon - \log M_X(s)\right\} \\ &= \max_{s>0}\left\{s\varepsilon - \frac{\sigma^2 s^2}{2N}\right\}.\end{aligned}$$

To maximize the function we take the derivative and set it to zero. This yields

$$\frac{d}{ds}\left\{s\varepsilon - \frac{\sigma^2 s^2}{2N}\right\} = 0 \quad \Rightarrow \quad s^* = \frac{N\varepsilon}{\sigma^2}.$$

Note that this s^* is a maximizer because $s\varepsilon - \frac{\sigma^2 s^2}{2N}$ is a concave function.

Substituting s^* into $\varphi(\varepsilon)$,

$$\begin{aligned}\varphi(\varepsilon) &= \max_{s>0}\left\{\frac{s\varepsilon - \sigma^2 s^2}{2N}\right\} \\ &= s^*\varepsilon - \frac{\sigma^2 (s^*)^2}{2N} = \left(\frac{N\varepsilon}{\sigma^2}\right)\varepsilon - \frac{\sigma^2}{2N}\left(\frac{N\varepsilon}{\sigma^2}\right)^2 = \frac{\varepsilon^2 N}{2\sigma^2},\end{aligned}$$

and hence

$$\mathbb{P}[X \geq \varepsilon] \leq e^{-\varphi(\varepsilon)} = \exp\left\{-\frac{\varepsilon^2 N}{2\sigma^2}\right\}.$$

□

2.5 Hoeffding inequality

Chernoff's bound can be used to derive many powerful inequalities. Here we present an inequality for bounded random variables. This result is known as Hoeffding's inequality.

Theorem 5 (**Hoeffding's inequality**). *Let $X_1, \ldots, X_N$ be i.i.d. random variables with $0 \leq X_n \leq 1$, and $\mathbb{E}[X_n] = \mu$. Then*

$$\mathbb{P}\left[\left|\overline{X}_N - \mu\right| > \epsilon\right] \leq 2e^{-2\epsilon^2 N}, \tag{12}$$

where $\overline{X}_N = \frac{1}{N}\sum_{n=1}^N X_n$.

You may skip the proof of Hoeffding's inequality.

Proof. (Hoeffding's inequality) First, we show that

$$\begin{aligned}\mathbb{P}\left[\overline{X}_N - \mu > \epsilon\right] &= \mathbb{P}\left[\frac{1}{N}\sum_{n=1}^N X_n - \mu > \epsilon\right] = \mathbb{P}\left[\sum_{n=1}^N (X_n - \mu) > N\epsilon\right] \\ &= \mathbb{P}\left[e^{s\sum_{n=1}^N (X_n-\mu)} \geq e^{s\epsilon N}\right] \\ &\leq \frac{\mathbb{E}[e^{s\sum_{n=1}^N (X_n-\mu)}]}{e^{s\epsilon N}} = \left(\frac{\mathbb{E}[e^{s(X_n-\mu)}]}{e^{s\epsilon}}\right)^N.\end{aligned}$$

Let $Z_n = X_n - \mu$. Then $-\mu \leq Z_n \leq 1-\mu$. At this point we use Hoeffding Lemma (see below) that $\mathbb{E}[e^{sZ_n}] \leq e^{\frac{s^2}{8}}$ because $b-a = (1-\mu)-(-\mu) = 1$. Thus,

$$\mathbb{P}\left[\overline{X}_N - \mu > \epsilon\right] \leq \left(\frac{\mathbb{E}[e^{sZ_n}]}{e^{s\epsilon}}\right)^N \leq \left(\frac{e^{\frac{s^2}{8}}}{e^{s\epsilon}}\right)^N = e^{\frac{s^2 N}{8} - s\epsilon N}, \qquad \forall s.$$

This result holds for all s, and thus it holds for the s that minimizes the right-hand side. This implies that

$$\mathbb{P}\left[\overline{X}_N - \mu > \epsilon\right] \leq \min_s \left\{\exp\left\{\frac{s^2 N}{8} - s\epsilon N\right\}\right\}.$$

Minimizing the exponent gives $\frac{d}{ds}\left\{\frac{s^2 N}{8} - s\epsilon N\right\} = \frac{sN}{4} - \epsilon N = 0$. Thus we have $s = 4\epsilon$. Hence,

$$\mathbb{P}\left[\overline{X}_N - \mu > \epsilon\right] \leq \exp\left\{\frac{(4\epsilon)^2 N}{8} - (4\epsilon)\epsilon N\right\} = e^{-2\epsilon^2 N}.$$

By symmetry, $\mathbb{P}\left[\overline{X}_N - \mu < -\epsilon\right] \leq e^{-2\epsilon^2 N}$. Then by union bound we show that

$$\begin{aligned}\mathbb{P}\left[|\overline{X}_N - \mu| > \epsilon\right] &= \mathbb{P}\left[\overline{X}_N - \mu > \epsilon\right] + \mathbb{P}\left[\overline{X}_N - \mu < -\epsilon\right] \\ &\leq e^{-2\epsilon^2 N} + e^{-2\epsilon^2 N} \\ &= 2e^{-2\epsilon^2 N}. \qquad \square\end{aligned}$$

Lemma 3 (**Hoeffding's lemma**). *Let $a \leq X \leq b$ be a random variable with $\mathbb{E}[X] = 0$. Then*

$$M_X(s) \stackrel{def}{=} \mathbb{E}\left[e^{sX}\right] \leq \exp\left\{\frac{s^2(b-a)^2}{8}\right\}. \tag{13}$$

Proof. Since $a \leq X \leq b$, we can write X as a linear combination of a and b:

$$X = \lambda b + (1-\lambda)a,$$

where $\lambda = \frac{X-a}{b-a}$. Since $\exp(\cdot)$ is a convex function, it follows that $e^{\lambda b + (1-\lambda)a} \leq \lambda e^b + (1-\lambda)e^a$. (Recall that h is convex if $h(\lambda x + (1-\lambda)y) \leq \lambda h(x) + (1-\lambda)h(y)$.) Therefore, we have

$$\begin{aligned}e^{sX} &\leq \lambda e^{sb} + (1-\lambda)e^{sa} \\ &= \frac{X-a}{b-a}e^{sb} + \frac{b-X}{b-a}e^{sa}.\end{aligned}$$

Taking expectations on both sides of the equation,

$$\mathbb{E}[e^{sX}] \leq \frac{-a}{b-a}e^{sb} + \frac{b}{b-a}e^{sa},$$

because $\mathbb{E}[X] = 0$. Now, if we let $\theta = -\frac{a}{b-a}$, then

$$\begin{aligned}\frac{-a}{b-a}e^{sb} + \frac{b}{b-a}e^{sa} &= \theta e^{sb} + (1-\theta)e^{sa} \\ &= e^{sa}\left(1 - \theta + \theta e^{s(b-a)}\right) = \left(1 - \theta + \theta e^{s(b-a)}\right)e^{-s\theta(b-a)} \\ &= \left(1 - \theta + \theta e^u\right)e^{-\theta u} = e^{-\theta u + \log(1-\theta+\theta e^u)},\end{aligned}$$

where we let $u = s(b-a)$. This can be simplified as $\mathbb{E}[e^{sX}] \leq \mathbb{E}[e^{\phi(u)}]$ by defining

$$\phi(u) = -\theta u + \log(1 - \theta + \theta e^u).$$

The final step is to approximate $\phi(u)$. To this end, we use Taylor approximation:

$$\phi(u) = \phi(0) + u\phi'(0) + \frac{u^2}{2}\phi''(\xi),$$

for some $\xi \in [a, b]$. Since $\phi(0) = 0$, $\phi'(0) = 0$, and $\phi''(u) \leq \frac{1}{4}$ for all u, it follows that

$$\phi(u) = \frac{u^2}{2}\phi''(\xi) \leq \frac{u^2}{8} = \frac{s^2(b-a)^2}{8}. \qquad \square$$

End of the proof.

What is so special about the Hoeffding's inequality?

- Since Hoeffding's inequality is derived from Chernoff's bound, it inherits the tightness. Hoeffding's inequality is much stronger than Chebyshev's inequality in bounding the tail distributions.
- Hoeffding's inequality is one of the few inequalities that do not require $\mathbb{E}[X]$ and $\text{Var}[X]$ on the right-hand side.
- A downside of the inequality is that boundedness is not always easy to satisfy. For example, if X_n is a Gaussian random variable, Hoeffding does not apply. There are more advanced inequalities for situations like these.

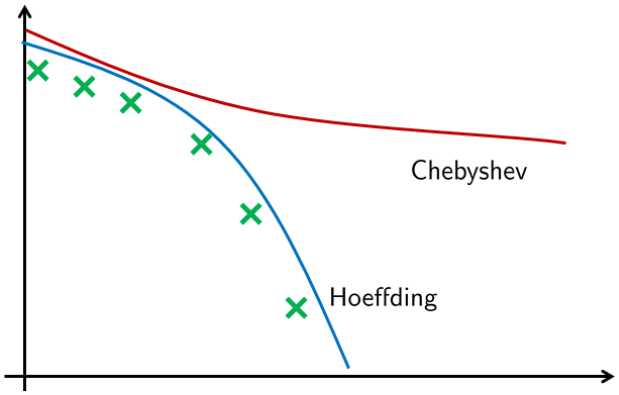


Figure 10: Comparing Hoeffding inequality and Chebyshev inequality to predict the actual probability bound.

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

$$\mathbb{P}\left[\left|\frac{1}{N}\sum_{n=1}^{N} X_n - \mu\right| > \epsilon\right] \leq \frac{\sigma^2}{\epsilon^2 N}. \tag{14}$$

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 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

$$\mathbb{P}\left[|E_{\text{in}}(h) - E_{\text{out}}(h)| > \epsilon\right] \leq 2e^{-2\epsilon^2 N}. \tag{15}$$

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.

2.6 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

$$\mathbb{P}\Big[\,|E_{\text{in}}(h) - E_{\text{out}}(h)| > \epsilon\Big] \leq 2e^{-2\epsilon^2 N} \tag{16}$$

  as a measure to quantify the error.

- **Approximately**: The in-sample error is an approximation of the out-sample error, as given by

$$\mathbb{P}\left[|E_{\text{in}}(h) - E_{\text{out}}(h)| > \epsilon\right] \leq 2e^{-2\epsilon^2 N}. \tag{17}$$

  The approximation error is controlled by ϵ.

- **Correct**: The error is bounded by the right hand side of the Hoeffding inequality:

$$\mathbb{P}\left[|E_{\text{in}}(h) - E_{\text{out}}(h)| > \epsilon\right] \leq 2e^{-2\epsilon^2 N}. \tag{18}$$

  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

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

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 6. *Consider a learning problem where we have a dataset $\mathcal{D}=\{\mathbf{x}_1,\ldots,\mathbf{x}_N\}$, 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 $\epsilon>0$,*

$$\mathbb{P}\Big\{\,|E_{\text{in}}(g)-E_{\text{out}}(g)|>\epsilon\Big\} \leq 2Me^{-2\epsilon^2 N}. \tag{19}$$

Significance of the Theorem:

Message 1: You can bound $E_{\text{out}}(h)$ using $E_{\text{in}}(h)$.

- $E_{\text{in}}(h)$: You know. $E_{\text{out}}(h)$: You don't know, but you want to know.
- They are close if N is large.

Message 2: The right hand side is independent of h and $p(\mathbf{x})$

- So it is a universal upper bound
- Works for any $\mathcal{A}$, any $\mathcal{H}$, any f, and any $p(\mathbf{x})$

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 $p_{\mathbf{X}}(\mathbf{x})$ which generates both the in-samples and the out-samples. Thus, whatever $p_{\mathbf{X}}$ 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:

$$E_{\text{out}}(g) \quad \underset{\text{Hoeffding Inequality}}{\approx} \quad E_{\text{in}}(g) \quad \underset{\text{Training Error}}{\approx} \quad 0 \tag{20}$$

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 (20) 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.

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**.

3.1 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 as

$$\underbrace{\mathbb{P}\Big\{ |E_{\text{in}}(g) - E_{\text{out}}(g)| > \epsilon \Big\}}_{\mathbb{P}[\mathcal{B}]} \leq \underbrace{2Me^{-2\epsilon^2 N}}_{\delta}.$$

for some event $\mathcal{B}$ (the $\mathcal{B}$ad event). This is equivalent to say that $\mathcal{B}$ happens with a tiny probability δ.

Let's rewrite equation further by rewriting

$$\mathbb{P}\Big\{ |E_{\text{in}}(g) - E_{\text{out}}(g)| > \epsilon \Big\} \leq \delta$$

as the following:

$$\mathbb{P}\Big\{ |E_{\text{in}}(g) - E_{\text{out}}(g)| \leq \epsilon \Big\} \geq 1 - \delta.$$

This can be read as

$$\text{with probability } 1-\delta, \qquad E_{\text{in}}(g) - \epsilon \leq E_{\text{out}}(g) \leq E_{\text{in}}(g) + \epsilon.$$

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

$$\delta = 2Me^{-2\epsilon^2 N} \quad \Longleftrightarrow \quad \epsilon = \sqrt{\frac{1}{2N}\log\frac{2M}{\delta}}. \tag{21}$$

Therefore, we arrive at the following inequality.

Theorem 7. *Consider a learning problem where we have a dataset $\mathcal{D} = \{\mathbf{x}_1, \ldots, \mathbf{x}_N\}$, 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 $1-\delta$,*

$$E_{\text{in}}(g) - \sqrt{\frac{1}{2N}\log\frac{2M}{\delta}} \leq E_{\text{out}}(g) \leq E_{\text{in}}(g) + \sqrt{\frac{1}{2N}\log\frac{2M}{\delta}}. \tag{22}$$

The inequality given by (22) 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 $E_{\text{in}}(g) - \epsilon$.

3.2 Dichotomy

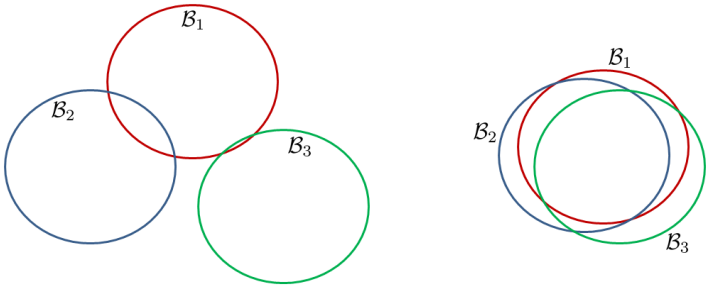
To make sense of the generalization bound, we need to ensure that $\epsilon \to 0$ as $N \to \infty$. In doing so, we need to assume that M does not grow exponentially fast, for otherwise term $\log 2M$ will cancel out the effect of $1/N$. However, if $\mathcal{H}$ is an infinite set, then M is unavoidably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

3.2.1 The problem of M

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.

3.2.2 Intuition

Consider the following situation. We have two hypothesis functions h_1 and h_2. They are not very different. If we go from h_1 to h_2, two things will happen:

- There is a change of the out-sample error:

$$\Delta E_{\text{out}} = \text{change in the +1 and -1 area.}$$

  Since h_1 is close to h_2, the change ΔE_{out} is very small.

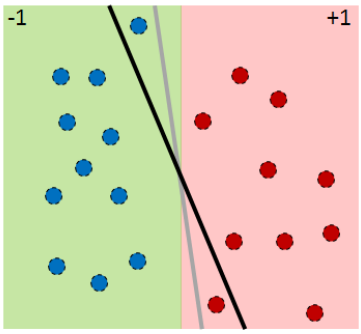
- There is a change of the in-sample error:

$$\Delta E_{\text{in}} = \text{change in labels of the training samples.}$$

We expect that ΔE_{in} also does not change much.

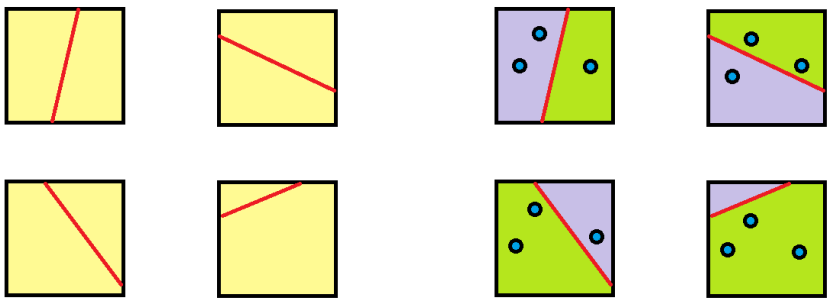
So we should expect the probabilities

$$\mathbb{P}[|E_{\text{in}}(h_1) - E_{\text{out}}(h_1)| > \epsilon] \approx \mathbb{P}[|E_{\text{in}}(h_2) - E_{\text{out}}(h_2)| > \epsilon].$$



In other words, if we are inspecting two hypothesis functions, their "similarity" or "difference" can be quantified. Even if we have an infinite set of hypothesis functions, we can "group" hypothesis functions together. The question is how.

The idea to resolve the "M problem" is to look at the input space. Consider the class of linear classifiers. For a 2D problem, the decision boundaries are shown on the left hand side of the figure below. Remember that there are infinitely many of these hypothesis functions. As we assign the labels to the half spaces, we can see that the hypothesis functions have partitioned the input space into two half-spaces.



The magic happens when we consider the training samples. For a set of fixed training samples, different hypothesis functions will be able to group the training samples. For example, we can have two on the left and one on the right, or two on the top and one at the bottom (See Figure above). But we notice one thing: if two hypothesis functions are very close, then their effect to the training samples is none. The effect will be here only when one training sample flips it label. Therefore, if we count the number of **unique** hypothesis functions, we can reduce the effective number of hypothesis functions.

3.2.3 Definition

Consider a dataset containing N data points $\mathbf{x}_1, \ldots, \mathbf{x}_N$. 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 $(\mathbf{x}_1, \ldots, \mathbf{x}_N)$, we will get a N-tuple $(h(\mathbf{x}_1), \ldots, h(\mathbf{x}_N))$ 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 $\mathcal{H}(\mathbf{x}_1, \ldots, \mathbf{x}_N)$. For example, if $\mathcal{H}$ contains two hypotheses h_α

and h_β such that h_α turns all training samples $\mathbf{x}_n$ to $+1$ and h_β turns all training samples $\mathbf{x}_n$ to -1, then we have two dichotomies and $\mathcal{H}(\mathbf{x}_1,\ldots,\mathbf{x}_N)$ is defined as

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

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

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

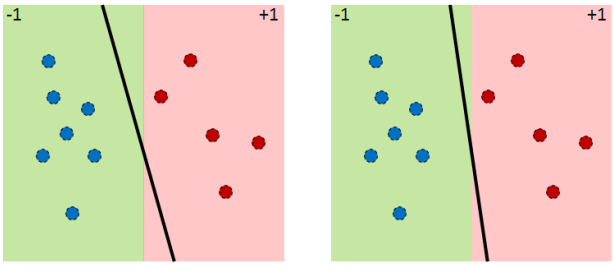
$$\mathcal{H}(\mathbf{x}_1,\ldots,\mathbf{x}_N) = \{(h(\mathbf{x}_1),\ldots,h(\mathbf{x}_N)) \mid h\in\mathcal{H}\}. \tag{23}$$

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

The table below shows the difference between a hypothesis and a dichotomy. In brief, there could be an infinite number of hypothesis functions because a hypothesis is defined by the entire input space. The set of dichotomies is usually much smaller because a dichotomy is defined by the training set. If we have finite number of training samples, the number of dichotomies will also be smaller.

Hypothesis	Dichotomy
$h:\mathcal{X}\to\{+1,-1\}$	$h:\{\mathbf{x}_1,\ldots,\mathbf{x}_N\}\to\{+1,-1\}$
for all population samples	for training samples only
number can be infinite	number is at most 2^N

Here is an example where the two hypothesis functions are different, but there is only one dichotomy.



3.3 The growth function

3.3.1 Counting the number of dichotomies

Suppose there are $N=3$ data points in $\mathcal{X}$ so that we have $\mathbf{x}_1,\mathbf{x}_2,\mathbf{x}_3$. 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 $\mathbf{x}_1,\mathbf{x}_2,\mathbf{x}_3$ are located at position A, B, C respectively, as illustrated in Figure 11. These locations are fixed, and the 3 data points $\mathbf{x}_1,\mathbf{x}_2,\mathbf{x}_3$ 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 11, 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 11. As we move around the decision boundary, we can construct at most 8 dichotomies for $\mathbf{x}_1,\mathbf{x}_2,\mathbf{x}_3$ located at A, B and C.

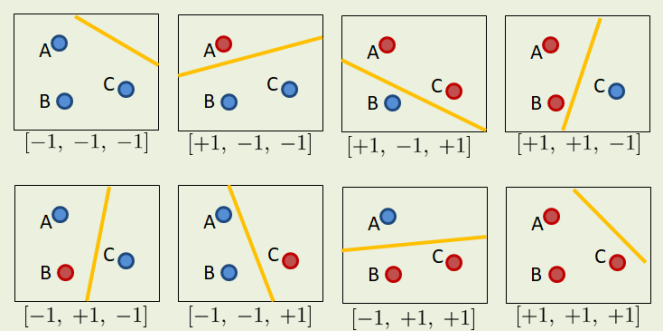


Figure 11: Example of number of dichotomies.

What if we move $\mathbf{x}_1, \mathbf{x}_2, \mathbf{x}_3$ to somewhere else, for example the locations specified by Figure 12? 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.

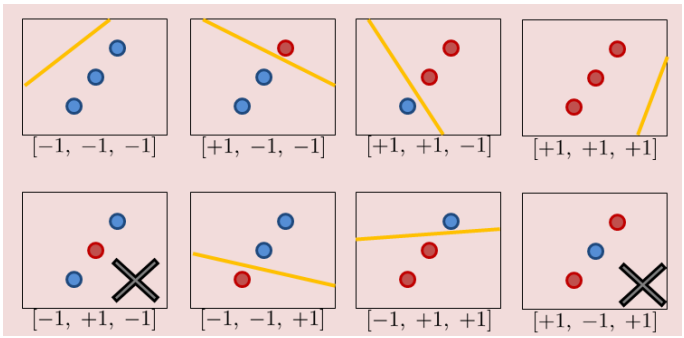


Figure 12: Example of number of dichotomies.

3.3.2 Growth function

Now we want to define a quantity that measures the number of dichotomies. This quantity should be universal for any configuration of $\mathbf{x}_1, \ldots, \mathbf{x}_N$, 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 $\mathbf{x}_1, \ldots, \mathbf{x}_N$, we realize that among all the possible configurations of $\mathbf{x}_1, \ldots, \mathbf{x}_N$, there exists one that can maximize the size of $\mathcal{H}(\mathbf{x}_1, \ldots, \mathbf{x}_N)$. Define this maximum as the **growth function**.

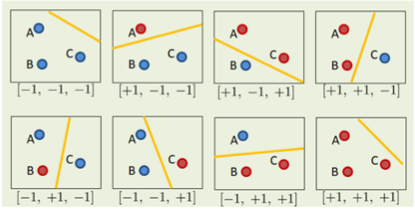
Definition 4. *The growth function for a hypothesis set $\mathcal{H}$ is the maximum number of dichotomies we can possibly assign by moving around the training samples. It is defined as*

$$m_{\mathcal{H}}(N) = \max_{\mathbf{x}_1, \ldots, \mathbf{x}_N \in \mathcal{X}} |\mathcal{H}(\mathbf{x}_1, \ldots, \mathbf{x}_N)|, \tag{24}$$

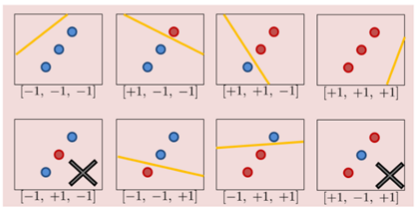
where $|\cdot|$ denotes the cardinality of a set.

Example 1. Consider three training samples in a 2D space. Suppose we choose a linear model as the classifier. The growth number, i.e, $m_{\mathcal{H}}(3)$, is 8. This is because if we configure $\mathbf{x}_1, \mathbf{x}_2, \mathbf{x}_3$ like the ones in the green part of Figure 11, we will get 8 dichotomies. Of course, if we land on the red case we will get 6 dichotomies only. However, since the definition of $m_{\mathcal{H}}(3)$ asks for the maximum, we conclude that the growth number is 8.

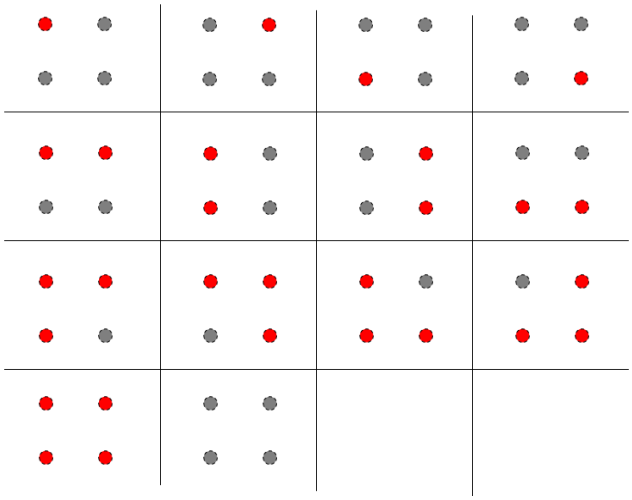
- $\mathcal{H}$ = linear models in 2D
- $N = 3$
- How many dichotomies can I generate by moving the three points?
- This gives you 8. Are we the best?



- $\mathcal{H}$ = linear models in 2D
- $N = 3$
- How many dichotomies can I generate by moving the three points?
- This gives you 6. The previous is the best. So $m_{\mathcal{H}}(3) = 8$.



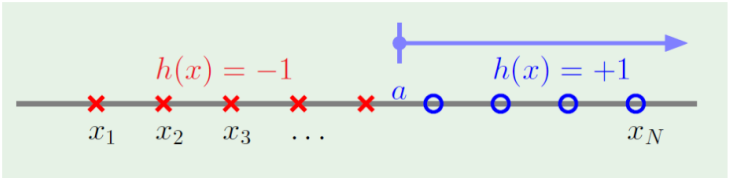
Example 2. How about $m_{\mathcal{H}}(N)$ when $N = 4$? It turns out that there are at most 14 dichotomies no matter where we put the four data points $\mathbf{x}_1, \mathbf{x}_2, \mathbf{x}_3, \mathbf{x}_4$.



Example 3. In this example we consider a set of data points sitting on a 1D line. The hypothesis set is an indicator function that has a cutoff point a such that

$$h(x) = \text{sign}(x - a).$$

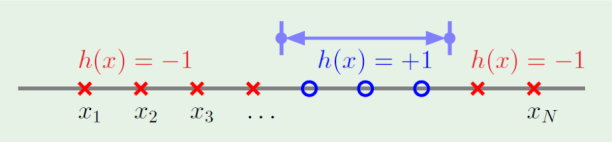
Thus, if a training sample lives on the right hand side of a, the label will be $+1$. Otherwise the label will be -1. The hypothesis function will cut the real line into two halves.



- $\mathcal{H}$ = set of h: $\mathbb{R} \to \{+1, -1\}$
- $h(x) = \text{sign}(x - a)$
- Cut the line into two halves
- You can only move along the line
- $m_{\mathcal{H}}(N) = N + 1$
- The N comes from the N points
- The $+1$ comes from the two ends

For this problem, the number of dichotomies is determined by the number of cuts we can possible make. If we have N points, then we have $N + 1$ cuts. So the maximum number of dichotomies is $N + 1$. This will give us the growth function.

Example 4. This example is similar to the previous example, with the exception that now the hypothesis does not cut the real line into two halves. It segments the real line with an interval.



- $\mathcal{H}$ = set of h: $\mathbb{R} \to \{+1, -1\}$
- Put an interval
- Length of the interval is N points
- $$m_{\mathcal{H}}(N) = \binom{N+1}{2} + 1 = \frac{N^2}{2} + \frac{N}{2} + 1$$
- Think of $N+1$ balls, pick 2.

To determine the number of dichotomies, we notice that an interval is specified by the start point and the end point. Since there are $N+1$ possible locations we can put a cut, and since we need to put two cuts, the number of dichotomies boils down to

$$m_{\mathcal{H}}(N) = \binom{N+1}{2}.$$

3.4 VC dimension

3.4.1 Shatter

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 $\mathcal{H}(\mathbf{x}_1, \ldots, \mathbf{x}_N)$ must be a subset in $\{+1, -1\}^N$, and hence there are at most 2^N dichotomies:

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

What is **shatter**?

If a hypothesis set $\mathcal{H}$ is able to generate all 2^N dichotomies, then we say that $\mathcal{H}$ **shatter** $\mathbf{x}_1, \ldots, \mathbf{x}_N$.

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 $m_{\mathcal{H}}(4) = 14 < 2^4$.

3.4.2 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 $m_{\mathcal{H}}(N) = 2^N$.*

The way to think about the VC dimension can be summarized in the bullet points below:

- You give me a hypothesis set $\mathcal{H}$, e.g., linear model
- You tell me the number of training samples N
- Start with a small N
- I will be able to shatter for a while, until I hit a bump
- E.g., linear in 2D: $N = 3$ is okay, but $N = 4$ is not okay
- So I find the **largest** N such that $\mathcal{H}$ can shatter N training samples
- E.g., linear in 2D: $d_{\text{VC}} = 3$
- If $\mathcal{H}$ is complex, then expect large d_{VC}
- Does not depend on $p(\boldsymbol{x})$, $\mathcal{A}$ and f

3.4.3 VC dimension of a Perceptron algorithm

To give readers a non-trivial example, we consider the VC dimension of a perceptron algorithm.

Theorem 8 (VC Dimension of Perceptron). *Consider the input space $\mathcal{X} = \mathbb{R}^d \cup \{1\}$. The VC dimension of the perceptron algorithm is*

$$d_{\text{VC}} = d + 1. \tag{25}$$

First of all, what is a perceptron algorithm? Perceptron algorithm is nothing but a **linear model** obtained through a special algorithm. So, it should inherent all the properties of a linear model. The dimension d, if you recall, is the number of parameters in the input vector $\mathbf{x}$. But since $\mathbf{x} = [x_1, \ \ldots, \ x_d, \ 1]^T$, the dimension of $\mathbf{x}$ is $d + 1$.

Intuition. The way to think about the VC dimension of the perceptron algorithm is to ask yourself: If we have $N = 3$ data points, can a linear model with an order $d = 2$ shatter the three data points. If $N = 4$, can we do it with $d = 3$? And so on.

Suppose that $d = 2$, so we are considering a 2D perceptron. Let's ask ourselves two cases:

- $N = 3$. 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.

- $N = 4$. 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 $d_{\text{VC}} = 3$.

The following proof can be skipped.

Proof of Theorem. 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 $\mathbf{x}_1, \ldots, \mathbf{x}_{d+1}$ by defining $\mathbf{x}_n = [1,\ 0, \ldots,\ 1,\ \ldots,\ 0]^T$, 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 $\mathbf{x}_n$), 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

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

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

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

If we can find $\mathbf{w}$, 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* $\mathbf{w}$. If we solve the above system of equations without the sign, and if we obtain a $\mathbf{w}$ that flips the sign of one of the rows, then the $\mathbf{w}$ we found is not able to shatter. But if the $\mathbf{w}$ 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.

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

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 $\mathbf{w}$ for any given $\mathbf{y}$, 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 $\mathbf{x}_1, \ldots, \mathbf{x}_{d+1}, \mathbf{x}_{d+2}$, we can write

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

Construct a dichotomy with labels

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

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

3.5 Bounding the growth function

3.5.1 Tightening the generalization bound

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

$$m_{\mathcal{H}}(N) \leq \sum_{i=0}^{d_{\text{VC}}} \binom{N}{i}. \tag{26}$$

We shall skip the proof which can be found in Theorem 2.4 of *Learning from Data*. 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

$$\epsilon = \sqrt{\frac{1}{2N} \log \frac{2m_{\mathcal{H}}(N)}{\delta}} \leq \sqrt{\frac{1}{2N} \log \frac{2(N^{d_{\text{VC}}} + 1)}{\delta}}. \tag{27}$$

3.5.2 Interpreting the VC dimension

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 $p_{\mathbf{X}}$ and independent of the target function f. It only depends on the hypothesis set $\mathcal{H}$ and the training examples $\mathbf{x}_1, \ldots, \mathbf{x}_N$.

- $d_{\text{VC}} = \infty$. 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 10 (Generalization Bound)**.** *For any tolerance $\delta > 0$*

$$E_{\text{out}}(g) \leq E_{\text{in}}(g) + \sqrt{\frac{8}{N} \log \frac{4m_{\mathcal{H}}(2N)}{\delta}}, \tag{28}$$

with probability at least $1-\delta$.

3.5.3 Slackness in the generalization bound

The VC generalization bound in (28) 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.

3.6 Sample and model complexity

The generalization bound helps us understand the sample and model complexity.

3.6.1 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 $10 \times d_{\text{VC}}$.

3.6.2 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 13. 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.

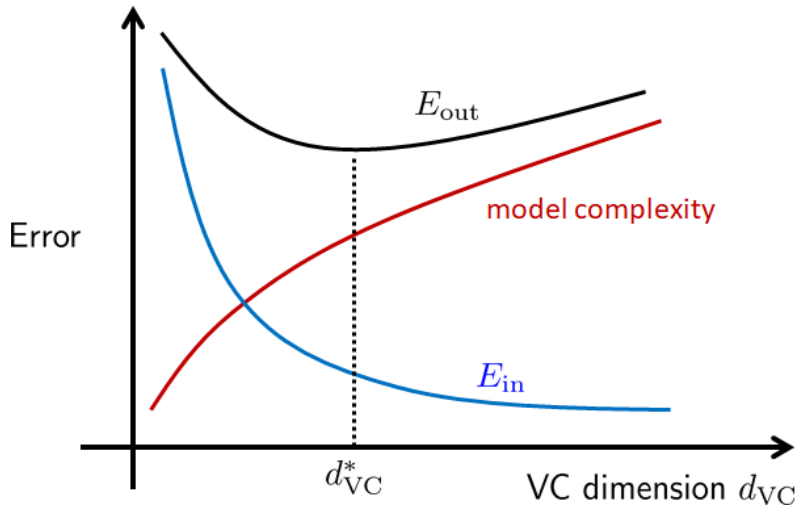


Figure 13: The VC generalization bound suggests a trade-off between model complexity and generalization. If we use a more complex model, the in-sample error drops but the out-sample error increases. The optimal model complexity is determined when the out-sample error is minimized.

3.6.3 Generalization bound for testing (not training)

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

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

contains L samples drawn from the distribution $p_{\mathbf{X}}(\mathbf{x})$. No testing data $\mathbf{x}_m$ 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 Bias and variance analysis

4.1 From VC analysis to bias-variance analysis

4.1.1 Rewriting the out-sample error

The bias-variance analysis is an alternative way of analyzing the out-sample error. In the VC analysis, the out-sample error is defined as

$$E_{\text{out}}(g) = \mathbb{P}[g(\mathbf{x}) \neq f(\mathbf{x})]. \tag{29}$$

Basically, it is the probability that the final hypothesis $g(\mathbf{x})$ is not the same as the target function $f(\mathbf{x})$, i.e., the *testing error*. The random variable $\mathbf{x}$ is a testing sample drawn from a distribution $p(\mathbf{x})$. By evaluating the probability over $\mathbf{x}$, we ask that with all the testing samples from the distribution $p(\mathbf{x})$, what is the probability of making a testing error.

The inequality "$g(\mathbf{x}) \neq f(\mathbf{x})$" implies that we are interested in the binary event. So if we let $B = \{g(\mathbf{x}) \neq f(\mathbf{x})\}$ be the event, then $B \in \{0, 1\}$ is a binary variable. Then, the out-sample error becomes

$$\begin{aligned} E_{\text{out}}(g) &= \mathbb{P}[g(\mathbf{x}) \neq f(\mathbf{x})] \\ &= \mathbb{P}[B = 1] \\ &= 1 \cdot \mathbb{P}[B = 1] + 0 \cdot \mathbb{P}[B = 0] \\ &= \mathbb{E}[B]. \end{aligned} \tag{30}$$

In other words, the out-sample error $E_{\text{out}}(g)$ (which should be the probability) can be written as an expectation

$$E_{\text{out}}(g) = \mathbb{E}_{\mathbf{x}}[\mathbf{1}\{g(\mathbf{x}) \neq f(\mathbf{x})\}], \tag{31}$$

where $\mathbf{1}$ is the indicator function. The expectation is taken over all $\mathbf{x} \sim p(\mathbf{x})$.

4.1.2 0-1 loss and square loss

The difference between the VC analysis the bias-variance analysis is to recognize that in the VC analysis, we have

$$E_{\text{out}}(g) = \mathbb{E}_{\mathbf{x}}\Big[\mathbf{1}\{g(\mathbf{x}) \neq f(\mathbf{x})\}\Big].$$

The expectation is the **0-1 loss**. In bias-variance analysis, we replace the 0-1 loss by the **square loss**:

$$E_{\text{out}}(g) = \mathbb{E}_{\mathbf{x}}\Big[(g(\mathbf{x}) - f(\mathbf{x}))^2\Big].$$

Since the square loss is differentiable, we can potentially say something that VC analysis is not able to say.

4.1.3 Dependency on training set

One thing we need to clarify is the explicit dependency of the training set $\mathcal{D}$ in the bias-variance analysis. In VC analysis, the training set is fixed and the final hypothesis is selected based on this particular training set. However, since Hoeffding inequality is **uniform** across all training sets, the conclusion of the VC analysis does not change regardless which $\mathcal{D}$ we use and which final hypothesis is picked. One way to think about the Hoeffding inequality is that it provides the *worst case* bound. Since it is the worst case bound, it does not matter which specific $\mathcal{D}$ and g we are using.

The bias-variance analysis is an average case analysis. Since the average case analysis requires taking the expectation, the meaningful way of defining the out-sample error is to consider

$$E_{\text{out}}(g^{(\mathcal{D})}) = \mathbb{E}_{\mathbf{x}}\Big[(g^{(\mathcal{D})}(\mathbf{x}) - f(\mathbf{x}))^2\Big]. \tag{32}$$

This means 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:

$$\mathbb{E}_{\mathcal{D}}\Big[\mathbb{E}_{\text{out}}(g^{(\mathcal{D})})\Big] = \mathbb{E}_{\mathcal{D}}\Big[\mathbb{E}_{\mathbf{x}}\Big[(g^{(\mathcal{D})}(\mathbf{x}) - f(\mathbf{x}))^2\Big]\Big]. \tag{33}$$

Therefore, while the VC analysis is the **worst case** analysis, the bias-variance is the **average case** analysis because we take average over all the possible training sets $\mathcal{D}$.

What is the difference between VC analysis and bias-variance analysis?

- VC analysis: 0-1 loss. Bias-variance: square loss.
- VC analysis: $\mathcal{D}$ does not matter. Bias-variance: expectation over $\mathcal{D}$.
- VC analysis: worst case. Bias-variance: average case.

4.2 Decomposition of the bias-variance

In this subsection we shall discuss how the bias-variance analysis is derived.

4.2.1 Decomposition

Let's start by considering the out-sample error. If we use the square loss of the out-sample error, and if we consider that the training set $\mathcal{D}$ is random, we can do the following calculation:

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

Here, we define

$$\overline{g}(\mathbf{x}) = \mathbb{E}_{\mathcal{D}}[g^{(\mathcal{D})}(\mathbf{x})].$$

We will discuss the meaning of $\overline{g}(\mathbf{x})$ later.

With additional calculations, we can show that

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

Based on this decomposition, we can define two terms:

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

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

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

where

$$\text{bias} = \mathbb{E}_{\mathbf{x}}[\text{bias}(\mathbf{x})] \tag{34}$$

is the average bias over the distribution $p(\mathbf{x})$, and

$$\text{var} = \mathbb{E}_{\mathbf{x}}[\text{var}(\mathbf{x})] \tag{35}$$

is the average variance over $p(\mathbf{x})$.

4.2.2 Analysis of the bias

Let's take a closer look at the average hypothesis:

$$\overline{g}(\mathbf{x}) = \mathbb{E}_{\mathcal{D}}[g^{(\mathcal{D})}(\mathbf{x})]. \tag{36}$$

The average hypothesis can be considered as the asymptotic limit of the estimate

$$\overline{g}(\mathbf{x}) \approx \frac{1}{K}\sum_{k=1}^{K} g_k(\mathbf{x}), \tag{37}$$

where each $g_k(\mathbf{x})$ denotes the final hypothesis picked by considering the training set $\mathcal{D}_k$. For each $\mathcal{D}_k$, we will pick a different g_k. As shown in Figure 14, this will give us a whole collection of final hypothesis functions $g_1, \ldots, g_K$. Therefore, for any fixed $\mathbf{x}$, $g_k(\mathbf{x})$ is a random variable over the training set $\mathcal{D}_k$.

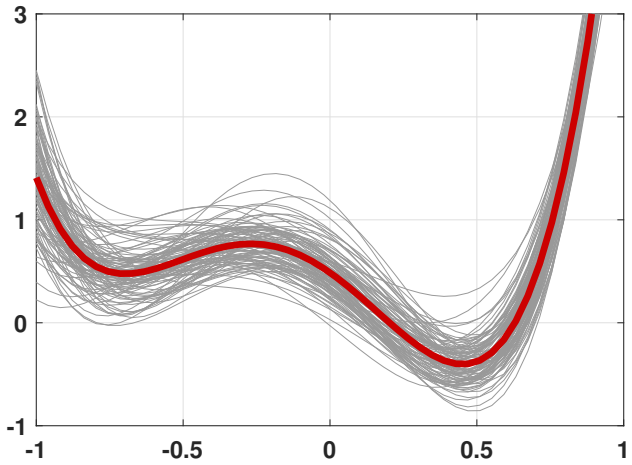


Figure 14: We run linear regression many times for different training datasets. Each one consists of different random realizations of noise. The gray curves are the regression lines returned by each of the training datasets. We then take the average of these gray curves to obtain the red curve, which is the average predictor.

Remark: 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.

```
import numpy as np
import matplotlib.pyplot as plt
from scipy.special import eval_legendre
np.set_printoptions(precision=2, suppress=True)

N = 20
x = np.linspace(-1,1,N)
a = np.array([0.5, -2, -3, 4, 6])
yhat = np.zeros((50,100))
for i in range(100):
  y = a[0] + a[1]*x    + a[2]*x**2 + \
             a[3]*x**3 + a[4]*x**4 + 0.5*np.random.randn(N)
  X = np.column_stack((np.ones(N), x, x**2, x**3, x**4))

  theta = np.linalg.lstsq(X, y, rcond=None)[0]
  t = np.linspace(-1,1,50)
  Xhat = np.column_stack((np.ones(50), t, t**2, t**3, t**4))
```



```
  yhat[:,i] = np.dot(Xhat, theta)
  plt.plot(t, yhat[:,i], c='gray')
plt.plot(t, np.mean(yhat, axis=1), c='r', linewidth=4)
```

What is bias?

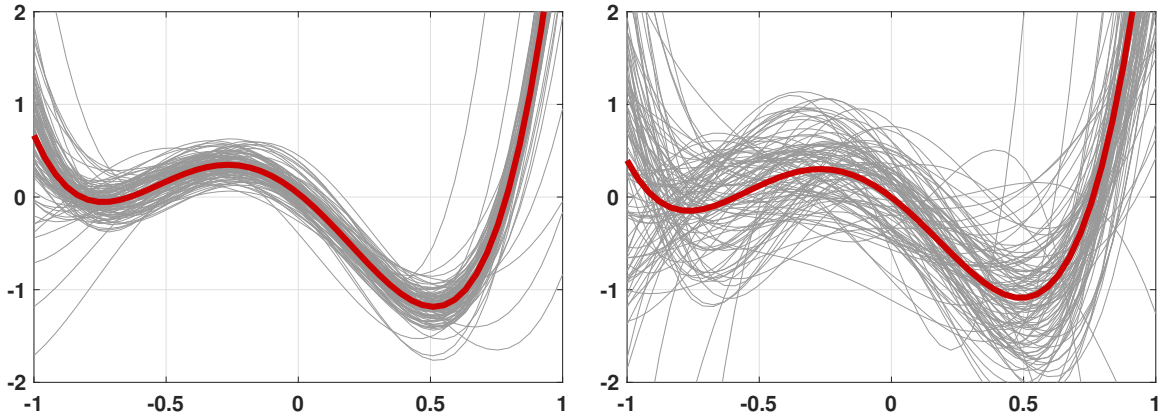
- Bias is defined as bias $= \mathbb{E}_{\mathbf{x}}[(\overline{g}(\mathbf{x}) - f(\mathbf{x}))^2]$, where $\mathbf{x}$ is a testing sample.
- It is the deviation from the average predictor to the true predictor.
- Bias is not necessarily a bad thing. A good predictor can have some bias as long as it helps to reduce the variance.

4.2.3 Analysis of variance

The other quantity in the game is the **variance**. Variance at a testing sample $\mathbf{x}$ is defined as

$$\text{var}(\mathbf{x}) \stackrel{\text{def}}{=} \mathbb{E}_{\mathcal{D}}[(g^{(\mathcal{D})}(\mathbf{x}) - \overline{g}(\mathbf{x}))^2]. \tag{38}$$

As the equation suggests, the variance measures the fluctuation between the predictor $g^{(\mathcal{D})}$ and the average predictor $\overline{g}$. Figure 15 illustrates the polynomial-fitting problem we discussed above. In this figure we consider two levels of variance by varying the noise strength of e_n. The figure shows that as the observation becomes noisier, the predictor $g^{(\mathcal{D})}$ will have a larger fluctuation for the average predictor.



(a) small variance (b) large variance

Figure 15: Variance measures the magnitude of fluctuation between the particular predictor $g^{(\mathcal{D})}$ and the average predictor $\overline{g}$.

What is variance?

- Variance is the deviation between the predictor $g^{(\mathcal{D})}$ and its average $\overline{g}$.
- It can be reduced by using more training samples.

4.2.4 Interpreting the bias-variance decomposition

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 bias ≈ 0. The variance, however, is large because there are many training sets under consideration.

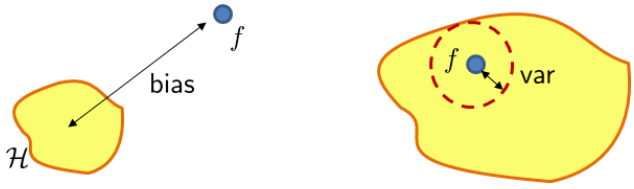
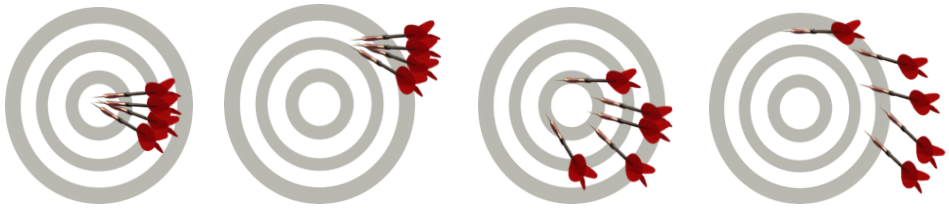


Figure 16: [Left] Large bias but small variance. [Right] Small bias but large variance.

Figure 17 gives a pictorial representation of bias and variance. In this figure, we construct four scenarios of bias and variance. Each cross represents the predictor $g^{(\mathcal{D})}$, with the true predictor f at the origin. Figure 17(a) shows the case with a low bias and a low variance. All these predictors $g^{(\mathcal{D})}$ are very close to the ground truth, and they have small fluctuations around their average. Figure 17(b) shows the case of a high bias and a low variance. It has a high bias because the entire group of $g^{(\mathcal{D})}$ is shifted to the corner. The bias, which is the distance from the truth to the average, is therefore large. The variance remains small because the fluctuation around the average is small. Figure 17(c) shows the case of a low bias but high variance. In this case, the fluctuation around the average is large. Figure 17 shows the case of high bias and high variance. We want to avoid this case.



Bias low
Var low
(a)

Bias high
Var low
(b)

Bias low
Var high
(c)

Bias high
Var high
(d)

Figure 17: Imagine that you are throwing a dart with a target at the center. The four subfigures show the levels of bias and variance.

How to interpret the bias and variance?

- Bias = How close is the **average function** $\overline{g}$ to the target f.
- Variance = How much **uncertainty** you have around $\overline{g}$.

4.3 Overfitting

The bias and variance analysis allows us to understand the phenomenon of overfitting. In this subsection, our goal is to illustrate the concept through several examples.

4.3.1 Fitting a sinusoid

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 18 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.

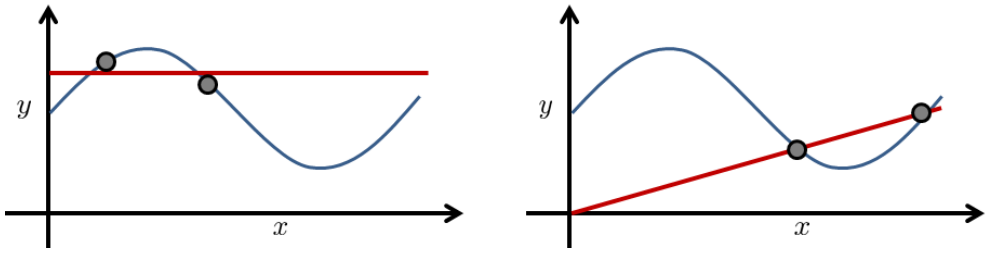


Figure 18: [Left] Fitting two data points using $\mathcal{M}_0$. [Right] Fitting two data points using $\mathcal{M}_1$.

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.

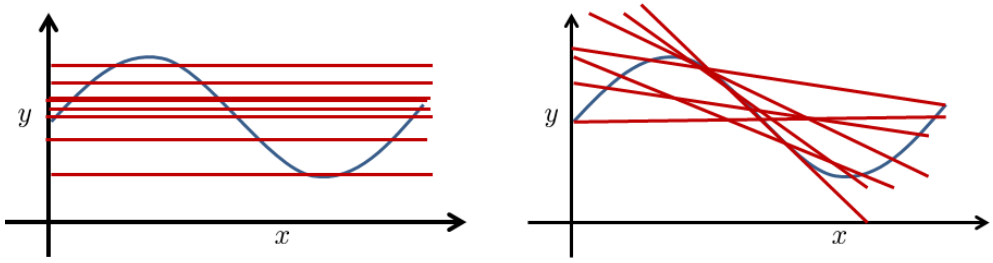


Figure 19: [Left] Possible lines generated by $\mathcal{M}_0$. [Right] Possible lines generated by $\mathcal{M}_1$.

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 20 we draw the possible lines that are within one standard deviation from the mean function, i.e., $\overline{g} \pm \sqrt{\text{var}(x)}$.

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!

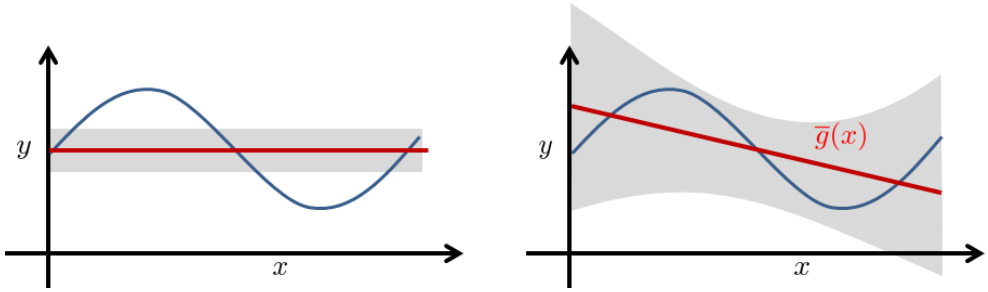


Figure 20: Average hypothesis function $\overline{g}(x)$ and the variance $\text{var}(x)$.

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.

4.3.2 Fitting a polynomial

Suppose that we are given a target function f. We do not know what it is because it is a target function. For visualization we draw the target funtion in red. We draw 5 **noisy** samples from this target function.

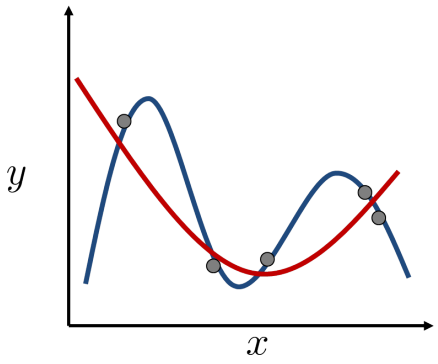
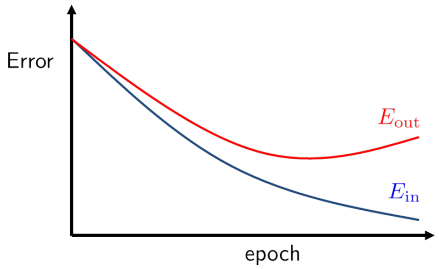


Figure 21: Target function: red. Hypothesis function: blue, which is a 4th order polynomial in this case.

Perfect fitting. Since we only have 5 data points but we use a 4th order polynomial, we can *perfectly* fits the five data points. (The polynomial degree is high enough to fit every one of the 5 data points, perfectly!) Therefore, we have $E_{\text{in}} = 0$ because the training error is zero. However, if we consider the out-sample error E_{out}, we know that it will be terrible because it is overfitted to the 5 data points. In other words, we are not able to generalize.

Is model complexity the reason? So, is model complexity the reason? Since we use a 4th order polynomial, perhaps our model is too complex. But this cannot be true. Today's neural networks are very complex. However, the generalization error goes down and then goes up as we increase the number of training

epochs. Our network capacity remains the same throughout the training, but how come the generalization error goes up and down? Certainly, it is not just the model complexity.

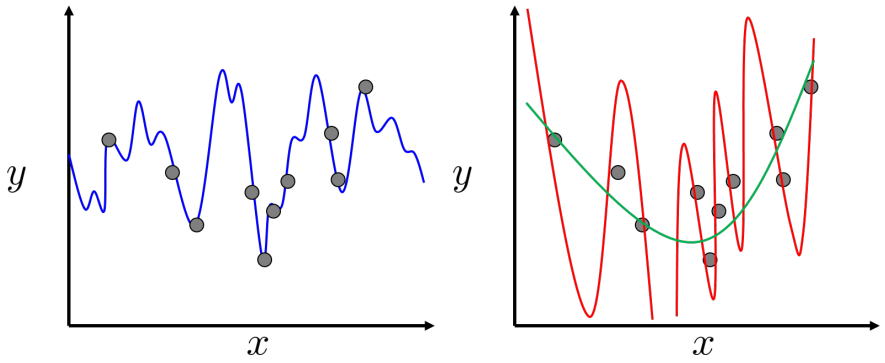


Is noise the reason? Noise could be a reason. But what if we use a 50th order polynomial to fit a *clean* dataset? Suppose we have a target function shown below and its corresponding data points. We fit the data using a 50th order polynomial, and we also fit the data using using a 2nd order polynomial. Since there is no noise, we do not need to worry about the corruption caused by noise.

But the result below shows something interesting. Even though there is no noise in the dataset, the 50th order polynomial still overfits (very significantly). We can evaluate the in-sample and out-sample error for the model:

- (i) Use a 2nd order polynomial: $E_{\text{in}} = 0.029$, $E_{\text{out}} = 0.120$
- (ii) Use a 10-th order polynomial: $E_{\text{in}} = 10^{-5}$, $E_{\text{out}} = 7680$

See, we overfit the *clean* data. So, noise is not the only reason for overfitting.

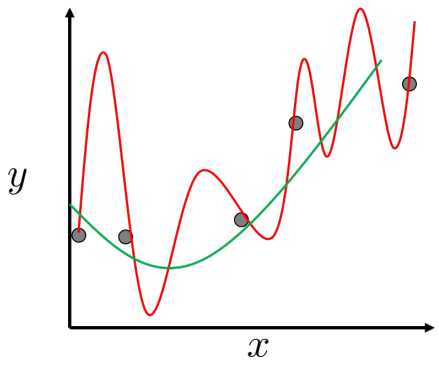


Is model mismatch the reason? Suppose the target function is a 10th order polynomial. What if the fitted curve is also a 10th order polynomial. Since now the hypothesis model matches with the target function, there should not be any overfitting. However, suppose we draw 5 data points from this 10th order polynomial. We assume that the data points are *clean*.

As you can see in the figure below, with 5 data points, a 10th order polynomial still overfits. In contrast, a 2nd order polynomial appears to fit very well to the data. So, the problem has to do with the number of training samples relative to the model we choose.

What have we learned about overfitting from the above two examples?

- Better functional approximation $\neq$ better generalization. (Sinusoid example)
- Model complexity alone is not the cause of overfitting. (Think about neural network size does not change during training.)
- Noise could be a reason, but it is not the only reason for overfitting. (We overfit even if the data is clean)
- Model mismatch could be a reason, but not the only reason. (We can match the model to the



target function, but we may still overfit.)

Therefore, overfitting, is the result of **a mix of** these issues, not one single factor alone.

4.4 Bias-variance analysis vs VC analysis

4.4.1 What does bias-variance tell us about overfitting?

So, how do we use the bias-variance analysis to understand the overfitting issue? We start by recalling our derivation of the out-sample error:

$$\begin{aligned} E_{\text{out}} &= \mathbb{E}_{\mathcal{D}}\left[\mathbb{E}_{\text{out}}(g^{(\mathcal{D})})\right] \\ &= \mathbb{E}_{\mathbf{x}}\left[\mathbb{E}_{\mathcal{D}}\left[g^{(\mathcal{D})}(\mathbf{x})^2\right] - 2\mathbb{E}_{\mathcal{D}}[g^{(\mathcal{D})}(\mathbf{x})]f(\mathbf{x}) + f(\mathbf{x})^2\right] \\ &= \mathbb{E}_{\mathbf{x}}\Big[\underbrace{\mathbb{E}_{\mathcal{D}}\left[g^{(\mathcal{D})}(\mathbf{x})^2\right] - \overline{g}(\mathbf{x})^2}_{\mathbb{E}_{\mathcal{D}}[(g^{(\mathcal{D})}(\mathbf{x})-\overline{g}(\mathbf{x}))^2]} + \underbrace{\overline{g}(\mathbf{x})^2 - 2\mathbb{E}_{\mathcal{D}}[g^{(\mathcal{D})}(\mathbf{x})]f(\mathbf{x}) + f(\mathbf{x})^2}_{(\overline{g}(\mathbf{x})-f(\mathbf{x}))^2}\Big]. \end{aligned} \tag{39}$$

In the presence of noise, what if we replace the target function f with the following noisy version?

$$f(\mathbf{x}) \longleftarrow f(\mathbf{x}) + \epsilon(\mathbf{x}), \tag{40}$$

where $\mathbb{E}[\epsilon(\mathbf{x})] = 0$ (meaning that the noise is zero-mean).

If we substitute this f into the above derivations, we can show that

$$\begin{aligned} &\mathbb{E}_{\mathcal{D},\epsilon}\left[(g^{(D)}(\mathbf{x}) - (f(\mathbf{x}) + \epsilon(\mathbf{x})))^2\right] \\ &= \mathbb{E}_{\mathcal{D},\epsilon}\left[(g^{(D)}(\mathbf{x}) - \overline{g}(\mathbf{x}) + \overline{g}(\mathbf{x}) - f(\mathbf{x}) - \epsilon(\mathbf{x}))^2\right] \\ &= \mathbb{E}_{\mathcal{D},\epsilon}\left[\left(g^{(D)}(\mathbf{x}) - \overline{g}(\mathbf{x})\right)^2 + (\overline{g}(\mathbf{x}) - f(\mathbf{x}))^2 + (\epsilon(\mathbf{x}))^2\right] \end{aligned}$$

By assuming that the cross-terms involving $\mathbb{E}[\epsilon(\mathbf{x})]$ are zero, we can show

$$\begin{aligned} E_{\text{out}} &= \mathbb{E}_{\mathbf{x}}[\text{everything on the right hand side of (39)}] \\ &= \mathbb{E}_{\mathcal{D},\mathbf{x}}\left[\left(g^{(D)}(\mathbf{x}) - \overline{g}(\mathbf{x})\right)^2\right] + \mathbb{E}_{\mathbf{x}}\left[(\overline{g}(\mathbf{x}) - f(\mathbf{x}))^2\right] + \mathbb{E}_{\mathbf{x},\epsilon}\left[\epsilon(\mathbf{x})^2\right] \end{aligned} \tag{41}$$

So, we observe that the out-sample error is affected, *simultaneously*, by three terms:

- Variance: $\mathbb{E}_{\mathcal{D},\mathbf{x}}\left[\left(g^{(D)}(\mathbf{x}) - \overline{g}(\mathbf{x})\right)^2\right]$. Larger variance means that you have larger fluctuation between the mean and the target. So even if the mean is close to the target, the huge variance will make the generalization worse. To reduce the variance, the only way is to increase the number of training samples. Once we have enough training samples, it matter less which particular training set we choose because all the training sets would have enough variety of data.

- Bias: $\mathbb{E}_{\mathbf{x}}\left[(\overline{g}(\mathbf{x})-f(\mathbf{x}))^2\right]$. The bias cannot be reduced unless we use a more sophisticated model (so that the model capacity increases. But if the model capacity increases, you will suffer from a high variance. So, ultimately, we go back to the equation of asking for more training data.
- Noise: $\mathbb{E}_{\mathbf{x},\epsilon}\left[\epsilon(\mathbf{x})^2\right]$. The amount of noise will be reduced if we have enough training samples. Since we assume that the noise is independent, the more samples we see, the weaker the contribution from the noise will be.

Therefore, to summarize

How to improve overfitting?

- Overfitting ↓ if number of data points ↑
- Overfitting ↑ if noise ↑
- Overfitting ↑ if target complexity ↑

4.4.2 Decomposing the learning curve

Both bias-variance and VC analysis provide a trade-off between model complexity and sample complexity. Figure 22 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 22.

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.

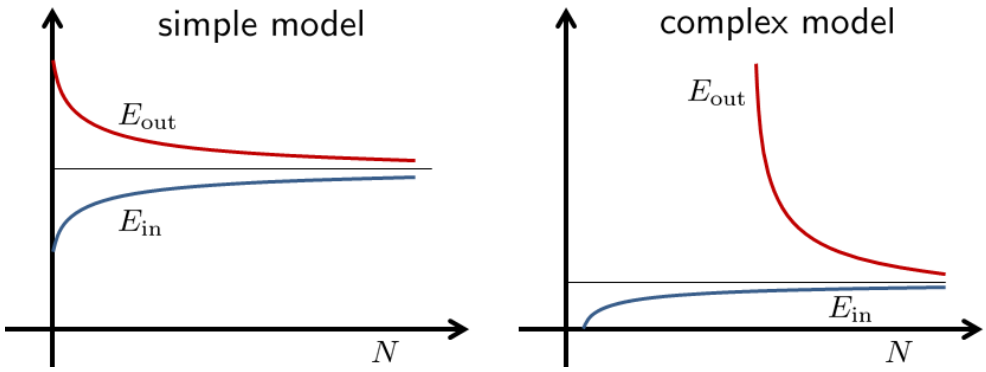


Figure 22: Learning curves of a simple and a complex model.

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 $\overline{g}$. 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.

How do we compare the decomposition of the learning curve using the bias-variance analysis and the VC analysis? As shown in Figure 23, the VC analysis gives us a decomposition based on the in-sample error and the generalization error, where the sum becomes the testing error. For bias-variance analysis, the decomposition is based on the mean hypothesis. Asymptotically, the in-sample error and the out-sample error will both converge to the error produced by the mean hypothesis. But at any instant, the testing error is decomposed into the deviation from mean to the truth, and then from the testing error to the mean.

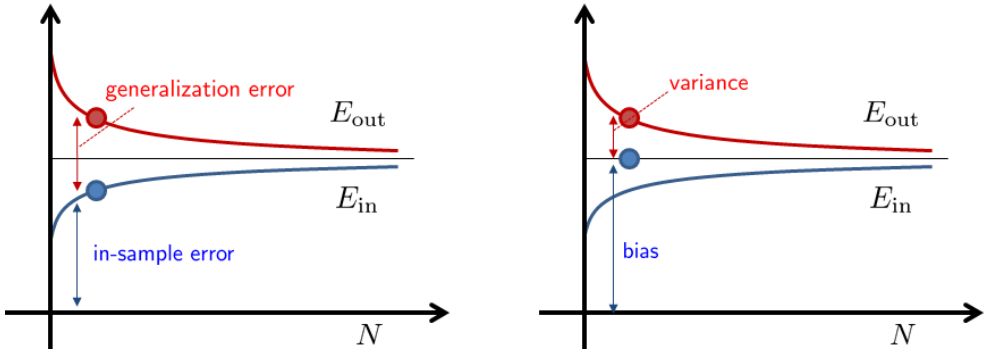


Figure 23: [Left] VC analysis. [Right] Bias-variance analysis

How does VC and bias-variance decompose the learning curve?

- VC decomposes the learning curve as out-sample error = in-sample error + generalization error.
- Bias-variance decomposes the learning curve as out-sample error = bias + variance.

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.