Chapter 2 The Learning Problem

2.1 What Is Learning?

Throughout we shall be concerned with signal processing aspect of machine learning. State it in other words, our primary interest is to study methods and algorithms that facilitate our *learning from data*. Most of the time in this course, the term *data* is taken to mean *measurements* that may come from real world, for example financial data from a stock exchange and precipitation readings from weather broadcasting, or from synthetic environments such as Monte-Carlo simulations that statistically evaluate certain computational algorithms.

Suppose we observe a data set, $\mathcal{D}$, which has an explicit and causal input-output structure. We want to utilize the data to learn the "*governing law*" (some authors in the literature call it *underlining process*, *target function*, *target distribution* [8], *target operator*, or *supervisor's operator* [1]) that is responsible for how the data were produced. Such a desire to learn is quite natural because understanding this governing law would greatly facilitate reliable prediction of what will occur next or, stated in a more formally, facilitate us to infer something *outside* $\mathcal{D}$. As a matter of fact, to infer what will occur outside $\mathcal{D}$ is the central goal of every (supervised) learning process.

At a glance, learning the governing law from a given set of data appears to be a hard mission, if not impossible.

Example 2.1

Figure 2.1 explains this point of view, where a data set may come from completely different "governing laws" – in this case the governing laws are two polynomial functions of order 2 and 5.

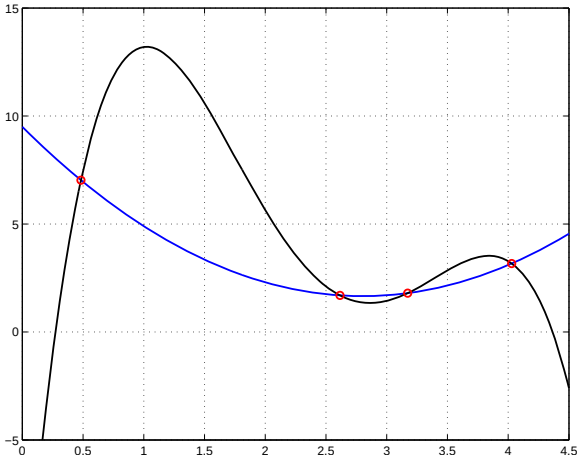


Figure 2.1 The data set $\{(x_i, y_i)$, for $i = 1, 2, 3, 4\}$, marked by small circles in red, may come from widely different governing laws which are represented by the blue and black curves respectively, casting doubts as whether or not one can infer the true governing law behind the data observed.

■

So is learning from data a hopeless business? As we will see shortly, the answer depends on how we look at the problem. Below we present an example of *learning by examples* and examine the

feasibility of learning from a probabilistic perspective.

Example 2.2

Suppose we have a box containing many balls which appear identical except the colors: they are either red or blue. The ratio of the number of red balls to the total number of balls is μ, in other words, the probability of red ball is μ. Note that μ is *not* random, in fact it is a constant. Obviously, μ is an important character of the box, thus we decide to conduct an experiment to learn μ by example. We try to learn μ by picking N balls at random from the box (each time a ball is picked, its color is recorded and the ball is then returned to the box) and computing the ratio ν of the number of red balls in the N samples to N. The value of ν is random because when the experiment is repeated a different value of ν might be obtained. On the other hand, it is quite clear that the true probability μ effects ν, and in this regard μ may be regarded as the governing law in this example, we try to learn the value of μ by examples (i.e. samples). To address the question of whether the learning is feasible, we notice the following:

- The probability of having a ν that is widely different from μ is small. For example, if $\mu = 0.9$, and $N = 10$, the probability of having a $\nu \le 0.1$ is 9.1×10^{-9}, meaning that the event of ($\mu = 0.9$) and the event of ($\nu \le 0.1$) are highly unlikely to occur simultaneously.
- When the sample number N is large, then with high probability ν well approximates μ. In fact, for any sample size N, the probability of "ν differs from μ by more than ε" is bounded from above as

$$P[|\nu - \mu| > \varepsilon] \le 2e^{-2\varepsilon^2 N} \tag{2.1}$$

for any $\varepsilon > 0$. The inequality in (2.1) is known as the *Hoeffding inequality* [8], see the appendix for a proof of (2.1). From (2.1) we see that for a given $\varepsilon > 0$, the chances of "ν deviating from μ by more than ε" are vanishing exponentially quickly as more (random) samples are utilized in computing ν.

In conclusion, *statistically* one can learn the (unknown) probability μ via ν as long as the sample size N is large enough, and the ability of statistical leaning in this particular case is quantified by (2.1). ■

2.2 Types of Learning

The basic assumption in our machine learning framework is that we have collected a set of observations which we called *data*, and the purpose of learning is to uncover the governing law, in one way or another, that is responsible for the production of the data observed. In the literature, this kind of learning strategy is called "*learning from examples*", and the observations used in the learning process are called *training data*.

Depending on the structure of the training data, machine learning methods may be categorized as *supervised* and *unsupervised* leaning.

A. *Supervised Learning*

A training data in supervised learning assumes the form $\{(\boldsymbol{x}_n, y_n), n = 1,2,...,N\}$ where $\boldsymbol{x}_n$ are examples from an input space $\mathcal{X}$ and $y_n \in \mathcal{Y}$ are produced by a (unknown) causal system (i.e., a

governing law) with $\boldsymbol{x}_n$ as input. For example, in a machine learning system that deals with credit card applications, $\boldsymbol{x}_n$ may represent a historic record of a customer and $y_n \in \mathcal{Y} = \{-1, 1\}$ labels the bank's approval or disapproval of the customer's application.

The most popular problems in supervised learning are *classification* and *regression*. In a classification problem, the goal is to assign each input (that is outside the training data) to one of the *finite number* of *discrete* categories. In a regression problem, the learning is aimed at finding a continuous-valued approximating function to fit a set of training data.

B. *Unsupervised Learning*

In unsupervised learning, we are just given a set of samples $\{\boldsymbol{x}_n, n = 1, 2, \ldots, N\}$, but the labels y_n are not available. Because y_n are missing, the available observations are called unlabeled data. The learning problems in this case are typically related to pattern classifications and structure identification (such as number of data clusters, centers of clusters, and clusters themselves) of the input data.

2.3 Perceptron Learning for Classification

A. *Perceptron Model and Perceptron Learning Algorithm (PLA)*

Let $\mathcal{X} = R^m$ and $\mathcal{Y} = \{-1, 1\}$ be the input and output spaces, respectively. Thus each input $\boldsymbol{x} \in \mathcal{X}$ is an m-dimensional vector whose components $\{x_i, i = 1, \ldots, m\}$ are called *features*. In a credit-card application problem, for example, these input components may include years in residence, salary, etc., and the output is a binary label 1 or –1, representing "approval" or "disapproval" of the application, respectively.

Perceptron, originally developed by F. Rosenblatt in 1958 [9], is based on a simple functional model

$$y = h(\boldsymbol{x}) = \text{sign}\left(b + \sum_{i=1}^{m} w_i x_i\right) \tag{2.2}$$

where sign(z) is the so-called *sign function* which equals 1 if $z > 0$ and –1 if $z < 0$; and b and w_i for $i = 1, \ldots, m$ are model parameters that can be trained by training data so as for (2.2) to make good decisions *outside* the training data. Note that, given parameters b and w_i's, the output y of the perceptron depends only on the *sign* of $b + \sum_{i=1}^{m} w_i x_i$, hence the equation

$$b + \sum_{i=1}^{m} w_i x_i = 0 \tag{2.3}$$

plays an important role in the decision making process. Geometrically, (2.3) represents an $(m - 1)$-dimensional hyperplane in the input space R^m, which is often called a *decision boundary*, that divides space R^m into two parts with the points $\boldsymbol{x}$ satisfying $b + \sum_{i=1}^{m} w_i x_i > 0$ (hence the associated output $y = 1$) on side of the boundary and the points $\boldsymbol{x}$ satisfying $b + \sum_{i=1}^{m} w_i x_i < 0$ (hence an output $y = -1$) on the other side of the boundary. From this perspective, the classification problem at hand amounts to utilizing the information provided by the training data $\mathcal{D} = \{(\boldsymbol{x}_n, y_n), n = 1, 2, \ldots, N\}$ to tune parameters b and w_i's so that the decision boundary in (2.3)

separates data pairs $(\boldsymbol{x}_n, y_n)$ with $y_n = 1$ from those $(\boldsymbol{x}_n, y_n)$ with $y_n = -1$.

Before describing the perceptron learning algorithm (PLA), let us simplify the notation by defining $\boldsymbol{w} = \begin{bmatrix} b & w_1 & \cdots & w_m \end{bmatrix}^T$ (in some literature the first component b is renamed as w_0) and $\boldsymbol{x} = \begin{bmatrix} 1 & x_1 & \cdots & x_m \end{bmatrix}^T$ so that (2.2) becomes

$$y = h(\boldsymbol{x}) = \text{sign}\left(\boldsymbol{w}^T \boldsymbol{x}\right) \tag{2.4}$$

Given training data $\mathcal{D} = \{(\boldsymbol{x}_n, y_n), n = 1, 2, ..., N\}$ where each $\boldsymbol{x}_k$ is now understood as an $(m+1)$-dimensional vector, the PLA updates the weights in an iterative manner. We start by setting the weight vector $\boldsymbol{w}$ in (2.4) to an initial guess $\boldsymbol{w}(0)$ and an iteration counter t to 0. Then for each pair $(\boldsymbol{x}_n, y_n)$ from the training data, we verify to see if the value of y in (2.4) at $\boldsymbol{x}_k$ coincides with y_k. With $\boldsymbol{w} = \boldsymbol{w}(0)$, (2.4) is said to classify $\boldsymbol{x}_n$ correctly if $\text{sign}\left(\boldsymbol{w}(0)^T \boldsymbol{x}_n\right) = y_n$, otherwise (2.4) is said to misclassify $\boldsymbol{x}_n$. If no data pairs in $\mathcal{D}$ are misclassified, then $\boldsymbol{w}(0)$ is claimed to be a solution that is expected to work well outside the training data. Otherwise, weight $\boldsymbol{w}$ is updated as follows.

Suppose the current weight $\boldsymbol{w}(t)$ misclassifies several data pairs from $\mathcal{D}$, we pick one of these pairs *at random*, denote it by $(\boldsymbol{x}(t), y(t))$, and update $\boldsymbol{w}(t)$ to $\boldsymbol{w}(t+1)$ using

$$\boldsymbol{w}(t+1) = \boldsymbol{w}(t) + y(t)\boldsymbol{x}(t) \tag{2.5}$$

With $\boldsymbol{w} = \boldsymbol{w}(1)$ and $t = 1$, the above procedure repeats itself and again, there are two possible outcomes – no misclassifications are found for the training data so that $\boldsymbol{w}(1)$ is a solution weight vector, or misclassifications occur and (2.5) is applied to update from $\boldsymbol{w}(1)$ to $\boldsymbol{w}(2)$, and so on. The iteration continues until the current $\boldsymbol{w}$ classifies all data pairs from $\mathcal{D}$ correctly.

B. *Remarks*

• To understand how (2.5) works, suppose $\boldsymbol{w}(t)$ misclassifies $(\boldsymbol{x}(t), y(t))$ in the training data. In other words, we have $\text{sign}(\boldsymbol{x}(t)^T \boldsymbol{w}(t)) = -y(t)$. Multiplying (2.5) by $\boldsymbol{x}(t)^T$ yields

$$\boldsymbol{x}(t)^T \boldsymbol{w}(t+1) = \underbrace{\boldsymbol{x}(t)^T \boldsymbol{w}(t)}_{\text{its sign is } -y(t)} + y(t) \cdot \underbrace{\boldsymbol{x}(t)^T \boldsymbol{x}(t)}_{\substack{\text{this part is always} \\ \text{positive, so the sign} \\ \text{of this term is } y(t)}} \tag{2.6}$$

We see that the last term in the equation always tries to fix the misclassification regardless of the value $y(t)$.

• The PLA iterations based on (2.5) is guaranteed to converge as long as the training data $\mathcal{D}$ is separable by a linear decision boundary. Here convergence is meant to identify a weight vector $\boldsymbol{w}^*$ which yields a decision boundary $\boldsymbol{x}^T \boldsymbol{w}^* = 0$ that separates the points (in $\mathcal{D}$) labeled by 1 from those points (in $\mathcal{D}$) labeled by –1. It can be shown [8] that the number of iterations for the PLA to converge is bounded from above by

$$t \le \frac{R^2 \, \| \boldsymbol{w}^* \|^2}{\rho^2} \tag{2.7a}$$

where

$$\rho = \min_{1 \le n \le N} y_n (\boldsymbol{x}_n^T \boldsymbol{w}^*) \quad \text{and} \quad R = \max_{1 \le n \le N} \| \boldsymbol{x}_n \| \tag{2.7b}$$

Example 2.3

A set of raining data $\mathcal{D} = \{(\boldsymbol{x}_n, y_n), n = 1, 2, ..., 15\}$ with $x_n \in R^2$ and $y_n \in \{1, -1\}$ are given below, where each column of three numbers is arranged as $[\boldsymbol{x}_n; y_n]$. The data are supposedly associated with a credit application record with the components of $\boldsymbol{x}_n$ representing years in residence and salary (in 10K dollars), respectively, of the nth applicant, and y_n representing the bank's approval (+1) or otherwise (–1).

Columns 1 through 10

1.0000	1.5000	2.1000	3.1000	3.5000	4.0000	5.0000	5.9000	6.9000	7.9000
4.5000	6.5000	3.5000	4.9000	6.5000	4.2000	5.0000	2.7000	3.7000	2.2000
-1.0000	1.0000	-1.0000	-1.0000	1.0000	-1.0000	1.0000	-1.0000	1.0000	-1.0000

Columns 11 through 15

8.4000	9.0000	10.0000	10.5000	11.2000
2.6000	1.6000	5.0000	0.8000	1.5000
1.0000	-1.0000	1.0000	-1.0000	1.0000

Fig. 2.2a depicts the training data where the points in blue are those in the training data with y_n = + 1 while the points in red are those with $y_n = -1$. Applying PLA to the training data with initial weight set to $\boldsymbol{w}(0) = [0\ 0\ 0\ ]^T$, it took the PLA 183 iterations to converge to the solution weights as

$$\boldsymbol{w}^* = \begin{bmatrix} -27 & 2 & 4.2 \end{bmatrix}^T$$

which yields a linear decision boundary

$$-27 + 2x_1 + 4.2x_2 = 0$$

This decision boundary represented by a straight line in black shown in Fig. 2.2b. We note the decision boundary successfully separates the two classes of training data.

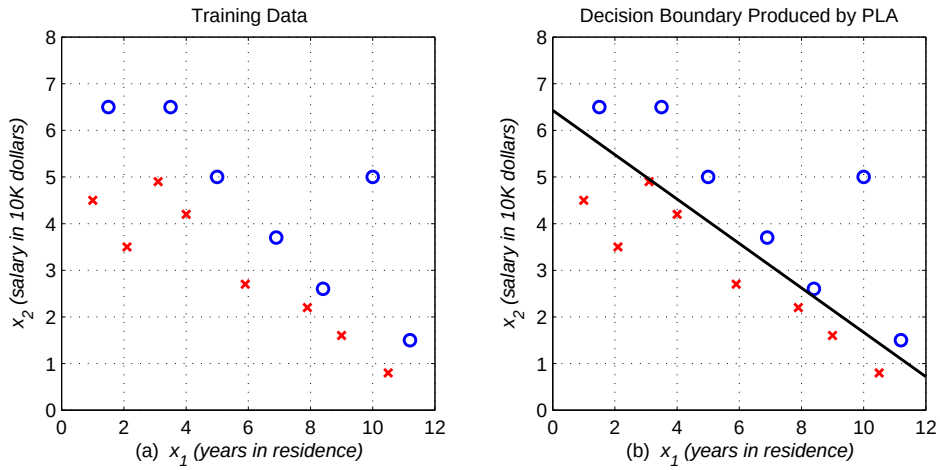


Figure 2.2

The MATLAB code implementing PLA for Example 2.3 is enclosed below.

```
% Perceptron learning algorithm (PLA) as applied to  training data in Example 2.3.
```

```
% Example: load data_ex2_3; [wt,t] = ex2_3_PLA(x,y,xp,xn,[0 0 0]',9);
% Written by W.-S. Lu, University of Victoria. Last modified: Jan. 1, 2015.
function [wt,t] = ex2_3_PLA(x,y,xp,xn,w0,st)
N = length(y);
t = 0;
wt = w0(:);
D = [ones(N,1) x'];
y = y(:);
flag = 0;
rand('state',st)
while flag == 0,
   Nw = randperm(N);
   tu = 0;
   tv = 0;
   while tu < N & tv == 0,
        ti = Nw(tu + 1);
        if sign(D(ti,:)*wt) ~= y(ti),
          tv = 1;
          t_ind = ti;
        end
        tu = tu + 1;
   end
   if tv == 0,
     flag = 1;
   else
     wt = wt + y(t_ind)*D(t_ind,:)';
     t = t + 1;
   end
end
z = sign(D*wt) - y;
ind = find(z ~= 0);
L = length(ind);
disp('number of perceptron iterations:')
t
disp('final weights:')
wt
disp('in-sample error:')
length(ind)/N
figure(1)
subplot(121)
plot(xp(1,:),xp(2,:),'bo','linewidth',1.5)
hold on
plot(xn(1,:),xn(2,:),'rx','linewidth',1.5)
grid
xlabel('(a)  \itx_1 (years in residence)')
ylabel('\itx_2 (salary in 10K dollars)')
axis([0 12 0 8])
axis square
title('Training Data')
hold off
subplot(122)
```

```
plot(xp(1,:),xp(2,:),'bo','linewidth',1.5)
hold on
plot(xn(1,:),xn(2,:),'rx','linewidth',1.5)
grid
xlabel('(b)  \itx_1 (years in residence)')
ylabel('\itx_2 (salary in 10K dollars)')
p1 = 0;
p2 = (-wt(2)*p1-wt(1))/wt(3);
q1 = 12;
q2 = (-wt(2)*q1-wt(1))/wt(3);
plot([p1 q1],[p2 q2],'k-','linewidth',1.5)
axis([0 12 0 8])
axis square
title('Decision Boundary Produced by PLA')
hold off
```
■

2.4 Adaptive Linear Neuron Algorithm for Perceptron Learning

A variant of PLA, known as the adaptive linear neuron (Adaline) algorithm for perceptron learning [8], modifies the updating law in (2.5) by taking into account the actual difference between the value $\boldsymbol{w}(t)^T \boldsymbol{x}(t)$ and label $y(t)$. To be specific, in each iteration one picks a data pair from the training data at random (hence it may or may not be misclassified by the current weight) and call it $(\boldsymbol{x}(t), y(t))$, then defines

$$\rho(t) = \boldsymbol{w}(t)^T \boldsymbol{x}(t)$$

and updates the weight vector as follows

$$\boldsymbol{w}(t+1) = \begin{cases} \boldsymbol{w}(t) + \eta \cdot (y(t) - \rho(t)) \cdot \boldsymbol{x}(t) & \text{if } y(t) - \rho(t) \le 1 \\ \boldsymbol{w}(t) & \text{otherwise} \end{cases} \tag{2.8}$$

where $\eta > 0$ is a scaling constant that needs to be set carefully for convergence.

Example 2.4

Apply Adaline algorithm to the problem in Example 2.3. With $\boldsymbol{w}(0) = [0\ 0\ 0]^T$ and $\eta = 0.03$, it took Adaline 727 iterations to yield a weight vector

$$\boldsymbol{w}^* = \begin{bmatrix} -45.5944 & 3.5885 & 6.7679 \end{bmatrix}^T$$

hence a linear decision boundary

$$-45.5944 + 3.5885 + 6.7679 x_2 = 0$$

This decision boundary is represented by a straight line in black in Fig. 2.3b. Note that the decision boundary also successfully separates the two classes of training data.

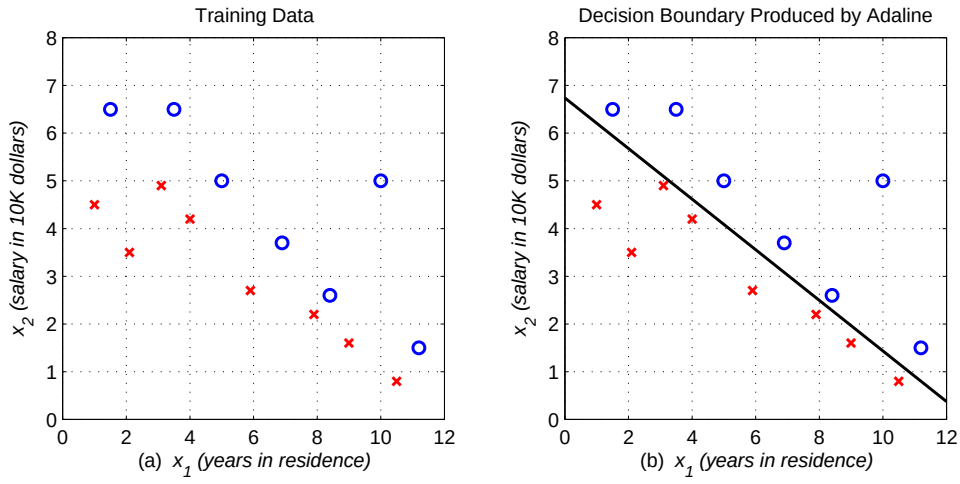


Figure 2.3

The MATLAB code that implements Adaline for Example 2.4 is enclosed below.

```
% Adaptive linear neuron (Adaline) algorithm as applied to training data in Example 2.3.
% Example: load data_ex2_3;  [wt,t] = ex2_3_adaline(x,y,xp,xn,[0 0 0]',0.03,9);
% Written by W.-S. Lu, University of Victoria. Last modified: Jan. 1, 2015.
function [wt,t] = ex2_3_adaline(x,y,xp,xn,w0,et,st)
N = length(y);
t = 0;
wt = w0(:);
D = [ones(N,1) x'];
y = y(:);
flag = 0;
rand('state',st)
while flag == 0,
   Nw = randperm(N);
   tu = 0;
   tv = 0;
   while tu < N & tv == 0,
        ti = Nw(tu + 1);
        if y(ti)*(D(ti,:)*wt) <= 1,
           tv = 1;
           t_ind = ti;
        end
        tu = tu + 1;
   end
   if tv == 0,
      flag = 1;
   else
      yi = y(t_ind);
      di = D(t_ind,:);
      wt = wt + et*(yi-di*wt)*di';
      t = t + 1;
   end
end
```

```
z = sign(D*wt) - y;
ind = find(z ~= 0);
L = length(ind);
disp('number of perceptron iterations:')
t
disp('final weights:')
wt
disp('in-sample error:')
length(ind)/N
figure(1)
subplot(121)
plot(xp(1,:),xp(2,:),'bo','linewidth',1.5)
hold on
plot(xn(1,:),xn(2,:),'rx','linewidth',1.5)
grid
xlabel('(a)  \itx_1 (years in residence)')
ylabel('\itx_2 (salary in 10K dollars)')
axis([0 12 0 8])
axis square
title('Training Data')
hold off
subplot(122)
plot(xp(1,:),xp(2,:),'bo','linewidth',1.5)
hold on
plot(xn(1,:),xn(2,:),'rx','linewidth',1.5)
grid
xlabel('(b)  \itx_1 (years in residence)')
ylabel('\itx_2 (salary in 10K dollars)')
p1 = 0;
p2 = (-wt(2)*p1-wt(1))/wt(3);
q1 = 12;
q2 = (-wt(2)*q1-wt(1))/wt(3);
plot([p1 q1],[p2 q2],'k-','linewidth',1.5)
axis([0 12 0 8])
axis square
title('Decision Boundary Produced by Adaline')
hold off
```
■

2.5 The Learning Problem

The supervised learning problem is illustrated in Fig. 2.4. The figure, which is from reference [6], depicts the basic structure of the problem as well as the data and information flow within and between various functional blocks. As such, the figure is useful for us to understand the inner working mechanism of supervised machine learning systems. In what follows we offer further explanations and comments on each of these functional blocks.

• In supervised learning, we are given a set of training data $\{(\boldsymbol{x}_n, y_n), n = 1,2,...,N\}$ and a set of hypotheses $\mathcal{H}$. The training data are produced by an unknown governing law which we want to learn, in Fig. 2.4 this governing law is seen as *unknown target distribution* $P(y \mid \boldsymbol{x})$ (or unknown target function $y = f(\boldsymbol{x})$). In compliance with the terminology in the literature, from now on we

shall use *target function* (or *target distribution*) to mean the governing law we want to uncover.

In practical applications, we encounter noisy targets when the output is not uniquely determined by the input. A general model for a noisy target is the conditional distribution $P(y \mid \boldsymbol{x})$ which, by definition, is the probability for a given input $\boldsymbol{x}$ the output assumes value y. A concrete way to think of the conditional distribution model for a noisy target is to express it as sum of a deterministic target function and a noise distribution, namely

$$y = f(\boldsymbol{x}) + \varepsilon \tag{2.9}$$

where function $f(\boldsymbol{x})$ is deterministic and ε is a random noise. For example, for the noise target in (2.9) with a Gaussian white noise ε with mean zero and variance σ^2, the target distribution is given by

$$P(y \mid \boldsymbol{x}) = \frac{1}{\sigma\sqrt{2\pi}} \int_{-\infty}^{y} e^{-(z-f(\boldsymbol{x}))^2/2\sigma^2} dz$$

It is worth pointing out that a deterministic target function may be considered as a special case of the more general model based conditional distributions. As a matter of fact, the conditional probability distribution given by

$$P(y \mid \boldsymbol{x}) = \begin{cases} 1 & \text{if } y = f(\boldsymbol{x}) \\ 0 & \text{if } y \neq f(\boldsymbol{x}) \end{cases} \tag{2.10}$$

models exactly a deterministic function $f(\boldsymbol{x})$.

• The hypothesis set $\mathcal{H}$ contains a (finite or infinite) set of candidate target functions (distributions) which is supposed to be rich enough to include the target function itself or candidates that are good approximations of the target function. On the other hand, it is very important to avoid using an unnecessarily large $\mathcal{H}$. Intuitively, a large hypothesis set makes it difficult for the learning algorithm to search and find an appropriate final hypothesis. It is true that the in-sample error reduces as the hypothesis set gets larger, however, having a small in-sample error is merely a step in a machining learning process, but not its goal -- a candidate hypothesis cannot be considered appropriate unless it yields a small out-of-sample error. We shall revisit this issue later with analysis tools such as VC bound and bias-variance decomposition.

• Based on the training data $\mathcal{D}$ and hypothesis set $\mathcal{H}$, it is the "learning algorithm" block that identifies a candidate hypothesis, say $g(\boldsymbol{x})$, such that $g(\boldsymbol{x}) \approx f(\boldsymbol{x})$. The candidate hypothesis $g(\boldsymbol{x})$ is shown in the "final hypothesis" block as an outcome of the learning algorithm.

• From the points made above, we realize that the target distribution and training examples are dictated by the problem, while the learning algorithm and hypothesis set are solution tools we have to choose. Together the hypothesis set and learning algorithm are referred to as the *learning model* [6].

• The "unknown input distribution" block is connected to "training examples" because the x_n's in the training data are generated in a way that obeys a certain probabilistic distribution $P(\boldsymbol{x})$. For example, the x_n's may be obtained by sampling a certain domain in input space $\mathcal{X}$ uniformly randomly.

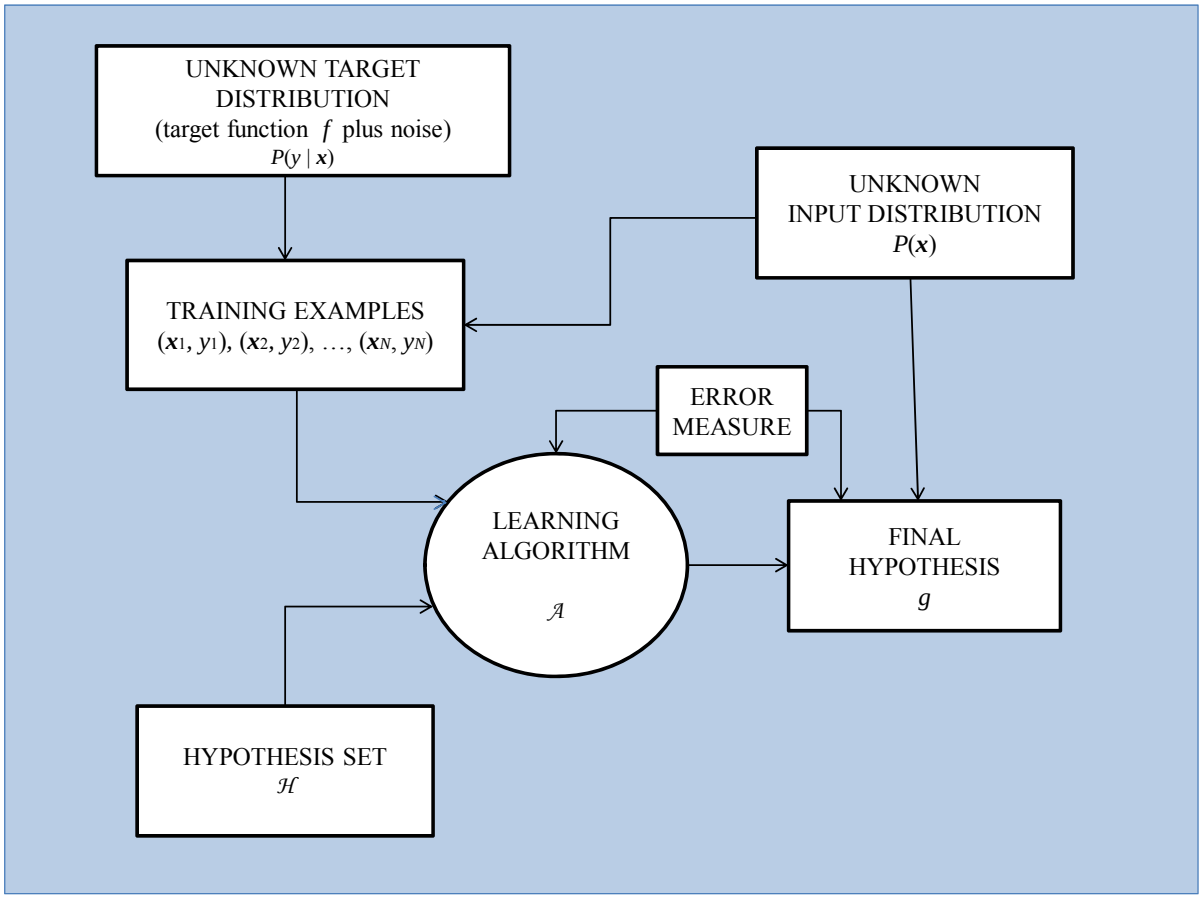


Figure 2.4 The supervised learning problem [6].

• The "unknown input distribution" block is also connected to the "final hypothesis" block because in order to evaluate the closeness of $g(\boldsymbol{x})$ to $f(\boldsymbol{x})$, the data $\boldsymbol{x}$ used must obey the *same distribution* $P(\boldsymbol{x})$ as in the training data.

• The "error measure" block is seen to be connected to "learning algorithm" as well as the "final hypothesis" block. This is because a learning algorithm inevitably involves computational error analysis, where having an adequately defined error measure is critical. Likewise, the justification of $g(\boldsymbol{x}) \approx f(\boldsymbol{x})$ clearly requires an error measure. We remark that in the literature error measure is often called *loss function*.

Let $h \in \mathcal{H}$ be a hypothesis that approximates target function f. We use $E(h, f)$ to denote an error measure to quantify the closeness between a hypothesis h and target function f. For the in-sample error, an error measure is substantiated through errors at individual input points $\boldsymbol{x}_n$, namely $e(h(\boldsymbol{x}_n), f(\boldsymbol{x}_n))$, and the average of this pointwise error is taken as an error measure, namely

$$E_{\text{in}}(h, f) = \frac{1}{N}\sum_{n=1}^{N} e(h(\boldsymbol{x}_n), f(\boldsymbol{x}_n)) = \frac{1}{N}\sum_{n=1}^{N} e(h(\boldsymbol{x}_n), y_n) \tag{2.11}$$

The performance of hypothesis h is not justified by whether or not its in-sample error is small but by whether or not its error outside the training data is small. A formal term to call the process of

applying a hypothesis $h(\boldsymbol{x})$ to outside training data in the input space is *generalization*, and generalization performance is evaluated by an error measure, called *out-of-sample* error and denoted by $E_{\text{out}}(h, f)$, that computes average of the *same* type of pointwise error $e(h(\boldsymbol{x}), f(\boldsymbol{x}))$ over the entire input space. Statistically the "average" becomes expectation, and the out-of sample error is given by

$$E_{\text{out}}(h, f) = E_x[e(h(\boldsymbol{x}), f(\boldsymbol{x}))] \tag{2.12}$$

In practice, the expectation in (2.12) is estimated by the average of the pointwise error over a finite set of points outside the training data, namely

$$E_{\text{out}}(h, f) \approx \frac{1}{K}\sum_{k=1}^{K} e(h(\boldsymbol{x}_k), f(\boldsymbol{x}_k)) = \frac{1}{K}\sum_{k=1}^{K} e(h(\boldsymbol{x}_k), y_k), \quad \text{for } (\boldsymbol{x}_k, y_k) \in \text{testing data} \tag{2.13}$$

We remark that for notation simplicity the "f" in both in-sample and out-of-sample error measures are often omitted so they can be written as $E_{\text{in}}(h)$ and $E_{\text{out}}(h)$, respectively.

• Several specific realizations of error measure that are found useful in machine learning are as follows (where only in-sample error measures are considered).

(i) *Classification error*

$$\begin{aligned} E_{in}(h, f) &= \text{fraction of } \mathcal{D} \text{ where } f \text{ and } h \text{ disagree} \\ &= \frac{1}{N}\sum_{n=1}^{N} [\![h(\boldsymbol{x}_n) \neq f(\boldsymbol{x}_n)]\!] \end{aligned} \tag{2.14}$$

where $[\![\text{statement}]\!]$ equals 1 if the statement is true, and equals 0 if the statement is false.

(ii) *L_2 error*

$$E_{in}(h) = \sum_{n=1}^{N} (h(\boldsymbol{x}_n) - y_n)^2 \tag{2.15}$$

(iii) *L_1 error*

$$E_{in}(h) = \sum_{n=1}^{N} | h(\boldsymbol{x}_n) - y_n | \tag{2.16}$$

(iv) *Error measure for logistic regression*

Logistic regression is a type of probabilistic classification model used for predicting a class labels based on input features. It is worth mentioning that as a classifier the target distribution in

logistic regression is a Bernoulli distribution, hence it relates to a hypothesis $h(\boldsymbol{x})$ in the form of

$$P(y \mid \boldsymbol{x}) = \begin{cases} h(\boldsymbol{x}) & \text{for } y = 1 \\ 1 - h(\boldsymbol{x}) & \text{for } y = -1 \end{cases} \tag{2.17}$$

Consequently, $h(\boldsymbol{x})$ is restricted to [0, 1] and acts like a probability. For these reasons hypotheses in logistic regression assume the form

$$h(\boldsymbol{x}) = \theta(\boldsymbol{w}^T \boldsymbol{x}) \tag{2.18a}$$

where θ is defined by

$$\theta(z) = \frac{e^z}{1+e^z} \tag{2.18b}$$

and is called *logistic function*. As shown in Fig. 2.5, function $\theta(z)$ goes from 0 to 1 monotonically as z goes from $-\infty$ to ∞.

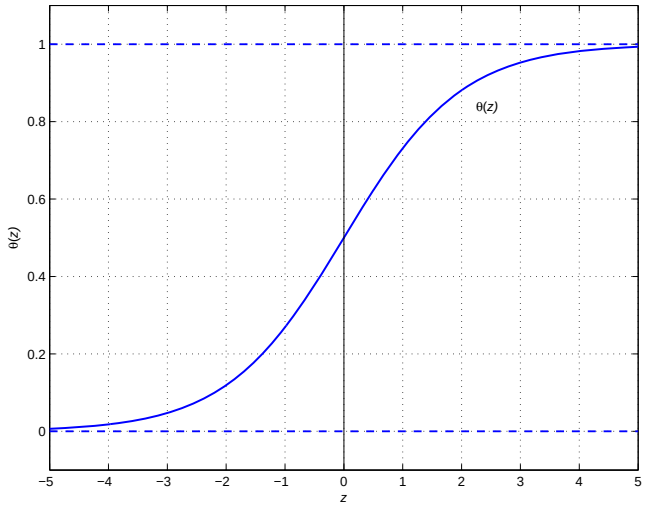


Figure 2.5 Function $\theta(z)$

The error measure for logistic regression is based on *maximum likelihood* (See Sec. 1.1.5). Suppose the training data $\mathcal{D} = \{(\boldsymbol{x}_n, y_n), n = 1, 2, ..., N\}$ are i.i.d., the probability of having this data set is given by

$$\prod_{n=1}^{N} P(y_n \mid \boldsymbol{x}_n) \tag{2.19}$$

where $P(y \mid \boldsymbol{x})$ is related to hypothesis $h(\boldsymbol{x})$ via (2.17). We maximize the probability in (2.19) in order for $h(\boldsymbol{x})$ to be a good hypothesis. Since maximizing (2.19) is equivalent to minimizing negative logarithm of the probability, in conjunction with (2.17) this leads us to minimizing

$$\frac{1}{N}\sum_{n=1}^{N} \ln\left(\frac{1}{P(y_n \mid \boldsymbol{x}_n)}\right) = \frac{1}{N}\sum_{n=1}^{N} \ln\left(\frac{1}{\theta(y_n \boldsymbol{w}^T \boldsymbol{x}_n)}\right) = \frac{1}{N}\sum_{n=1}^{N} \ln\left(1 + e^{-y_n \boldsymbol{w}^T \boldsymbol{x}_n}\right)$$

Therefore, the in-sample error measure for logistic regression problems is defined by

$$E_{in}(\boldsymbol{w}) = \frac{1}{N}\sum_{n=1}^{N} \ln\left(1 + e^{-y_n \boldsymbol{w}^T \boldsymbol{x}_n}\right) \tag{2.20}$$

2.6 Hoeffding Inequality

A. ***Hoeffding inequality***

The Hoeffding inequality was introduced early in Example 2.1 for learning the probability of a box of balls with two colors. We recall the Hoeffding inequality

$$P[|\nu - \mu| > \varepsilon] \le 2e^{-2\varepsilon^2 N} \tag{2.1}$$

for any $\varepsilon > 0$, where μ is the probability of (picking a) red ball and ν is the probability of a red ball in a sample of *N* balls. To see the implication of the inequality in learning, suppose we have a target function $f(\boldsymbol{x})$ and a hypothesis $h(\boldsymbol{x})$. In both training and testing, the event of $h(\boldsymbol{x}_n) = f(\boldsymbol{x}_n)$ is associated with a blue ball and the event of $h(\boldsymbol{x}_n) \neq f(\boldsymbol{x}_n)$ is associated with a red ball. In this way, a learning problem may be connected to the ball model as described in Example 2.2. Moreover, the correspondence between $h(\boldsymbol{x}_n) \neq f(\boldsymbol{x}_n)$ and a red ball implies the correspondence between the in-sample error $E_{\text{in}}(h, f)$ and probability ν, and the correspondence between the in-sample error $E_{\text{out}}(h, f)$ and probability μ. Consequently, in the context of machine learning, the Hoeffding inequality implies that

$$P[|E_{\text{in}}(h) - E_{\text{out}}(h)| > \varepsilon] \le 2e^{-2\varepsilon^2 N} \tag{2.21}$$

It should be stressed that (2.21) is valid only if for the case of a hypothesis that was *fixed before* the data set is generated. For a learning problem with a set $\mathcal{H}$ of finite *M* hypothesis, the question becomes how to bound the probability $P[|E_{\text{in}}(g) - E_{\text{out}}(g)| > \varepsilon]$ where *g* is the final hypothesis. Since how the final hypothesis is identified depends on the data set used, *g* is not fixed before the data set is generated, bounding $P[|E_{\text{in}}(g) - E_{\text{out}}(g)| > \varepsilon]$ is not as simple as plugging *g* into (2.21). In the case of a finite set of *M* hypothesis in $\mathcal{H} = \{h_1, h_2, \ldots, h_M\}$, the event $|E_{\text{in}}(g) - E_{\text{out}}(g)| > \varepsilon$ occurs if at least one of the events $(|E_{\text{in}}(h_1) - E_{\text{out}}(h_1)| > \varepsilon)$, $(|E_{\text{in}}(h_2) - E_{\text{out}}(h_2)| > \varepsilon)$, …, $(|E_{\text{in}}(h_M) - E_{\text{out}}(h_M)| > \varepsilon)$. Therefore we have

$$P[(|E_{\text{in}}(g) - E_{\text{out}}(g)| > \varepsilon)] \le \sum_{m=1}^{M} P[(|E_{\text{in}}(h_m) - E_{\text{out}}(h_m)| > \varepsilon)]$$

which in conjunction with (2.21) leads to

$$P[(|E_{\text{in}}(g) - E_{\text{out}}(g)| > \varepsilon)] \le 2Me^{-2\varepsilon^2 N} \tag{2.22}$$

Note that the bound in (2.22) is very conservative because the events $(|E_{\text{in}}(h_1) - E_{\text{out}}(h_1)| > \varepsilon)$, $(|E_{\text{in}}(h_2) - E_{\text{out}}(h_2)| > \varepsilon)$, …, $(|E_{\text{in}}(h_M) - E_{\text{out}}(h_M)| > \varepsilon)$ are usually highly overlapping with each other. Another problem with (2.22) is that it makes sense only when the hypothesis set is finite, which is hardly the case for many practical learning problems.

B. Why do we care about estimations in (2.21) and (2.22)?

The type of estimations in (2.21) and (2.22) is important because it offers insights of how a good learning strategy could be conceived. In case of a finite hypothesis set $\mathcal{H}$ with a small *M*, for example, (2.22) implies with a probability at least $1 - \delta$ that

$$E_{\text{out}}(g) \le E_{\text{in}}(g) + \sqrt{\frac{1}{2N} \ln \frac{2M}{\delta}} \tag{2.23}$$

Note that give *M* and δ the second term on the right-hand side of (2.23) can be made arbitrarily small provided that the samples in the training data, *N*, is large enough. Under this circumstance, (2.23) reveals that the out-of-sample error $E_{\text{out}}(g)$ shall be small as long as the in-sample error $E_{\text{in}}(g)$ is small.

2.7 The Vapnik-Chervonenkis (VC) Dimension

As noted above, (2.23) fails to work for learning systems with infinite hypothesis set $\mathcal{H}$. Besides, for a finite hypothesis set $\mathcal{H}$, (2.23) does not take highly likely overlap among the events $(|E_{\text{in}}(h_1) - E_{\text{out}}(h_1)| > \varepsilon)$, $(|E_{\text{in}}(h_2) - E_{\text{out}}(h_2)| > \varepsilon)$, …, $(|E_{\text{in}}(h_M) - E_{\text{out}}(h_M)| > \varepsilon)$ into account.

The VC dimension is a key quantity in the statistical learning theory [1], [2] that resolves these issues. There are plenty of examples where the "working domain" contains infinitely many items, yet each item can be characterized with a finite (and same) set of basic elements or parameters: infinite number of vectors and points in space R^n, for example, can all be expressed as linear combinations of (the same) n basis vectors.

Definition 2.1 [3] *VC dimension*

In plain terms, the VC dimension of a hypothesis set of the form $\{h(\boldsymbol{x}, \boldsymbol{w})\}$ is defined as the largest number of points (in some configuration) that can be shattered by members of $\{h(\boldsymbol{x}, \boldsymbol{w})\}$ (a set of points is said to be *shattered* by a hypothesis set $\mathcal{H}$ if, no matter how one assigns a binary label to each point, there exists a member of $\mathcal{H}$ that can perfectly separate them). ■

Example 2.5

Here we consider input space R^2. Suppose $\mathcal{H}$ is the set of perceptrons, i.e., $h(\boldsymbol{x}, \boldsymbol{w}) = \text{sign}(w_0 + w_1 x_1 + w_2 x_2)$, whose decision boundaries are straight lines on the plane. Since the largest number of points a straight line can separate regardless of how these points are labeled is three, the VC dimension of hypothesis set $\mathcal{H} = \{h(\boldsymbol{x}, \boldsymbol{w}) = \text{sign}(w_0 + w_1 x_1 + w_2 x_2)\}$, denoted by d_{vc}, is three. Note that the VC dimension in this case coincides with the number parameters that characterize the hypothesis set. ■

This example indicates that although the number of hypothesis in $\mathcal{H}$ is infinite, depending on the "complexity" of the hypothesis set, its VC dimension might be finite, and it measures the effective parameters or degree of freedom of the hypothesis set.

Example 2.6

The d-dimensional perceptrons $\{h(\boldsymbol{x}, \boldsymbol{w}) = \text{sign}(\boldsymbol{w}^T \boldsymbol{x})\}$ with $\boldsymbol{x} = \begin{bmatrix} 1 & x_1 & \cdots & x_d \end{bmatrix}^T$ is $d_{\text{vc}} = d + 1$ [8]. ■

The importance of the notion of VC dimension may be appreciated via the following theorem which provides a generalization bound that is way better than (2.23).

Theorem *VC generalization bound* [8]

For any tolerance $\delta > 0$,

$$E_{\text{out}}(g) \le E_{\text{in}}(g) + \sqrt{\frac{8}{N} \ln \frac{4 m_{\mathcal{H}}(2N)}{\delta}} \tag{2.24}$$

with probability $\ge 1 - \delta$, where $m_{\mathcal{H}}$ is called the *growth function* for hypothesis set $\mathcal{H}$, which is shown to be a polynomial in N and bounded by

$$m_{\mathcal{H}}(N) \le N^{d_{\text{vc}}} + 1 \qquad \blacksquare \tag{2.25}$$

By combining (2.24) with (2.25), we obtain a generalization bound that is directly related the VC dimension d_{vc} as

$$E_{\text{out}}(g) \le E_{\text{in}}(g) + \sqrt{\frac{8}{N}\ln\frac{4(2N)^{d_{vc}}+4}{\delta}} \tag{2.26}$$

Several remarks on (2.24) and (2.26) are now in order.

• Bounds in (2.24) and (2.26) are widely applicable as they are valid for all hypothesis sets, learning algorithms, input spaces, probability distributions, and binary target functions. As such, however, these bounds turn out to be quite loose. On all account, there are two points worth mentioning: First, although the bounds are loose, they tend to be *equally* loose for different learning models, hence the bounds remain useful for comparing the performance of these models. Second, analysis based on VC dimension is found useful in practice because learning models with lower d_{vc} are found to generalize better than those with higher d_{vc}. Moreover, requiring N (the number of example in training data) to be at least $10 \times d_{\text{vc}}$ for decent performance has been as a popular rule of thumb [8].

• A very important contribution of the bounds in (2.24) and (2.26) is that they establishes the feasibility of learning models with infinite hypothesis sets. Specifically, these bounds assert that as far as the VC dimension of the hypothesis $\mathcal{H}$ is finite, the generalization error can be made arbitrarily close to the in-sample error when sufficiently many training examples are utilized. The generalization performance can be characterized by parameters ε and δ where ε is the error tolerance of $E_{\text{out}}(g)$ deviating from $E_{\text{in}}(g)$, and the confidence parameter δ denotes the probability that the error tolerance ε is violated. By the definition of ε and (2.24), it follows that

$$\sqrt{\frac{8}{N}\ln\frac{4m_{\mathcal{H}}(2N)}{\delta}} \le \varepsilon$$

which leads to

$$N \ge \frac{8}{\varepsilon^2}\ln\frac{4m_{\mathcal{H}}(2N)}{\delta} \tag{2.27}$$

The counterpart of (2.27) for the bound in (2.26) is given by

$$N \ge \frac{8}{\varepsilon^2}\ln\frac{4(2N)^{d_{vc}}+4}{\delta} \tag{2.28}$$

These lower bounds for N describe *sample complicity* for achieving desirable generalization performance (ε,δ). As expected, these bounds are quite loose, namely the number of examples needed to achieve the performance are way more than necessary. Nevertheless, the bounds are explicitly indicative of the feasibility of learning as long as the VC dimension d_{vc} is *finite*.

• Let the second term on the right-hand side of (2.24) be denoted by $\Omega(N,\mathcal{H},\delta)$, namely

$$\Omega(N,\mathcal{H},\delta) = \sqrt{\frac{8}{N}\ln\frac{4m_{\mathcal{H}}(2N)}{\delta}} \le \sqrt{\frac{8}{N}\ln\frac{4(2N)^{d_{vc}}+4}{\delta}} \tag{2.29}$$

so we can write

$$E_{\text{out}}(g) \le E_{\text{in}}(g) + \Omega(N,\mathcal{H},\delta) \tag{2.30}$$

In a typical learning problem the training data $\mathcal{D}$ is given, hence N is fixed. Now suppose the

confidence parameter δ is also given. The term $\Omega(N,\mathcal{H},\delta)$ in this case is determined by the VC dimension hence the model complexity. As such, $\Omega(N,\mathcal{H},\delta)$ serves as a *penalty* for the generalization performance. On the other hand, increasing model complexity (i.e. allowing $\mathcal{H}$ to include more hypotheses) in general reduces the in-sample error because with a richer $\mathcal{H}$ we are likely to fit the training data better. Therefore, we are in a scenario of having an increasing $\Omega(N,\mathcal{H},\delta)$ and a decreasing $E_{\text{in}}(g)$ as the model complexity goes up, in other words the generalization performance depends on the interplay between the two factors. This situation is illustrated in Fig. 2.6 [8].

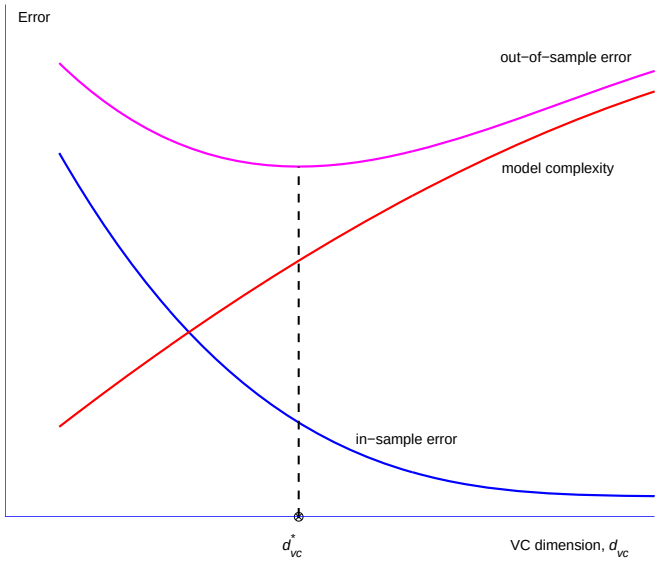


Figure 2.6 As VC dimension increases, in-sample error reduces while penalty $\Omega(N,\mathcal{H},\delta)$ due to the increase of model complexity goes up. As a result, the out-of-sample error will not be small if the hypothesis set $\mathcal{H}$ is too small or too large. The optimal model with VC dimension d_{vc}^{*} is a compromise that minimizes the combination of $E_{\text{in}}(g)$ and $\Omega(N,\mathcal{H},\delta)$ [8].

2.8 Bias-Variance Decomposition

The VC analysis provides a generalization bound as a sum of $E_{\text{in}}(g)$ plus a penalty term $\Omega(N,\mathcal{H},\delta)$. In this section, we study generalization performance through another approach known as *bias-variance decomposition* [3][7][8]. The approach is well suited for L_2 error measures and the results are statistically intuitive. The following analysis follows [8].

From (2.12) and (2.15), it follows that when the L_2-error is employed the out-of-sample error for the final hypothesis $g(\boldsymbol{x})$ is given by

$$E_{\text{out}}(g^{(\mathcal{D})}) = E_{\boldsymbol{x}}\left[\left(g^{(\mathcal{D})}(\boldsymbol{x}) - f(\boldsymbol{x})\right)^2\right] \qquad (2.31)$$

where the notation $g^{(\mathcal{D})}$ stresses that the final hypothesis depends on training data $\mathcal{D}$. To obtain a generalization performance bound that is independent of any particular selection of $\mathcal{D}$, we compute the expectation of $E_{\text{out}}(g^{(\mathcal{D})})$ with respect to $\mathcal{D}$. From (2.31), we can write

$$
\begin{aligned}
E_{\mathcal{D}}\left[E_{\text{out}}(g^{(\mathcal{D})})\right] &= E_{\mathcal{D}}\left[E_{\boldsymbol{x}}\left[\left(g^{(\mathcal{D})}(\boldsymbol{x})-f(\boldsymbol{x})\right)^2\right]\right] \\
&= E_{\boldsymbol{x}}\left[E_{\mathcal{D}}\left[\left(g^{(\mathcal{D})}(\boldsymbol{x})-f(\boldsymbol{x})\right)^2\right]\right] \\
&= E_{\boldsymbol{x}}\left[E_{\mathcal{D}}[g^{(\mathcal{D})}(\boldsymbol{x})^2]-2E_{\mathcal{D}}[g^{(\mathcal{D})}(\boldsymbol{x})]f(\boldsymbol{x})+f(\boldsymbol{x})^2\right]
\end{aligned}
\tag{2.32}
$$

where $E_{\mathcal{D}}[g^{(\mathcal{D})}(\boldsymbol{x})]= \bar{g}(\boldsymbol{x})$ represents an average function where the average is performed with respect to all possible data set of size N. Obviously $\bar{g}(\boldsymbol{x})$ may be approximated by generating enough number of data sets $\mathcal{D}_1$, $\mathcal{D}_2$, …, $\mathcal{D}_K$, applying the learning algorithm to each $\mathcal{D}_k$ to obtain $g^{(\mathcal{D}_k)}(\boldsymbol{x})$, and computing $\bar{g}(\boldsymbol{x}) \approx \frac{1}{K}\sum_{k=1}^{K} g^{(\mathcal{D}_k)}(\boldsymbol{x})$. It now follows from (2.32) that

$$
\begin{aligned}
E_{\mathcal{D}}\left[E_{\text{out}}(g^{(\mathcal{D})})\right] &= E_{\boldsymbol{x}}\underbrace{\left[\left(\bar{g}(\boldsymbol{x})-f(\boldsymbol{x})\right)^2\right]}_{\text{bias}(\boldsymbol{x})}+E_{\boldsymbol{x}}\underbrace{\left[E_{\mathcal{D}}\left[\left(g^{(\mathcal{D})}(\boldsymbol{x})-\bar{g}(\boldsymbol{x})\right)^2\right]\right]}_{\text{variance}(\boldsymbol{x})} \\
&= E_{\boldsymbol{x}}[\text{bias}(\boldsymbol{x})]+E_{\boldsymbol{x}}[\text{var}(\boldsymbol{x})] \\
&= \text{bias} + \text{var}
\end{aligned}
\tag{2.33}
$$

In the case of a small hypothesis set $\mathcal{H}$, the chances of finding a $g(\boldsymbol{x})$ from $\mathcal{H}$ that well approximates the target function $f(\boldsymbol{x})$ are not be great, hence the bias is expected to be large. On the other hand, when $\mathcal{H}$ contains small number of candidate hypotheses, the chances of having a large difference between $\bar{g}(\boldsymbol{x})$ and $g^{(\mathcal{D}_k)}(\boldsymbol{x})$ is not great either, hence the variance in this case is expected to be small. As the model complexity increases, the approximation of target function by the average final hypothesis is likely to improve, thus the bias term in (2.33) will in general reduce. On the other hand, as more candidate hypotheses are included in $\mathcal{H}$, deferent training data sets are more likely to lead to different final hypotheses, hence an increased variance. Eq. (2.33) indicates that the average out-of-sample error depends on the model complexity through both the bias and variance terms as quantifies by (2.33). We also remark that as functions of model complexity, bias and especially variance are not necessarily monotonic. In fact the variance can be viewed as a measure of instability in the sense that it reflects the sensitivity of learning outcomes to small variations in the data.

Example 2.7

This example is aimed at illustrating the bias-variance decomposition. A total of $K = 400$ sets of training data are generated, each consists of $N = 10$ points $\{(x_n, y_n), n = 1, 2, \ldots, 10\}$ where xn are sampled uniformly randomly from interval [0, 1], and y_n's were produced by target function $y_n = f(x_n)$ where

$$f(x) = \cos(2\pi x) + 0.5\sin(3.5\pi x)$$

We examine ten learning models, each consists of all mth-order polynomials, with $m = 0, 1, \ldots, 9$. For a given hypothesis set $\mathcal{H}_m$ of mth-order polynomials of the form

$$h(x, \boldsymbol{w}) = w_0 + w_1 x + \cdots + w_m x^m \tag{2.34}$$

the final hypothesis was selected by minimizing the in-sample error measure

$$E_{\text{in}}(h, \boldsymbol{w}) = \frac{1}{N}\sum_{n=1}^{N}\big(h(x_n, \boldsymbol{w}) - y_n\big)^2$$

with respect to $\boldsymbol{w}$, where $\{(x_n, y_n), n = 1, 2, \ldots, 10\}$ is one of the training data set, say $\mathcal{D}^{(k)}$. Let $\boldsymbol{w}^*$ be the minimizer obtained, the final hypothesis is then identified as $g^{(\mathcal{D}_k)}(x) = h(x, \boldsymbol{w}^*)$. In this way, the average final hypothesis $\bar{g}(\boldsymbol{x})$ is found to be

$$\bar{g}(\boldsymbol{x}) = \frac{1}{400}\sum_{k=1}^{400} g^{(\mathcal{D}_k)}(x)$$

and the bias-variance decomposition (2.33) was carried out to obtain the bias and variance for that particular value of m which is obviously an indicator of the model complexity. The simulations were repeated for each of $m = 0, 1, \ldots, 9$, and the bias, variance as well as sum of the two quantities as average out-of-sample error with respect to m are depicted in Fig. 2.7.

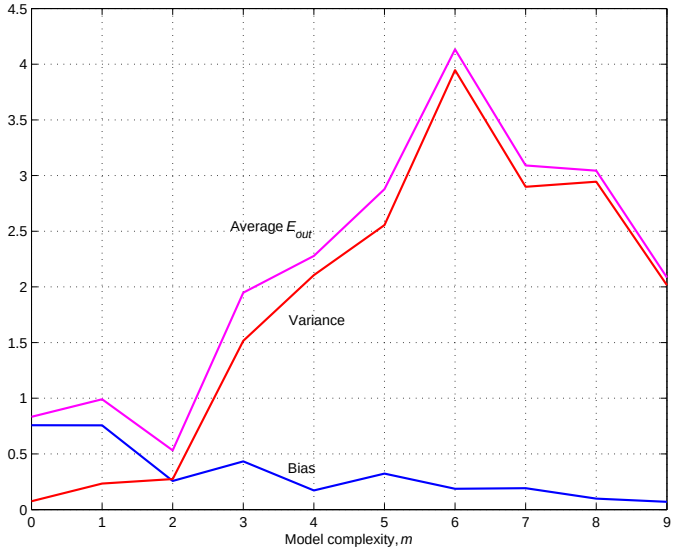


Figure 2.7 Bias and variance versus model complexity for Example 2.7.

We observe from the figure that the hypothesis set consisting of 2nd-oder polynomials appears to be a good choice as it offers a reasonable bias and is relatively insensitive to the variations in training data. This observation is confirmed by Fig. 2.8, where the 400 2nd-order final hypotheses (in blue) are shown in comparison with the target function $f(x)$ (in red). For comparison, Fig. 2.9 displays 400 6th-order final hypotheses versus the target function where the model is shown to be highly unstable against the variations in training data.

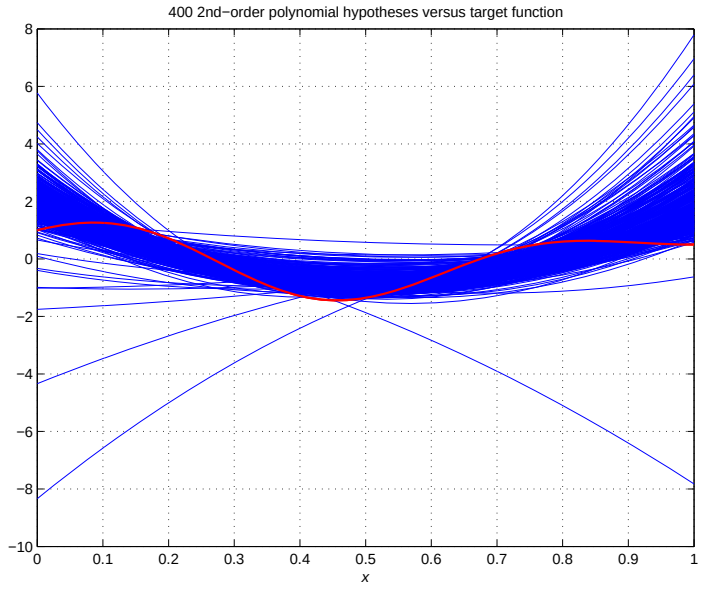


Figure 2.8 Second-order polynomial final hypotheses (in blue) versus target function (in red).

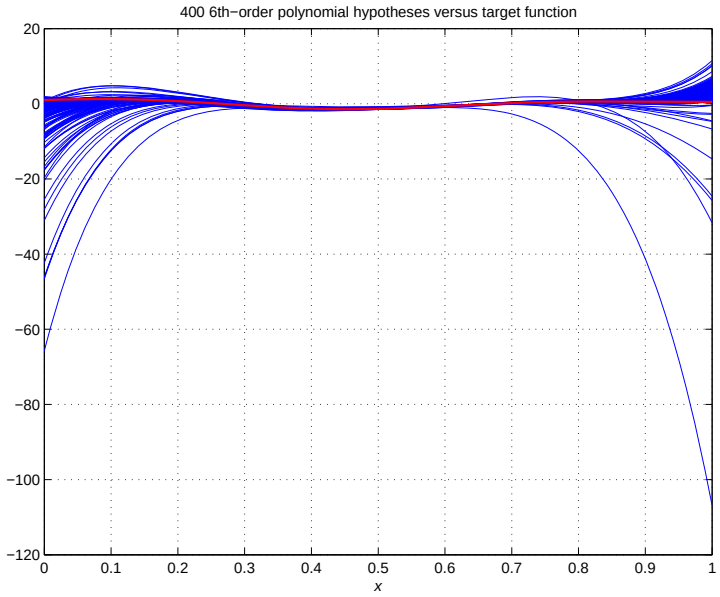


Figure 2.9 Sixth-order polynomial final hypotheses (in blue) versus target function (in red).

Below is the MATLAB code for the simulations described in Example 2.7.

```
% Bias-variance decomposition as discussed in Example 2.7.
% Example:  [bias,variance] = ex2_7(10,9,400,5);
% Written by W.-S. Lu, University of Victoria.
% Last modified: Dec. 15, 2014.
function [bias,variance] = ex2_7(N,M,K,st)
xe = 0:1/999:1;
xe = xe(:);
f = cos(2*pi*xe) + 0.5*sin(3.5*pi*xe);
```

```
bias = zeros(M+1,1);
variance = bias;
e_out = bias;
rand('state',st)
S = rand(N,K);
p = max(N,M+1);
if p > N,
   a = ones(p-N,1);
else
   a = [];
end
for t1 = 1:(M+1),
    gb = 0;
    Gx = [];
    for t2 = 1:K,
        x = S(:,t2);
        xx = [x; a];
        V = fliplr(vander(xx));
        X = V(1:N,1:t1);
        y = cos(2*pi*x) + 0.5*sin(3.5*pi*x);
        gw = flipud(pinv(X)*y);
        gb = gb + gw;
        gx = polyval(gw,xe);
        Gx = [Gx gx];
    end
    gb = gb/K;
    gbx = polyval(gb,xe);
    vw = 0;
    for i = 1:K,
        vw = vw + (Gx(:,i) - gbx).^2;
    end
    variance(t1) = mean(vw/K);
    bias(t1) = mean((gbx - f).^2);
    e_out(t1) = bias(t1) + variance(t1);
end
figure(1)
m = 0:1:M;
plot(m,bias,'b-','linewidth',1.5)
hold on
plot(m,variance,'r-','linewidth',1.5)
plot(m,e_out,'m-','linewidth',1.5)
xlabel('Model complexity, \itm')
grid
hold off
```

■

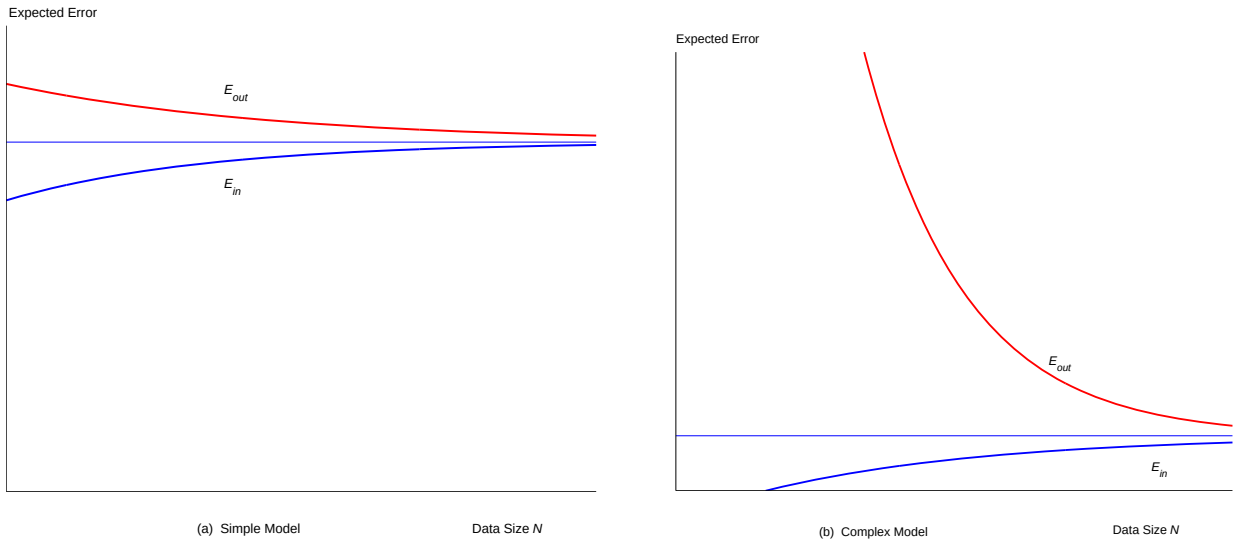
2.9 Expected Errors versus Number of Data Points

Through the VC bounds and bias-variance decomposition we have seen the general behavior of in-sample and out-of-sample errors with respect to model complexity. The VC bound is

explicitly related to the size N of the training data, it is also useful for analyzing the behavior of E_{in} , E_{out} , as well as their relation as N vary. In summary, we can state the following:

- Regardless of model complexity, as N gets larger E_{out} reduces and the difference between E_{out} and E_{in} narrows.
- Regardless of model complexity, the in-sample error will increase as size N increases because it becomes increasingly difficult to minimize in-sample error measure that has to take care of individual errors at more data points. However, since the data points are governed by a underlining target function $f(\boldsymbol{x})$ (or a conditional probability distribution), errors at data points are not at all random but rather coherent. As a matter of fact, as $N \to \infty$ the in-sample error will converge to $E_x[e(g(\boldsymbol{x}), f(\boldsymbol{x}))]$ (such as $E_x[|g(\boldsymbol{x}) - f(\boldsymbol{x})|^2]$). In other words, the in-sample error will reach a steady state as the sample size continues to grow.
- With a simple model (i.e. a small VC dimension), E_{out} will quickly going down to approach the level of E_{in}. However, it is usually very hard for a simple hypothesis to fit large number of data points, thus the steady-state level of the in-sample error (as well as out-of-sample error) will be high.
- Opposite to the above case, with a complex model (i.e. a larger VC dimension), E_{out} will approach the level of E_{in} much slowly. However, as data size N increases the steady-state level for both in-sample and out-of-sample errors are at a lower level.

The points made above are illustrated in Fig. 2.10a for the case of a simple model and Fig. 2.10b for the case of a complex model. In the literature, the type of curves in Fig. 2.10 are called learning curves [8].



(a) (b)

Figure 2.10 Expected in-sample and out-of-sample errors vs. data size N for (a) simple model and (b) complex model [8].

Appendix: A proof of the Hoeffding Inequality

The proof presented below follows R. Nowak's 2007 course notes on statistical learning theory. It consists of three steps.

Step 1 *Markov inequality*

Let z be a nonnegative random variable and t be a positive constant. Then

$$P(z \geq t) \leq \frac{E(z)}{t} \tag{A.1}$$

Proof: Let p be the density function of random variable z, by definition we can write

$$E(z) = \int_0^\infty yp(y)dy \geq \int_t^\infty yp(y)dy \geq t\int_t^\infty p(y)dy = tP(z \geq t)$$

which implies (A.1). ■

Now let u be a random variable with mean μ and variance σ^2, then $(u-\mu)^2$ is a nonnegative random variable whose mean is equal to σ^2. By applying (A.1) with $z = (u-\mu)^2$, we obtain

$$P[(u-\mu)^2 \geq t] \leq \frac{\sigma^2}{t} \tag{A.2}$$

Furthermore, let $u_1, u_2, \ldots, u_N$ be i.i.d. random variables, each with mean μ and variance σ^2, and let $u = \frac{1}{N}\sum_{n=1}^{n} u_n$. Obviously, $E(u) = \mu$ and $E[(u-\mu)^2] = \frac{\sigma^2}{N}$, hence (A.2) implies that

$$P[(u-\mu)^2 \geq t] \leq \frac{\sigma^2}{Nt} \tag{A.3}$$

As yet another application of the Markov inequality, let z be a nonnegative random variable and t and s be positive constants. Since $z \geq t$ if and only if $e^{sz} \geq e^{st}$, applying (A.1) to random variable e^{sz} yields

$$P(z \geq t) = P\left(e^{sz} \geq e^{st}\right) \leq e^{-st}E(e^{sz}) \tag{A.4}$$

Step 2 *Chernoff bound*

Now consider random variable $\sum_{n=1}^{N}\left(e(h(\boldsymbol{x}_n), y_n) - E_{\text{out}}(h)\right)$ and apply (A.4), we obtain

$$\begin{aligned} P\left(\sum_{n=1}^{N}\left(e(h(\boldsymbol{x}_n), y_n) - E_{\text{out}}(h)\right) \geq t\right) &\leq e^{-st}E\left[e^{s\sum_{n=1}^{N}\left(e(h(\boldsymbol{x}_n), y_n) - E_{\text{out}}(h)\right)}\right] \\ &= e^{-st}\prod_{n=1}^{N}E\left[e^{s\left(e(h(\boldsymbol{x}_n), y_n) - E_{\text{out}}(h)\right)}\right] \end{aligned} \tag{A.5}$$

The last equality is due to the fact that in statistical learning it is often assumed that the training data $\mathcal{D} = \{(\boldsymbol{x}_n, y_n), n = 1, 2, ..., N\}$ are i.i.d., hence $e(h(\boldsymbol{x}_n), y_n)$ are i.i.d. random variables.

Step 3 *Hoeffding inequality*

A general form of Hoeffding's inequality can be stated as follows: Let $u_1, u_2, \ldots, u_N$ be independent and bounded random variables that satisfy $u_n \in [a_n, b_n]$ with probability 1. Let

$\hat{z} = \sum_{n=1}^{N} u_n$. Then for any $t > 0$, we have

$$P\left(|\hat{z} - E[\hat{z}]| \geq t\right) \leq 2e^{\frac{-2t^2}{\sum_{i=1}^{n}(b_i - a_i)^2}} \tag{A.6}$$

Proof: A critical component of the proof is to show that for a random variable z with $E[z] = 0$ and $a \leq z \leq b$, $E\left[e^{sz}\right]$ has an upper bound

$$E\left[e^{sz}\right] \leq e^{\frac{s^2(b-a)^2}{8}} \tag{A.7}$$

To show (A.7), we use convexity of e^{sz} to write

$$e^{sz} \leq \frac{z-a}{b-a}e^{sb} + \frac{b-z}{b-a}e^{sa} \quad \text{for } a \leq z \leq b$$

It follows that

$$\begin{aligned} E\left[e^{sz}\right] &\leq E\left[\frac{z-a}{b-a}\right]e^{sb} + E\left[\frac{b-z}{b-a}\right]e^{sa} \\ &= \frac{b}{b-a}e^{sa} - \frac{a}{b-a}e^{sb} \\ &= \left(1 - \theta + \theta e^{s(b-a)}\right)e^{-\theta s(b-a)} \quad \text{with } \theta = -\frac{a}{b-a} \end{aligned} \tag{A.8}$$

By defining $\phi(v) = -\theta v + \ln(1 - \theta + \theta e^{v})$ with $v = s(b-a)$, (A.8) becomes

$$E\left[e^{sz}\right] \leq e^{\phi(v)} \tag{A.9}$$

Using calculus it can be shown that

$$\phi(v) \leq \frac{s^2(b-a)^2}{8}$$

which in conjunction with (A.8) yields (A.7). It follows that

$$P\left(\hat{z} - E[\hat{z}] \geq t\right) \leq e^{-st}\prod_{n=1}^{N} E\left[e^{s(u_n - E(u_n))}\right] \leq e^{-st}e^{s^2\sum_{n=1}^{N}\frac{(b_n - a_n)^2}{8}}$$

By choosing

$$s = \frac{4t}{\sum_{n=1}^{N}(b_n - a_n)^2}$$

we obtain

$$P\left(\hat{z} - E[\hat{z}] \geq t\right) \leq e^{\frac{-2t^2}{\sum_{i=1}^{n}(b_i - a_i)^2}}$$

Similarly, we can show

$$P\left(E[\hat{z}] - \hat{z} \geq t\right) \leq e^{\frac{-2t^2}{\sum_{i=1}^{n}(b_i - a_i)^2}}$$

combining the last two inequalities completes the proof. By applying the Hoeffding inequality with

$$u_n = e(h(\boldsymbol{x}_n), y_n) - E_{\text{out}}(h)$$

we obtain

$$P\left(|E_{\text{in}}(h) - E_{\text{out}}(h)| \geq \varepsilon\right) = P\left(\tfrac{1}{N}|\hat{z} - E(\hat{z})| \geq \varepsilon\right) \leq 2e^{-2\varepsilon^2 N}$$

which is (2.21). ■

Problems

2.1 [8] A sample of heads and tails is created by tossing a coin a number of times. Suppose we have a number of coins that generate different samples independently. For a given coin, let the probability of heads (we call it probability of error) be μ. The probability of obtaining *k* heads in *N* tosses of this coin is known to obey the binomial distribution given by

$$P[k \mid N, \mu] = \binom{N}{k} \mu^k (1-\mu)^{N-k}$$

where

$$\binom{N}{k} = \frac{N!}{k!(N-k)!}$$

Remember the training error is $\nu = \frac{k}{N}$.

(i) Assume the sample size is N = 10. If all coins have the same $\mu = 0.05$, compute the probability that at least one coin will have $\nu = 0$ for the case of 1 coin, 1000 coins, and 1,000,000 coins. Repeat the problem for $\mu = 0.8$.

(ii) For the case of N = 6 and 2 coins with $\mu = 0.5$ for both coins, plot the probability $P[\max_{i=1,2} |\nu_i - \mu| > \varepsilon]$ for ε in the range [0, 1]. On the same plot, depict the bound that would be obtained using the Hoeffding inequality. Remember that for a single coin, the Hoeffding bound is given by

$$P[|\nu - \mu| > \varepsilon] \le 2e^{-2N\varepsilon^2}$$

(Hint: To evaluate *P*[max…], use *P*[*A* or *B*] = *P*[*A*] + *P*[*B*] – *P*[*A* and *B*] = *P*[*A*] + *P*[*B*] – *P*[*A*]*P*[*B*] if *A* and *B* are independent events.)

2.2 [8] In this problem we derive a form of the law of large numbers, known as the *Chebyshev bound*, which is a bound that decreases linearly in *N*.

(i) Let *z* be a nonnegative random variable and *t* be a positive constant. Show that

$$P[z \ge t] \le \frac{E[z]}{t}$$

(ii) Let *u* be a random variable with mean μ and variance σ^2, show that for any $t > 0$

$$P[(u-\mu)^2 \ge t] \le \frac{\sigma^2}{t}$$

(iii) Let $u_1, u_2, \ldots, u_N$ be i.i.d. random variables, each with mean μ and variance σ^2, and $u = \frac{1}{N}\sum_{n=1}^{N} u_n$. Show that

$$P[(u-\mu)^2 \ge t] \le \frac{\sigma^2}{N \cdot t}$$

2.3 [8] In this problem we derive a form of the law of large numbers which is an *exponential* bound known as the *Chernoff bound*.

(i) Let *z* be a random variable, *t* be a positive constant, and *s* be a positive parameter. Show that

$$P[z \geq t] \leq e^{-st} E_z[e^{sz}]$$

(Hint: Use the result from part (i) of Prob. 2.4 and the fact that e^{sz} is monotonically increasing in z.)

(ii) Let $u_1, u_2, \ldots, u_N$ be i.i.d. random variables and $u = \frac{1}{N}\sum_{n=1}^{N} u_n$. Show that for any n

$$P[u \geq t] \leq \left(e^{-st} E_{u_n}[e^{su_n}]\right)^N$$

(iii) Suppose $P[u_n = 0] = P[u_n = 1] = \frac{1}{2}$. Evaluate $E_{u_n}[e^{su_n}]$ as a function of s and minimize $e^{-st} E_{u_n}[e^{su_n}]$ with respect to s for fixed t with $0 < t < 1$.

(iv) Based on the results from part (iii), show that for $0 < \varepsilon < \frac{1}{2}$

$$P[u \geq E(u) + \varepsilon] \leq 2^{-\beta N}$$

where $0 < \varepsilon < \frac{1}{2}\ \beta = 1 + (\frac{1}{2} + \varepsilon)\log_2(\frac{1}{2} + \varepsilon) + (\frac{1}{2} - \varepsilon)\log_2(\frac{1}{2} - \varepsilon)$ and $E(u) = \frac{1}{2}$.

2.4 [8] In this problem you are asked to create a binary-valued target function and a data set $\mathcal{D}$. The input space is $\mathcal{X} = R^2$.

(i) Choose a random line in the plane as your target function. Describe it by equation $w_0 + w_1 x_1 + w_2 x_2 = 0$. Report the numerical values of w_i for $i = 0, 1, 2$.

(ii) Generate a random data set of size $N = 20$. To ensure a meaningful problem for perceptron learning, make sure the data points appear on both sides of the straight line created in part (i). Map each data point on one side of the line to +1, and those on the other side of the line to –1. In this way, you have a training data $\{(\boldsymbol{x}_n, y_n), n = 1, 2, \ldots, 20\}$.

(iii) Apply the PLA to the data set. Report the numerical values of the initial weight vector $\boldsymbol{w}(0)$ and the algorithm's convergence in terms of number of updates. Report the numerical values of the final hypothesis obtained from PLA. Comment on its performance in terms of how well it separates the data set (for this purpose, it is best to include a plot) and how close it is to the true target function.

2.5 [8]

(i) Repeat Problem 2.4 using another randomly generated data set of size $N = 100$.

(ii) Repeat Problem 2.4 using another randomly generated data set of size $N = 1000$.

(iii) Comment on the results obtained from parts (i) and (ii) with respect to solution accuracy and running time..

2.6 [8] Repeat Problem 2.4 using the Adaline algorithm.

2.7 [8] Suppose one was asked to measure the length of the longer side of a cell phone, a total of N times, that yield N data points. The measurements are then indexed in an ascending manner as $y_1 \leq y_2 \leq \cdots \leq y_N$. Now one tires to run a representative value of the length, which in the language of machine learning means a hypothesis set $\mathcal{H}$ that contains all real-valued constants.

(i) Show that the hypothesis $h \in \mathcal{H}$ (i.e. a constant) that minimizes the L_2 in-sample error (which is the sum of squared deviations)

$$E_{in}(h) = \sum_{n=1}^{N} (h - y_n)^2$$

is the mean value of the measurements

$$h_{\text{mean}} = \frac{1}{N}\sum_{n=1}^{N} y_n$$

(ii) Find a hypothesis $h \in \mathcal{H}$ that minimizes the L_1 in-sample error (which is the sum of absolute deviations)

$$E_{in}(h) = \sum_{n=1}^{N} | h - y_n |$$

Show that the solution is the in-sample *median*, denoted by h_{med}, which is any value for which half of the data points are no at most h_{med} and half of the data points are at least h_{med}.

(iii) If data point y_N is perturbed to $y_N + \varepsilon$ where ε becomes increasingly large so that data point y_N becomes an *outlier*. Under the circumstances, what will happen to the two solutions h_{mean} and h_{med}?

2.8 [8] Suppose $m_{\mathcal{H}}(N) = N + 1$ so $d_{\text{vc}} = 1$ and you have $N = 100$ training examples. Use the VC bound in (2.26) to reduce a bound for E_{out} with confidence 90% (i.e., $\delta = 0.1$). Repeat the problem with $N = 10{,}000$.

2.9 [8] It can be shown that

$$P\left[\frac{E_{\text{out}}(g) - E_{in}(g)}{\sqrt{E_{\text{out}}(g)}} > \varepsilon\right] \le c \cdot m_{\mathcal{H}}(2N) \cdot e^{\frac{-\varepsilon^2 N}{4}} \qquad \text{(P2.1)}$$

where c is a constant slightly greater than 6. The above bound is useful when one cares about *relative* generalization error.

(i) Use (P2.1) to show that with probability $1 - \delta$ the generalization bound

$$E_{\text{out}}(g) \le E_{\text{in}}(g) + \frac{\xi}{2}\left(1 + \sqrt{1 + \frac{4E_{\text{in}}(g)}{\xi}}\right)$$

holds, where $\xi = \frac{4}{N}\ln\frac{c \cdot m_{\mathcal{H}}(2N)}{\delta}$.

(ii) Show that $E_{\text{out}}(g) \to E_{\text{in}}(g)$ as $N \to \infty$. Hint: Show that

$$\frac{\xi}{2}\left(1 + \sqrt{1 + \frac{4E_{\text{in}}(g)}{\xi}}\right) \to 0 \text{ as } N \to \infty$$

2.10 [8] When there is noise in the data, $E_{\text{out}}(g^{(\mathcal{D})}) = E_{\boldsymbol{x},y}\left[\left(g^{(\mathcal{D})}(\boldsymbol{x}) - y(\boldsymbol{x})\right)^2\right]$ where $y(\boldsymbol{x}) = f(\boldsymbol{x}) + \varepsilon$. Suppose we model the noise ε as a random variable with mean zero and variance σ^2, show that the bias-variance decomposition becomes

$$E_{\mathcal{D}}\left[E_{\text{out}}(g^{(\mathcal{D})})\right] = \text{ bias} + \text{var} + \sigma^2$$

2.11 [8] Consider a simple learning problem where the input space is one-dimensional; the input variable a is uniformly distributed in [–1, 1]; the data set consists of 2 points $\{x_1, x_2\}$; and the target function is $f(x) = x^2$ thus the full data set is $\mathcal{D} = \{(x_1, x_1^2), (x_2, x_2^2)\}$. The hypothesis set $\mathcal{H}$ consists of functions of the form $h(x) = ax + b$, and the learning algorithm returns with a straight

line as g that tries to fit these two data points. We are interested in the test performance E_{out} of the learning algorithm with respect to squared error measure, bias, and variance.

(i) Derive the analytic expression for the average function $\bar{g}(x)$.

(ii) Describe an experiment that you would run to numerically determine the $\bar{g}(x)$, E_{out}, bias, and variance.

(iii) Run the experiment and report you results. Compare E_{out} with bias + variance. Provide a plot of $\bar{g}(x)$ and $f(x)$ (in the same plot).

(iv) Compute analytically what E_{out}, bias, and variance should be.