Chapter 1

The Learning Problem

If you show a picture to a three-year-old and ask if there is a tree in it, you will likely get the correct answer. If you ask a thirty-year-old what the definition of a tree is, you will likely get an inconclusive answer. We didn't learn what a tree is by studying the mathematical definition of trees. We learned it by looking at trees. In other words, we learned from 'data'.

Learning from data is used in situations where we don't have an analytic solution, but we do have data that we can use to construct an empirical solution. This premise covers a lot of territory, and indeed learning from data is one of the most widely used techniques in science, engineering, and economics, among other fields.

In this chapter, we present examples of learning from data and formalize the learning problem. We also discuss the main concepts associated with learning, and the different paradigms of learning that have been developed.

1.1 Problem Setup

What do financial forecasting, medical diagnosis, computer vision, and search engines have in common? They all have successfully utilized learning from data. The repertoire of such applications is quite impressive. Let us open the discussion with a real-life application to see how learning from data works.

Consider the problem of predicting how a movie viewer would rate the various movies out there. This is an important problem if you are a company that rents out movies, since you want to recommend to different viewers the movies they will like. Good recommender systems are so important to business that the movie rental company Netflix offered a prize of one million dollars to anyone who could improve their recommendations by a mere 10%.

The main difficulty in this problem is that the criteria that viewers use to rate movies are quite complex. Trying to model those explicitly is no easy task, so it may not be possible to come up with an analytic solution. However, we

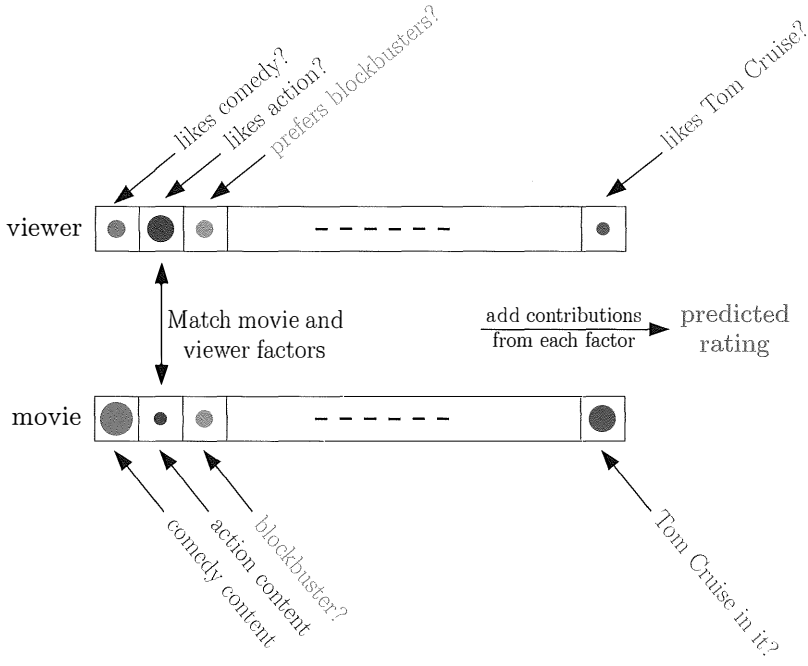


Figure 1.1: A model for how a viewer rates a movie

know that the historical rating data reveal a lot about how people rate movies, so we may be able to construct a good empirical solution. There is a great deal of data available to movie rental companies, since they often ask their viewers to rate the movies that they have already seen.

Figure 1.1 illustrates a specific approach that was widely used in the million-dollar competition. Here is how it works. You describe a movie as a long array of different factors, e.g., how much comedy is in it, how complicated is the plot, how handsome is the lead actor, etc. Now, you describe each viewer with corresponding factors; how much do they like comedy, do they prefer simple or complicated plots, how important are the looks of the lead actor, and so on. How this viewer will rate that movie is now estimated based on the match/mismatch of these factors. For example, if the movie is pure comedy and the viewer hates comedies, the chances are he won't like it. If you take dozens of these factors describing many facets of a movie's content and a viewer's taste, the conclusion based on matching all the factors will be a good predictor of how the viewer will rate the movie.

The power of learning from data is that this entire process can be automated, without any need for analyzing movie content or viewer taste. To do so, the learning algorithm 'reverse-engineers' these factors based solely on pre-

vious ratings. It starts with random factors, then tunes these factors to make them more and more aligned with how viewers have rated movies before, until they are ultimately able to *predict* how viewers rate movies in general. The factors we end up with may not be as intuitive as 'comedy content', and in fact can be quite subtle or even incomprehensible. After all, the algorithm is only trying to find the best way to predict how a viewer would rate a movie, not necessarily explain to us how it is done. This algorithm was part of the winning solution in the million-dollar competition.

1.1.1 Components of Learning

The movie rating application captures the essence of learning from data, and so do many other applications from vastly different fields. In order to abstract the common core of the learning problem, we will pick one application and use it as a metaphor for the different components of the problem. Let us take credit approval as our metaphor.

Suppose that a bank receives thousands of credit card applications every day, and it wants to automate the process of evaluating them. Just as in the case of movie ratings, the bank knows of no magical formula that can pinpoint when credit should be approved, but it has a lot of data. This calls for learning from data, so the bank uses historical records of previous customers to figure out a good formula for credit approval.

Each customer record has personal information related to credit, such as annual salary, years in residence, outstanding loans, etc. The record also keeps track of whether approving credit for that customer was a good idea, i.e., did the bank make money on that customer. This data guides the construction of a successful formula for credit approval that can be used on future applicants.

Let us give names and symbols to the main components of this learning problem. There is the input $\mathbf{x}$ (customer information that is used to make a credit decision), the unknown target function $f: \mathcal{X} \to \mathcal{Y}$ (ideal formula for credit approval), where $\mathcal{X}$ is the input space (set of all possible inputs $\mathbf{x}$), and $\mathcal{Y}$ is the output space (set of all possible outputs, in this case just a yes/no decision). There is a data set $\mathcal{D}$ of input-output examples $(\mathbf{x}_1, y_1), \cdots, (\mathbf{x}_N, y_N)$, where $y_n = f(\mathbf{x}_n)$ for $n = 1, ..., N$ (inputs corresponding to previous customers and the correct credit decision for them in hindsight). The examples are often referred to as data points. Finally, there is the learning algorithm that uses the data set $\mathcal{D}$ to pick a formula $g: \mathcal{X} \to \mathcal{Y}$ that approximates f. The algorithm chooses g from a set of candidate formulas under consideration, which we call the hypothesis set $\mathcal{H}$. For instance, $\mathcal{H}$ could be the set of all linear formulas from which the algorithm would choose the best linear fit to the data, as we will introduce later in this section.

When a new customer applies for credit, the bank will base its decision on g (the hypothesis that the learning algorithm produced), not on f (the ideal target function which remains unknown). The decision will be good only to the extent that g faithfully replicates f. To achieve that, the algorithm

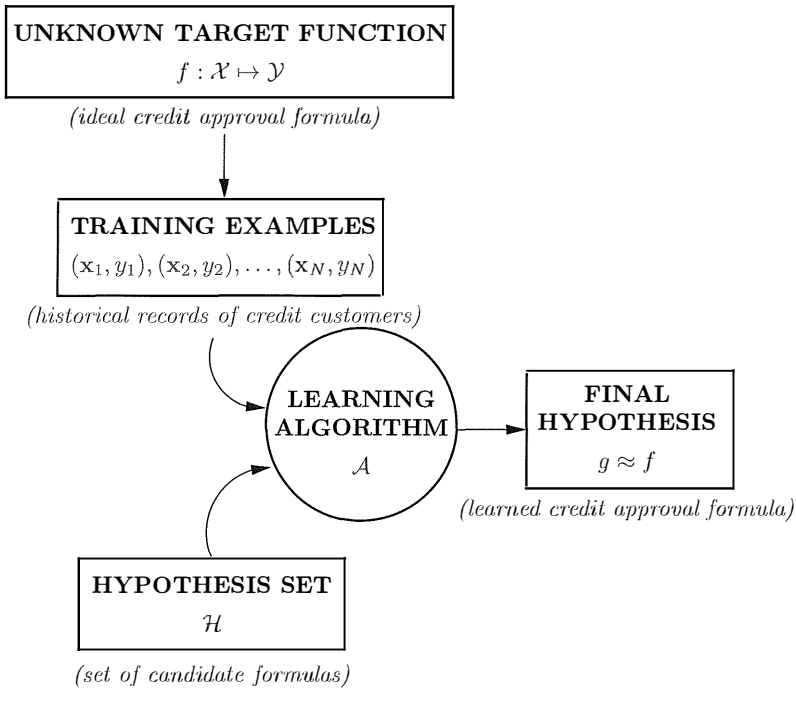


Figure 1.2: Basic setup of the learning problem

chooses g that best matches f on the *training* examples of previous customers, with the hope that it will continue to match f on new customers. Whether or not this hope is justified remains to be seen. Figure 1.2 illustrates the components of the learning problem.

Exercise 1.1

Express each of the following tasks in the framework of learning from data by specifying the input space $\mathcal{X}$, output space $\mathcal{Y}$, target function $f : \mathcal{X} \to \mathcal{Y}$, and the specifics of the data set that we will learn from.

(a) Medical diagnosis: A patient walks in with a medical history and some symptoms, and you want to identify the problem.

(b) Handwritten digit recognition (for example postal zip code recognition for mail sorting).

(c) Determining if an email is spam or not.

(d) Predicting how an electric load varies with price, temperature, and day of the week.

(e) A problem of interest to you for which there is no analytic solution, but you have data from which to construct an empirical solution.

We will use the setup in Figure 1.2 as our definition of the learning problem. Later on, we will consider a number of refinements and variations to this basic setup as needed. However, the essence of the problem will remain the same. There is a target to be learned. It is unknown to us. We have a set of examples generated by the target. The learning algorithm uses these examples to look for a hypothesis that approximates the target.

1.1.2 A Simple Learning Model

Let us consider the different components of Figure 1.2. Given a specific learning problem, the target function and training examples are dictated by the problem. However, the learning algorithm and hypothesis set are not. These are solution tools that we get to choose. The hypothesis set and learning algorithm are referred to informally as the *learning model.*

Here is a simple model. Let $\mathcal{X} = \mathbb{R}^d$ be the input space, where $\mathbb{R}^d$ is the d-dimensional Euclidean space, and let $\mathcal{Y} = \{+1, -1\}$ be the output space, denoting a binary (yes/no) decision. In our credit example, different coordinates of the input vector $\mathbf{x} \in \mathbb{R}^d$ correspond to salary, years in residence, outstanding debt, and the other data fields in a credit application. The binary output y corresponds to approving or denying credit. We specify the hypothesis set $\mathcal{H}$ through a functional form that all the hypotheses $h \in \mathcal{H}$ share. The functional form $h(\mathbf{x})$ that we choose here gives different weights to the different coordinates of $\mathbf{x}$, reflecting their relative importance in the credit decision. The weighted coordinates are then combined to form a 'credit score' and the result is compared to a threshold value. If the applicant passes the threshold, credit is approved; if not, credit is denied:

$$\text{Approve credit if} \quad \sum_{i=1}^{d} w_i x_i > \text{threshold},$$

$$\text{Deny credit if} \quad \sum_{i=1}^{d} w_i x_i < \text{threshold}.$$

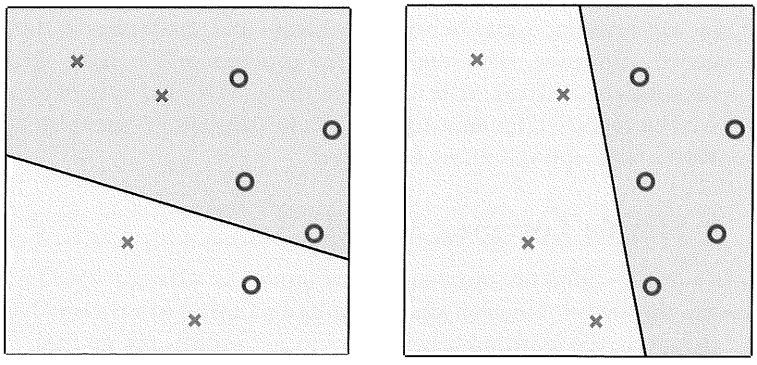
This formula can be written more compactly as

$$h(\mathbf{x}) = \text{sign}\left(\left(\sum_{i=1}^{d} w_i x_i\right) + b\right), \tag{1.1}$$

where $x_1, \cdots, x_d$ are the components of the vector $\mathbf{x}$; $h(\mathbf{x}) = +1$ means 'approve credit' and $h(\mathbf{x}) = -1$ means 'deny credit'; $\text{sign}(s) = +1$ if $s > 0$ and $\text{sign}(s) = -1$ if $s < 0$.[1] The weights are $w_1, \cdots, w_d$, and the threshold is determined by the bias term b since in Equation (1.1), credit is approved if $\sum_{i=1}^{d} w_i x_i > -b$.

This model of $\mathcal{H}$ is called the *perceptron*, a name that it got in the context of artificial intelligence. The learning algorithm will search $\mathcal{H}$ by looking for

[1] The value of $\text{sign}(s)$ when s 0 is a simple technicality that we ignore for the moment.



(a) Misclassified data (b) Perfectly classified data

Figure 1.3: Perceptron classification of linearly separable data in a two dimensional input space (a) Some training examples will be misclassified (blue points in red region and vice versa) for certain values of the weight parameters which define the separating line. (b) A final hypothesis that classifies all training examples correctly. ($\circ$ is $+1$ and $\times$ is -1.)

weights and bias that perform well on the data set. Some of the weights $w_1, \cdots, w_d$ may end up being negative, corresponding to an adverse effect on credit approval. For instance, the weight of the 'outstanding debt' field should come out negative since more debt is not good for credit. The bias value b may end up being large or small, reflecting how lenient or stringent the bank should be in extending credit. The optimal choices of weights and bias define the final hypothesis $g \in \mathcal{H}$ that the algorithm produces.

Exercise 1.2

Suppose that we use a perceptron to detect spam messages. Let's say that each email message is represented by the frequency of occurrence of keywords, and the output is $+1$ if the message is considered spam.

(a) Can you think of some keywords that will end up with a large positive weight in the perceptron?

(b) How about keywords that will get a negative weight?

(c) What parameter in the perceptron directly affects how many borderline messages end up being classified as spam?

Figure 1.3 illustrates what a perceptron does in a two-dimensional case ($d = 2$). The plane is split by a line into two regions, the $+1$ decision region and the -1 decision region. Different values for the parameters w_1, w_2, b correspond to different lines $w_1 x_1 + w_2 x_2 + b = 0$. If the data set is *linearly separable*, there will be a choice for these parameters that classifies all the training examples correctly.

To simplify the notation of the perceptron formula, we will treat the bias b as a weight $w_0 = b$ and merge it with the other weights into one vector $\mathbf{w} = [w_0, w_1, \cdots, w_d]^{\mathrm{T}}$, where $^{\mathrm{T}}$ denotes the transpose of a vector, so $\mathbf{w}$ is a column vector. We also treat $\mathbf{x}$ as a column vector and modify it to become $\mathbf{x} = [x_0, x_1, \cdots, x_d]^{\mathrm{T}}$, where the added coordinate x_0 is fixed at $x_0 = 1$. Formally speaking, the input space is now

$$\mathcal{X} = \{1\} \times \mathbb{R}^d = \{[x_0, x_1, \cdots, x_d]^{\mathrm{T}} \mid x_0 = 1, x_1 \in \mathbb{R}, \cdots, x_d \in \mathbb{R}\}.$$

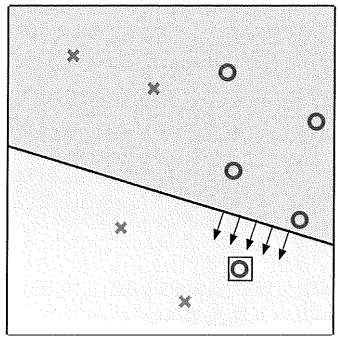
With this convention, $\mathbf{w}^{\mathrm{T}}\mathbf{x} = \sum_{i=0}^{d} w_i x_i$, and so Equation (1.1) can be rewritten in vector form as

$$h(\mathbf{x}) = \mathrm{sign}(\mathbf{w}^{\mathrm{T}}\mathbf{x}). \tag{1.2}$$

We now introduce the *perceptron learning algorithm* (PLA). The algorithm will determine what $\mathbf{w}$ should be, based on the data. Let us assume that the data set is linearly separable, which means that there is a vector $\mathbf{w}$ that makes (1.2) achieve the correct decision $h(\mathbf{x}_n) = y_n$ on all the training examples, as shown in Figure 1.3.

Our learning algorithm will find this $\mathbf{w}$ using a simple iterative method. Here is how it works. At iteration t, where $t = 0, 1, 2, \ldots$, there is a current value of the weight vector, call it $\mathbf{w}(t)$. The algorithm picks an example from $(\mathbf{x}_1, y_1) \cdots (\mathbf{x}_N, y_N)$ that is currently misclassified, call it $(\mathbf{x}(t), y(t))$, and uses it to update $\mathbf{w}(t)$. Since the example is misclassified, we have $y(t) \neq \mathrm{sign}(\mathbf{w}^{\mathrm{T}}(t)\mathbf{x}(t))$. The update rule is

$$\mathbf{w}(t+1) = \mathbf{w}(t) + y(t)\mathbf{x}(t). \tag{1.3}$$



This rule moves the boundary in the direction of classifying $\mathbf{x}(t)$ correctly, as depicted in the figure above. The algorithm continues with further iterations until there are no longer misclassified examples in the data set.

Exercise 1.3

The weight update rule in (1.3) has the nice interpretation that it moves in the direction of classifying $\mathbf{x}(t)$ correctly.

(a) Show that $y(t)\mathbf{w}^{\mathrm{T}}(t)\mathbf{x}(t) < 0$. *[Hint: $\mathbf{x}(t)$ is misclassified by $\mathbf{w}(t)$.]*

(b) Show that $y(t)\mathbf{w}^{\mathrm{T}}(t+1)\mathbf{x}(t) > y(t)\mathbf{w}^{\mathrm{T}}(t)\mathbf{x}(t)$. *[Hint: Use (1.3).]*

(c) As far as classifying $\mathbf{x}(t)$ is concerned, argue that the move from $\mathbf{w}(t)$ to $\mathbf{w}(t+1)$ is a move 'in the right direction'.

Although the update rule in (1.3) considers only one training example at a time and may 'mess up' the classification of the other examples that are not involved in the current iteration, it turns out that the algorithm is guaranteed to arrive at the right solution in the end. The proof is the subject of Problem 1.3. The result holds regardless of which example we choose from among the misclassified examples in $(\mathbf{x}_1, y_1) \cdots (\mathbf{x}_N, y_N)$ at each iteration, and regardless of how we initialize the weight vector to start the algorithm. For simplicity, we can pick one of the misclassified examples at random (or cycle through the examples and always choose the first misclassified one), and we can initialize $\mathbf{w}(0)$ to the zero vector.

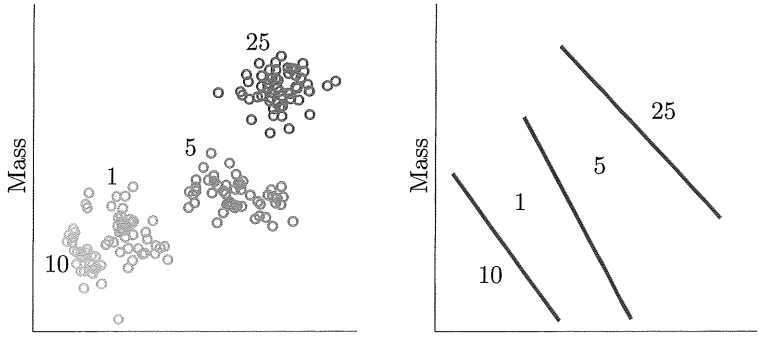
Within the infinite space of all weight vectors, the perceptron algorithm manages to find a weight vector that works, using a simple iterative process. This illustrates how a learning algorithm can effectively search an infinite hypothesis set using a finite number of simple steps. This feature is characteristic of many techniques that are used in learning, some of which are far more sophisticated than the perceptron learning algorithm.

Exercise 1.4

Let us create our own target function f and data set $\mathcal{D}$ and see how the perceptron learning algorithm works. Take $d = 2$ so you can visualize the problem, and choose a random line in the plane as your target function, where one side of the line maps to $+1$ and the other maps to -1. Choose the inputs $\mathbf{x}_n$ of the data set as random points in the plane, and evaluate the target function on each $\mathbf{x}_n$ to get the corresponding output y_n.

Now, generate a data set of size 20. Try the perceptron learning algorithm on your data set and see how long it takes to converge and how well the final hypothesis g matches your target f. You can find other ways to play with this experiment in Problem 1.4.

The perceptron learning algorithm succeeds in achieving its goal; finding a hypothesis that classifies all the points in the data set $\mathcal{D} = \{(\mathbf{x}_1, y_1) \cdots (\mathbf{x}_N, y_N)\}$ correctly. Does this mean that this hypothesis will also be successful in classifying new data points that are not in $\mathcal{D}$? This turns out to be the key question in the theory of learning, a question that will be thoroughly examined in this book.



(a) Coin data (b) Learned classifier

Figure 1.4: The learning approach to coin classification (a) Training data of pennies, nickels, dimes, and quarters (1, 5, 10, and 25 cents) are represented in a size mass space where they fall into clusters. (b) A classification rule is learned from the data set by separating the four clusters. A new coin will be classified according to the region in the size mass plane that it falls into.

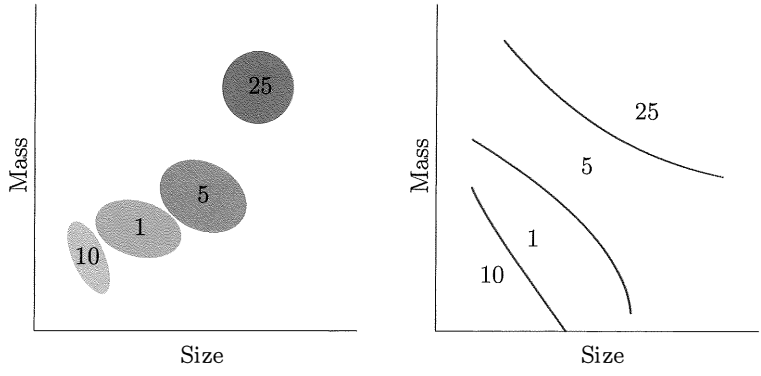
1.1.3 Learning versus Design

So far, we have discussed what learning is. Now, we discuss what it is not. The goal is to distinguish between learning and a related approach that is used for similar problems. While learning is based on data, this other approach does not use data. It is a 'design' approach based on specifications, and is often discussed alongside the learning approach in pattern recognition literature.

Consider the problem of recognizing coins of different denominations, which is relevant to vending machines, for example. We want the machine to recognize quarters, dimes, nickels and pennies. We will contrast the 'learning from data' approach and the 'design from specifications' approach for this problem. We assume that each coin will be represented by its size and mass, a two-dimensional input.

In the learning approach, we are given a sample of coins from each of the four denominations and we use these coins as our data set. We treat the size and mass as the input vector, and the denomination as the output. Figure 1.4(a) shows what the data set may look like in the input space. There is some variation of size and mass within each class, but by and large coins of the same denomination cluster together. The learning algorithm searches for a hypothesis that classifies the data set well. If we want to classify a new coin, the machine measures its size and mass, and then classifies it according to the learned hypothesis in Figure 1.4(b).

In the design approach, we call the United States Mint and ask them about the specifications of different coins. We also ask them about the number



(a) Probabilistic model of data (b) Inferred classifier

Figure 1.5: The design approach to coin classification (a) A probabilistic model for the size, mass, and denomination of coins is derived from known specifications. The figure shows the high probability region for each denom ination (1, 5, 10, and 25 cents) according to the model. (b) A classification rule is derived analytically to minimize the probability of error in classifying a coin based on size and mass. The resulting regions for each denomination are shown.

of coins of each denomination in circulation, in order to get an estimate of the relative frequency of each coin. Finally, we make a physical model of the variations in size and mass due to exposure to the elements and due to errors in measurement. We put all of this information together and compute the full joint probability distribution of size, mass, and coin denomination (Figure 1.5(a)). Once we have that joint distribution, we can construct the optimal decision rule to classify coins based on size and mass (Figure 1.5(b)). The rule chooses the denomination that has the highest probability for a given size and mass, thus achieving the smallest possible probability of error.[2]

The main difference between the learning approach and the design approach is the role that data plays. In the design approach, the problem is well specified and one can analytically derive f without the need to see any data. In the learning approach, the problem is much less specified, and one needs data to pin down what f is.

Both approaches may be viable in some applications, but only the learning approach is possible in many applications where the target function is unknown. We are not trying to compare the utility or the performance of the two approaches. We are just making the point that the design approach is distinct from learning. This book is about learning.

[2]This is called Bayes optimal decision theory. Some learning models are based on the same theory by estimating the probability from data.

Exercise 1.5

Which of the following problems are more suited for the learning approach and which are more suited for the design approach?

(a) Determining the age at which a particular medical test should be performed

(b) Classifying numbers into primes and non-primes

(c) Detecting potential fraud in credit card charges

(d) Determining the time it would take a falling object to hit the ground

(e) Determining the optimal cycle for traffic lights in a busy intersection

1.2 Types of Learning

The basic premise of learning from data is the use of a set of observations to uncover an underlying process. It is a very broad premise, and difficult to fit into a single framework. As a result, different learning paradigms have arisen to deal with different situations and different assumptions. In this section, we introduce some of these paradigms.

The learning paradigm that we have discussed so far is called *supervised* learning. It is the most studied and most utilized type of learning, but it is not the only one. Some variations of supervised learning are simple enough to be accommodated within the same framework. Other variations are more profound and lead to new concepts and techniques that take on lives of their own. The most important variations have to do with the nature of the data set.

1.2.1 Supervised Learning

When the training data contains explicit examples of what the correct output should be for given inputs, then we are within the supervised learning setting that we have covered so far. Consider the hand-written digit recognition problem (task (b) of Exercise 1.1). A reasonable data set for this problem is a collection of images of hand-written digits, and for each image, what the digit actually is. We thus have a set of examples of the form (image , digit). The learning is supervised in the sense that some 'supervisor' has taken the trouble to look at each input, in this case an image, and determine the correct output, in this case one of the ten categories $\{0, 1, 2, 3, 4, 5, 6, 7, 8, 9\}$.

While we are on the subject of variations, there is more than one way that a data set can be presented to the learning process. Data sets are typically created and presented to us in their entirety at the outset of the learning process. For instance, historical records of customers in the credit-card application, and previous movie ratings of customers in the movie rating application, are already there for us to use. This protocol of a 'ready' data set is the most

common in practice, and it is what we will focus on in this book. However, it is worth noting that two variations of this protocol have attracted a significant body of work.

One is *active learning*, where the data set is acquired through queries that we make. Thus, we get to choose a point $\mathbf{x}$ in the input space, and the supervisor reports to us the target value for $\mathbf{x}$. As you can see, this opens the possibility for strategic choice of the point $\mathbf{x}$ to maximize its information value, similar to asking a strategic question in a game of 20 questions.

Another variation is called *online learning*, where the data set is given to the algorithm one example at a time. This happens when we have streaming data that the algorithm has to process 'on the run'. For instance, when the movie recommendation system discussed in Section 1.1 is deployed, online learning can process new ratings from current users and movies. Online learning is also useful when we have limitations on computing and storage that preclude us from processing the whole data as a batch. We should note that online learning can be used in different paradigms of learning, not just in supervised learning.

1.2.2 Reinforcement Learning

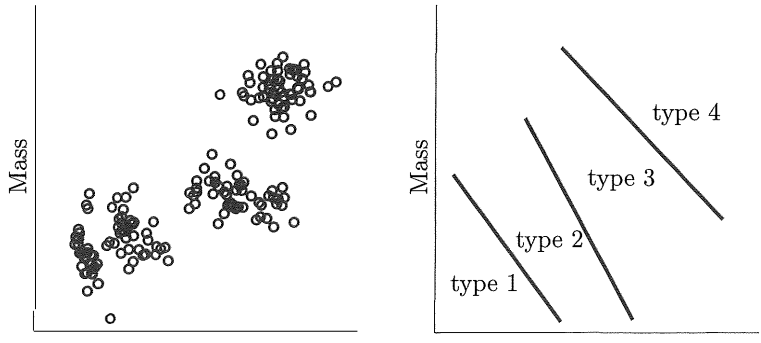
When the training data does not explicitly contain the correct output for each input, we are no longer in a supervised learning setting. Consider a toddler learning not to touch a hot cup of tea. The experience of such a toddler would typically comprise a set of occasions when the toddler confronted a hot cup of tea and was faced with the decision of touching it or not touching it. Presumably, every time she touched it, the result was a high level of pain, and every time she didn't touch it, a much lower level of pain resulted (that of an unsatisfied curiosity). Eventually, the toddler learns that she is better off not touching the hot cup.

The training examples did not spell out what the toddler should have done, but they instead graded different actions that she has taken. Nevertheless, she uses the examples to reinforce the better actions, eventually learning what she should do in similar situations. This characterizes *reinforcement* learning, where the training example does not contain the target output, but instead contains some possible output together with a measure of how good that output is. In contrast to supervised learning where the training examples were of the form (input , correct output), the examples in reinforcement learning are of the form

$$(\ \text{input}\ ,\ \text{some output}\ ,\ \text{grade for this output}\ ).$$

Importantly, the example does not say how good other outputs would have been for this particular input.

Reinforcement learning is especially useful for learning how to play a game. Imagine a situation in backgammon where you have a choice between different actions and you want to identify the best action. It is not a trivial task to ascertain what the best action is at a given stage of the game, so we cannot



(a) Unlabeled Coin data (b) Unsupervised learning

Figure 1.6: Unsupervised learning of coin classification (a) The same data set of coins in Figure 1.4(a) is again represented in the size mass space, but without being labeled. They still fall into clusters. (b) An unsupervised classification rule treats the four clusters as different types. The rule may be somewhat ambiguous, as type 1 and type 2 could be viewed as one cluster

easily create supervised learning examples. If you use reinforcement learning instead, all you need to do is to take some action and report how well things went, and you have a training example. The reinforcement learning algorithm is left with the task of sorting out the information coming from different examples to find the best line of play.

1.2.3 Unsupervised Learning

In the unsupervised setting, the training data does not contain any output information at all. We are just given input examples $\mathbf{x}_1, \cdots, \mathbf{x}_N$. You may wonder how we could possibly learn anything from mere inputs. Consider the coin classification problem that we discussed earlier in Figure 1.4. Suppose that we didn't know the denomination of any of the coins in the data set. This *unlabeled data* is shown in Figure 1.6(a). We still get similar clusters, but they are now unlabeled so all points have the same 'color'. The decision regions in unsupervised learning may be identical to those in supervised learning, but without the labels (Figure 1.6(b)). However, the correct clustering is less obvious now, and even the number of clusters may be ambiguous.

Nonetheless, this example shows that we can learn *something* from the inputs by themselves. Unsupervised learning can be viewed as the task of spontaneously finding patterns and structure in input data. For instance, if our task is to categorize a set of books into topics, and we only use general properties of the various books, we can identify books that have similar properties and put them together in one category, without naming that category.

Unsupervised learning can also be viewed as a way to create a higher-level representation of the data. Imagine that you don't speak a word of Spanish, but your company will relocate you to Spain next month. They will arrange for Spanish lessons once you are there, but you would like to prepare yourself a bit before you go. All you have access to is a Spanish radio station. For a full month, you continuously bombard yourself with Spanish; this is an unsupervised learning experience since you don't know the meaning of the words. However, you gradually develop a better representation of the language in your brain by becoming more tuned to its common sounds and structures. When you arrive in Spain, you will be in a better position to start your Spanish lessons. Indeed, unsupervised learning can be a precursor to supervised learning. In other cases, it is a stand-alone technique.

Exercise 1.6

For each of the following tasks, identify which type of learning is involved (supervised, reinforcement, or unsupervised) and the training data to be used. If a task can fit more than one type, explain how and describe the training data for each type.

(a) Recommending a book to a user in an online bookstore

(b) Playing tic tac toe

(c) Categorizing movies into different types

(d) Learning to play music

(e) Credit limit: Deciding the maximum allowed debt for each bank customer

Our main focus in this book will be supervised learning, which is the most popular form of learning from data.

1.2.4 Other Views of Learning

The study of learning has evolved somewhat independently in a number of fields that started historically at different times and in different domains, and these fields have developed different emphases and even different jargons. As a result, learning from data is a diverse subject with many aliases in the scientific literature. The main field dedicated to the subject is called *machine learning*, a name that distinguishes it from human learning. We briefly mention two other important fields that approach learning from data in their own ways.

Statistics shares the basic premise of learning from data, namely the use of a set of observations to uncover an underlying process. In this case, the process is a probability distribution and the observations are samples from that distribution. Because statistics is a mathematical field, emphasis is given to situations where most of the questions can be answered with rigorous proofs. As a result, statistics focuses on somewhat idealized models and analyzes them in great detail. This is the main difference between the statistical approach

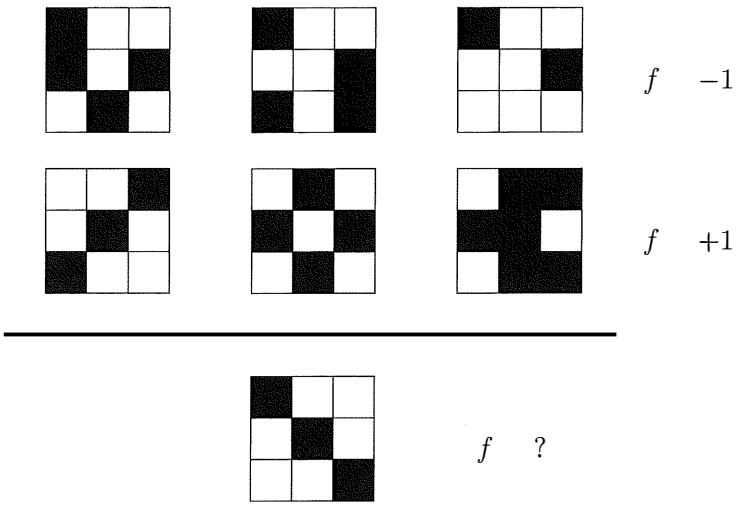


Figure 1.7: A visual learning problem. The first two rows show the training examples (each input $\mathbf{x}$ is a 9 bit vector represented visually as a 3×3 black and white array). The inputs in the first row have $f(\mathbf{x}) = -1$, and the inputs in the second row have $f(\mathbf{x}) = +1$. Your task is to learn from this data set what f is, then apply f to the test input at the bottom. Do you get -1 or $+1$?

to learning and how we approach the subject here. We make less restrictive assumptions and deal with more general models than in statistics. Therefore, we end up with weaker results that are nonetheless broadly applicable.

Data mining is a practical field that focuses on finding patterns, correlations, or anomalies in large relational databases. For example, we could be looking at medical records of patients and trying to detect a cause-effect relationship between a particular drug and long-term effects. We could also be looking at credit card spending patterns and trying to detect potential fraud. Technically, data mining is the same as learning from data, with more emphasis on data analysis than on prediction. Because databases are usually huge, computational issues are often critical in data mining. Recommender systems, which were illustrated in Section 1.1 with the movie rating example, are also considered part of data mining.

1.3 Is Learning Feasible?

The target function f is the object of learning. The most important assertion about the target function is that it is *unknown*. We really mean unknown.

This raises a natural question. How could a limited data set reveal enough information to pin down the entire target function? Figure 1.7 illustrates this

difficulty. A simple learning task with 6 training examples of a ± 1 target function is shown. Try to learn what the function is then apply it to the test input given. Do you get -1 or $+1$? Now, show the problem to your friends and see if they get the same answer.

The chances are the answers were not unanimous, and for good reason. There is simply more than one function that fits the 6 training examples, and some of these functions have a value of -1 on the test point and others have a value of $+1$. For instance, if the true f is $+1$ when the pattern is symmetric, the value for the test point would be $+1$. If the true f is $+1$ when the top left square of the pattern is white, the value for the test point would be -1. Both functions agree with all the examples in the data set, so there isn't enough information to tell us which would be the correct answer.

This does not bode well for the feasibility of learning. To make matters worse, we will now see that the difficulty we experienced in this simple problem is the rule, not the exception.

1.3.1 Outside the Data Set

When we get the training data $\mathcal{D}$, e.g., the first two rows of Figure 1.7, we know the value of f on all the points in $\mathcal{D}$. This doesn't mean that we have learned f, since it doesn't guarantee that we know anything about f outside of $\mathcal{D}$. We know what we have already seen, but that's not learning. That's memorizing.

Does the data set $\mathcal{D}$ tell us anything outside of $\mathcal{D}$ that we didn't know before? If the answer is yes, then we have learned *something*. If the answer is no, we can conclude that learning is not feasible.

Since we maintain that f is an unknown function, we can prove that f remains unknown outside of $\mathcal{D}$. Instead of going through a formal proof for the general case, we will illustrate the idea in a concrete case. Consider a Boolean target function over a three-dimensional input space $\mathcal{X} = \{0,1\}^3$. We are given a data set $\mathcal{D}$ of five examples represented in the table below. We denote the binary output by $\circ/\bullet$ for visual clarity,

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

where $y_n = f(\mathbf{x}_n)$ for $n = 1, 2, 3, 4, 5$. The advantage of this simple Boolean case is that we can enumerate the entire input space (since there are only $2^3 = 8$ distinct input vectors), and we can enumerate the set of all possible target functions (since f is a Boolean function on 3 Boolean inputs, and there are only $2^{2^3} = 256$ distinct Boolean functions on 3 Boolean inputs).

Let us look at the problem of learning f. Since f is unknown except inside $\mathcal{D}$, any function that agrees with $\mathcal{D}$ could conceivably be f. The table below shows all such functions $f_1, \cdots, f_8$. It also shows the data set $\mathcal{D}$ (in blue) and what the final hypothesis g may look like.

$\mathbf{x}$	y	g	f_1	f_2	f_3	f_4	f_5	f_6	f_7	f_8
0 0 0	○	○	○	○	○	○	○	○	○	○
0 0 1	●	●	●	●	●	●	●	●	●	●
0 1 0	●	●	●	●	●	●	●	●	●	●
0 1 1	○	○	○	○	○	○	○	○	○	○
1 0 0	●	●	●	●	●	●	●	●	●	●
1 0 1		?	○	○	○	○	●	●	●	●
1 1 0		?	○	○	●	●	○	○	●	●
1 1 1		?	○	●	○	●	○	●	○	●

The final hypothesis g is chosen based on the five examples in $\mathcal{D}$. The table shows the case where g is chosen to match f on these examples.

If we remain true to the notion of unknown target, we cannot exclude any of $f_1, \cdots, f_8$ from being the true f. Now, we have a dilemma. The whole purpose of learning f is to be able to predict the value of f on points that we haven't seen before. The quality of the learning will be determined by how close our prediction is to the true value. Regardless of what g predicts on the three points we haven't seen before (those outside of $\mathcal{D}$, denoted by red question marks), it can agree or disagree with the target, depending on which of $f_1, \cdots, f_8$ turns out to be the true target. It is easy to verify that any 3 bits that replace the red question marks are as good as any other 3 bits.

Exercise 1.7

For each of the following learning scenarios in the above problem, evaluate the performance of g on the three points in $\mathcal{X}$ outside $\mathcal{D}$. To measure the performance, compute how many of the 8 possible target functions agree with g on all three points, on two of them, on one of them, and on none of them.

(a) $\mathcal{H}$ has only two hypotheses, one that always returns '●' and one that always returns '○'. The learning algorithm picks the hypothesis that matches the data set the most.

(b) The same $\mathcal{H}$, but the learning algorithm now picks the hypothesis that matches the data set the *least*.

(c) $\mathcal{H} = \{\text{XOR}\}$ (only one hypothesis which is always picked), where XOR is defined by $\text{XOR}(\mathbf{x}) = \bullet$ if the number of 1's in $\mathbf{x}$ is odd and $\text{XOR}(\mathbf{x}) = \circ$ if the number is even.

(d) $\mathcal{H}$ contains all possible hypotheses (all Boolean functions on three variables), and the learning algorithm picks the hypothesis that agrees with all training examples, but otherwise disagrees the most with the XOR.

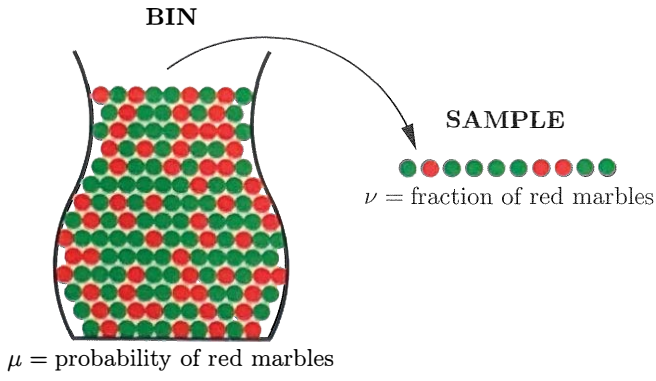


Figure 1.8: A random sample is picked from a bin of red and green marbles. The probability μ of red marbles in the bin is unknown. What does the fraction ν of red marbles in the sample tell us about μ?

It doesn't matter what the algorithm does or what hypothesis set $\mathcal{H}$ is used. Whether $\mathcal{H}$ has a hypothesis that perfectly agrees with $\mathcal{D}$ (as depicted in the table) or not, and whether the learning algorithm picks that hypothesis or picks another one that disagrees with $\mathcal{D}$ (different green bits), it makes no difference whatsoever as far as the performance outside of $\mathcal{D}$ is concerned. Yet the performance outside $\mathcal{D}$ is all that matters in learning!

This dilemma is not restricted to Boolean functions, but extends to the general learning problem. As long as f is an unknown function, knowing $\mathcal{D}$ cannot exclude any pattern of values for f outside of $\mathcal{D}$. Therefore, the predictions of g outside of $\mathcal{D}$ are meaningless.

Does this mean that learning from data is doomed? If so, this will be a very short book ☺. Fortunately, learning is alive and well, and we will see why. We won't have to change our basic assumption to do that. The target function will continue to be unknown, and we still mean *unknown*.

1.3.2 Probability to the Rescue

We will show that we can indeed infer something outside $\mathcal{D}$ using only $\mathcal{D}$, but in a probabilistic way. What we infer may not be much compared to learning a full target function, but it will establish the principle that we can reach outside $\mathcal{D}$. Once we establish that, we will take it to the general learning problem and pin down what we can and cannot learn.

Let's take the simplest case of picking a sample, and see when we can say something about the objects outside the sample. Consider a bin that contains red and green marbles, possibly infinitely many. The proportion of red and green marbles in the bin is such that if we pick a marble at random, the probability that it will be red is μ and the probability that it will be green is $1 - \mu$. We assume that the value of μ is unknown to us.

We pick a random sample of N independent marbles (with replacement) from this bin, and observe the fraction ν of red marbles within the sample (Figure 1.8). What does the value of ν tell us about the value of μ?

One answer is that regardless of the colors of the N marbles that we picked, we still don't know the color of any marble that we didn't pick. We can get mostly green marbles in the sample while the bin has mostly red marbles. Although this is certainly *possible*, it is by no means *probable*.

Exercise 1.8

If $\mu = 0.9$, what is the probability that a sample of 10 marbles will have $\nu \leq 0.1$? *[Hints: 1. Use binomial distribution. 2. The answer is a very small number.]*

The situation is similar to taking a poll. A random sample from a population tends to agree with the views of the population at large. The probability distribution of the random variable ν in terms of the parameter μ is well understood, and when the sample size is big, ν tends to be close to μ.

To quantify the relationship between ν and μ, we use a simple bound called the *Hoeffding Inequality*. It states that for any sample size N,

$$\mathbb{P}\left[|\nu - \mu| > \epsilon\right] \leq 2e^{-2\epsilon^2 N} \qquad \text{for any } \epsilon > 0. \tag{1.4}$$

Here, $\mathbb{P}[\cdot]$ denotes the probability of an event, in this case with respect to the random sample we pick, and ϵ is any positive value we choose. Putting Inequality (1.4) in words, it says that as the sample size N grows, it becomes exponentially unlikely that ν will deviate from μ by more than our 'tolerance' ϵ.

The only quantity that is random in (1.4) is ν which depends on the random sample. By contrast, μ is not random. It is just a constant, albeit unknown to us. There is a subtle point here. The utility of (1.4) is to infer the value of μ using the value of ν, although it is μ that affects ν, not vice versa. However, since the effect is that ν tends to be close to μ, we infer that μ 'tends' to be close to ν.

Although $\mathbb{P}\left[|\nu \quad \mu| > \epsilon\right]$ depends on μ, as μ appears in the argument and also affects the distribution of ν, we are able to bound the probability by $2e^{-2\epsilon^2 N}$ which does not depend on μ. Notice that only the size N of the sample affects the bound, not the size of the bin. The bin can be large or small, finite or infinite, and we still get the same bound when we use the same sample size.

Exercise 1.9

If $\mu = 0.9$, use the Hoeffding Inequality to bound the probability that a sample of 10 marbles will have $\nu \leq 0.1$ and compare the answer to the previous exercise.

If we choose ϵ to be very small in order to make ν a good approximation of μ, we need a larger sample size N to make the RHS of Inequality (1.4) small. We

can then assert that it is likely that ν will indeed be a good approximation of μ. Although this assertion does not give us the exact value of μ, and doesn't even guarantee that the approximate value holds, knowing that we are within $\pm\epsilon$ of μ most of the time is a significant improvement over not knowing anything at all.

The fact that the sample was randomly selected from the bin is the reason we are able to make any kind of statement about μ being close to ν. If the sample was not randomly selected but picked in a particular way, we would lose the benefit of the probabilistic analysis and we would again be in the dark outside of the sample.

How does the bin model relate to the learning problem? It seems that the unknown here was just the value of μ while the unknown in learning is an entire function $f: \mathcal{X} \to \mathcal{Y}$. The two situations can be connected. Take any single hypothesis $h \in \mathcal{H}$ and compare it to f on each point $\mathbf{x} \in \mathcal{X}$. If $h(\mathbf{x}) = f(\mathbf{x})$, color the point $\mathbf{x}$ green. If $h(\mathbf{x}) \neq f(\mathbf{x})$, color the point $\mathbf{x}$ red. The color that each point gets is not known to us, since f is unknown. However, if we pick $\mathbf{x}$ at random according to some probability distribution P over the input space $\mathcal{X}$, we know that $\mathbf{x}$ will be red with some probability, call it μ, and green with probability $1 - \mu$. Regardless of the value of μ, the space $\mathcal{X}$ now behaves like the bin in Figure 1.8.

The training examples play the role of a sample from the bin. If the inputs $\mathbf{x}_1, \cdots, \mathbf{x}_N$ in $\mathcal{D}$ are picked independently according to P, we will get a random sample of red ($h(\mathbf{x}_n) \neq f(\mathbf{x}_n)$) and green ($h(\mathbf{x}_n) = f(\mathbf{x}_n)$) points. Each point will be red with probability μ and green with probability $1-\mu$. The color of each point will be known to us since both $h(\mathbf{x}_n)$ and $f(\mathbf{x}_n)$ are known for $n = 1, \cdots, N$ (the function h is our hypothesis so we can evaluate it on any point, and $f(\mathbf{x}_n) = y_n$ is given to us for all points in the data set $\mathcal{D}$). The learning problem is now reduced to a bin problem, under the assumption that the inputs in $\mathcal{D}$ are picked independently according to some distribution P on $\mathcal{X}$. Any P will translate to some μ in the equivalent bin. Since μ is allowed to be unknown, P can be unknown to us as well. Figure 1.9 adds this probabilistic component to the basic learning setup depicted in Figure 1.2.

With this equivalence, the Hoeffding Inequality can be applied to the learning problem, allowing us to make a prediction outside of $\mathcal{D}$. Using ν to predict μ tells us something about f, although it doesn't tell us what f is. What μ tells us is the error rate h makes in approximating f. If ν happens to be close to zero, we can predict that h will approximate f well over the entire input space. If not, we are out of luck.

Unfortunately, we have no control over ν in our current situation, since ν is based on a particular hypothesis h. In real learning, we explore an entire hypothesis set $\mathcal{H}$, looking for some $h \in \mathcal{H}$ that has a small error rate. If we have only one hypothesis to begin with, we are not really learning, but rather 'verifying' whether that particular hypothesis is good or bad. Let us see if we can extend the bin equivalence to the case where we have multiple hypotheses in order to capture real learning.

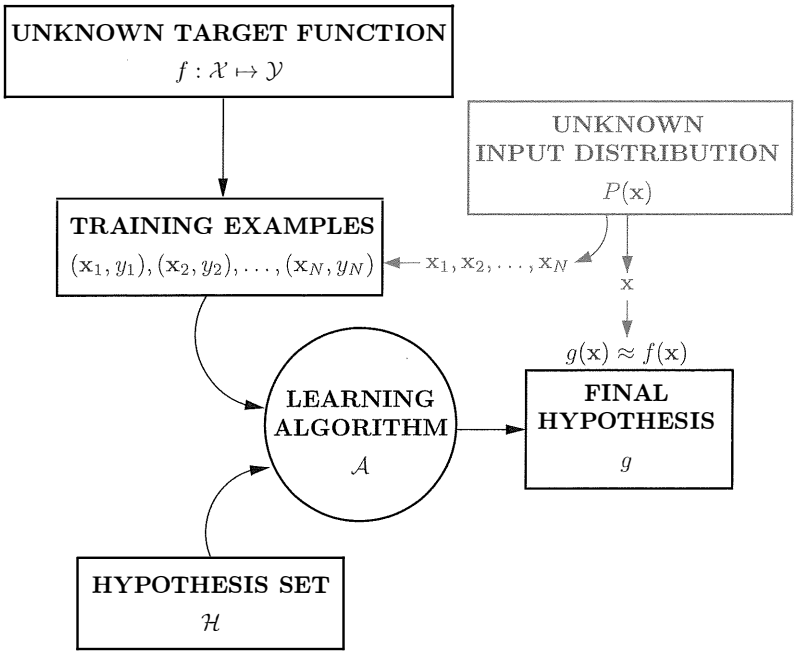


Figure 1.9: Probability added to the basic learning setup

To do that, we start by introducing more descriptive names for the different components that we will use. The error rate within the sample, which corresponds to ν in the bin model, will be called the *in-sample error*,

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

where $[\![\text{statement}]\!] = 1$ if the statement is true, and $= 0$ if the statement is false. We have made explicit the dependency of E_{in} on the particular h that we are considering. In the same way, we define the *out-of-sample error*

$$E_{\text{out}}(h) = \mathbb{P}\left[h(\mathbf{x}) \neq f(\mathbf{x}) \right] ,$$

which corresponds to μ in the bin model. The probability is based on the distribution P over $\mathcal{X}$ which is used to sample the data points $\mathbf{x}$.

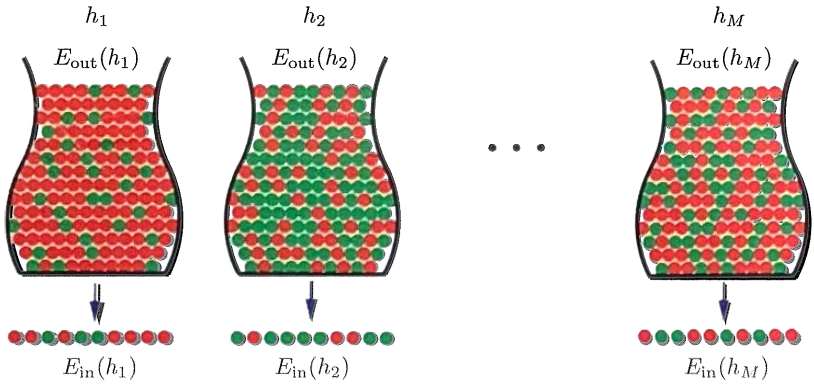


Figure 1.10: Multiple bins depict the learning problem with M hypotheses

Substituting the new notation E_{in} for ν and E_{out} for μ, the Hoeffding Inequality (1.4) can be rewritten as

$$\mathbb{P}\left[|E_{\text{in}}(h) - E_{\text{out}}(h)| > \epsilon\right] \leq 2e^{-2\epsilon^2 N} \qquad \text{for any } \epsilon > 0, \tag{1.5}$$

where N is the number of training examples. The in-sample error E_{in}, just like ν, is a random variable that depends on the sample. The out-of-sample error E_{out}, just like μ, is unknown but not random.

Let us consider an entire hypothesis set $\mathcal{H}$ instead of just one hypothesis h, and assume for the moment that $\mathcal{H}$ has a finite number of hypotheses

$$\mathcal{H} = \{h_1, h_2, \cdots, h_M\}.$$

We can construct a bin equivalent in this case by having M bins as shown in Figure 1.10. Each bin still represents the input space $\mathcal{X}$, with the red marbles in the mth bin corresponding to the points $\mathbf{x} \in \mathcal{X}$ where $h_m(\mathbf{x}) \neq f(\mathbf{x})$. The probability of red marbles in the mth bin is $E_{\text{out}}(h_m)$ and the fraction of red marbles in the mth sample is $E_{\text{in}}(h_m)$, for $m = 1, \cdots, M$. Although the Hoeffding Inequality (1.5) still applies to each bin individually, the situation becomes more complicated when we consider all the bins simultaneously. Why is that? The inequality stated that

$$\mathbb{P}\left[|E_{\text{in}}(h) - E_{\text{out}}(h)| > \epsilon\right] \leq 2e^{-2\epsilon^2 N} \qquad \text{for any } \epsilon > 0,$$

where the hypothesis h is *fixed* before you generate the data set, and the probability is with respect to random data sets $\mathcal{D}$; we emphasize that the assumption "h is fixed *before* you generate the data set" is critical to the validity of this bound. If you are allowed to change h after you generate the data set, the assumptions that are needed to prove the Hoeffding Inequality no longer hold. With multiple hypotheses in $\mathcal{H}$, the learning algorithm picks

the final hypothesis g based on $\mathcal{D}$, i.e. *after* generating the data set. The statement we would like to make is not

$$\text{“}\mathbb{P}[|E_{\text{in}}(h_m) - E_{\text{out}}(h_m)| > \epsilon] \text{ is small”}$$

(for any particular, fixed $h_m \in \mathcal{H}$), but rather

$$\text{“}\mathbb{P}[|E_{\text{in}}(g) - E_{\text{out}}(g)| > \epsilon] \text{ is small” for the final hypothesis } g.$$

The hypothesis g is *not fixed* ahead of time before generating the data, because which hypothesis is selected to be g depends on the data. So, we cannot just plug in g for h in the Hoeffding inequality. The next exercise considers a simple coin experiment that further illustrates the difference between a fixed h and the final hypothesis g selected by the learning algorithm.

Exercise 1.10

Here is an experiment that illustrates the difference between a single bin and multiple bins. Run a computer simulation for flipping $1{,}000$ fair coins. Flip each coin independently 10 times. Let's focus on 3 coins as follows: c_1 is the first coin flipped; c_{rand} is a coin you choose at random; $c_{\min}$ is the coin that had the minimum frequency of heads (pick the earlier one in case of a tie). Let ν_1, ν_{rand} and $\nu_{\min}$ be the fraction of heads you obtain for the respective three coins.

(a) What is μ for the three coins selected?

(b) Repeat this entire experiment a large number of times (e.g., $100{,}000$ runs of the entire experiment) to get several instances of ν_1, ν_{rand} and $\nu_{\min}$ and plot the histograms of the distributions of ν_1, ν_{rand} and $\nu_{\min}$. Notice that which coins end up being c_{rand} and $c_{\min}$ may differ from one run to another.

(c) Using (b), plot estimates for $\mathbb{P}[|\nu - \mu| > \epsilon]$ as a function of ϵ, together with the Hoeffding bound $2e^{-2\epsilon^2 N}$ (on the same graph).

(d) Which coins obey the Hoeffding bound, and which ones do not? Explain why.

(e) Relate part (d) to the multiple bins in Figure 1.10.

The way to get around this is to try to bound $\mathbb{P}[|E_{\text{in}}(g) - E_{\text{out}}(g)| > \epsilon]$ in a way that does not depend on which g the learning algorithm picks. There is a simple but crude way of doing that. Since g has to be one of the h_m's regardless of the algorithm and the sample, it is always true that

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

where $\mathcal{B}_1 \implies \mathcal{B}_2$ means that event $\mathcal{B}_1$ implies event $\mathcal{B}_2$. Although the events on the RHS cover a lot more than the LHS, the RHS has the property we want; the hypotheses h_m are fixed. We now apply two basic rules in probability;

$$\text{if } \mathcal{B}_1 \implies \mathcal{B}_2, \text{ then } \mathbb{P}[\mathcal{B}_1] \leq \mathbb{P}[\mathcal{B}_2],$$

and, if $\mathcal{B}_1, \mathcal{B}_2, \cdots, \mathcal{B}_M$ are any events, then

$$\mathbb{P}[\mathcal{B}_1 \textbf{ or } \mathcal{B}_2 \textbf{ or } \cdots \textbf{ or } \mathcal{B}_M] \leq \mathbb{P}[\mathcal{B}_1] + \mathbb{P}[\mathcal{B}_2] + \cdots + \mathbb{P}[\mathcal{B}_M].$$

The second rule is known as the *union bound*. Putting the two rules together, we get

$$\begin{aligned} \mathbb{P}[\,|E_{\text{in}}(g) - E_{\text{out}}(g)| > \epsilon\,] \quad \leq \quad & \mathbb{P}[\quad |E_{\text{in}}(h_1) - E_{\text{out}}(h_1)| > \epsilon \\ & \quad \textbf{or } |E_{\text{in}}(h_2) - E_{\text{out}}(h_2)| > \epsilon \\ & \quad \cdots \\ & \quad \textbf{or } |E_{\text{in}}(h_M) - E_{\text{out}}(h_M)| > \epsilon\,] \\ \leq \quad & \sum_{m=1}^{M} \mathbb{P}\left[|E_{\text{in}}(h_m) \quad E_{\text{out}}(h_m)| > \epsilon\right]. \end{aligned}$$

Applying the Hoeffding Inequality (1.5) to the M terms one at a time, we can bound each term in the sum by $2e^{-2\epsilon^2 N}$. Substituting, we get

$$\mathbb{P}[|E_{\text{in}}(g) - E_{\text{out}}(g)| > \epsilon] \leq 2Me^{-2\epsilon^2 N}. \tag{1.6}$$

Mathematically, this is a 'uniform' version of (1.5). We are trying to simultaneously approximate all $E_{\text{out}}(h_m)$'s by the corresponding $E_{\text{in}}(h_m)$'s. This allows the learning algorithm to choose any hypothesis based on E_{in} and expect that the corresponding E_{out} will uniformly follow suit, regardless of which hypothesis is chosen.

The downside for uniform estimates is that the probability bound $2Me^{-2\epsilon^2 N}$ is a factor of M looser than the bound for a single hypothesis, and will only be meaningful if M is finite. We will improve on that in Chapter 2.

1.3.3 Feasibility of Learning

We have introduced two apparently conflicting arguments about the feasibility of learning. One argument says that we cannot learn anything outside of $\mathcal{D}$, and the other says that we can. We would like to reconcile these two arguments and pinpoint the sense in which learning is feasible:

1. Let us reconcile the two arguments. The question of whether $\mathcal{D}$ tells us anything outside of $\mathcal{D}$ that we didn't know before has two different answers. If we insist on a deterministic answer, which means that $\mathcal{D}$ tells us something certain about f outside of $\mathcal{D}$, then the answer is no. If we accept a probabilistic answer, which means that $\mathcal{D}$ tells us something likely about f outside of $\mathcal{D}$, then the answer is yes.

Exercise 1.11

We are given a data set $\mathcal{D}$ of 25 training examples from an unknown target function $f: \mathcal{X} \to \mathcal{Y}$, where $\mathcal{X} = \mathbb{R}$ and $\mathcal{Y} = \{-1, +1\}$. To learn f, we use a simple hypothesis set $\mathcal{H} = \{h_1, h_2\}$ where h_1 is the constant $+1$ function and h_2 is the constant -1.

We consider two learning algorithms, S (smart) and C (crazy). S chooses the hypothesis that agrees the most with $\mathcal{D}$ and C chooses the other hypothesis deliberately. Let us see how these algorithms perform out of sample from the deterministic and probabilistic points of view. Assume in the probabilistic view that there is a probability distribution on $\mathcal{X}$, and let $\mathbb{P}[f(\mathbf{x}) = +1] = p$.

(a) Can S produce a hypothesis that is *guaranteed* to perform better than random on any point outside $\mathcal{D}$?

(b) Assume for the rest of the exercise that all the examples in $\mathcal{D}$ have $y_n = +1$. Is it *possible* that the hypothesis that C produces turns out to be better than the hypothesis that S produces?

(c) If $p = 0.9$, what is the probability that S will produce a better hypothesis than C?

(d) Is there any value of p for which it is more likely than not that C will produce a better hypothesis than S?

By adopting the probabilistic view, we get a positive answer to the feasibility question without paying too much of a price. The only assumption we make in the probabilistic framework is that the examples in $\mathcal{D}$ are generated independently. We don't insist on using any particular probability distribution, or even on knowing what distribution is used. However, whatever distribution we use for generating the examples, we must also use when we evaluate how well g approximates f (Figure 1.9). That's what makes the Hoeffding Inequality applicable. Of course this ideal situation may not always happen in practice, and some variations of it have been explored in the literature.

2. Let us pin down what we mean by the feasibility of learning. Learning produces a hypothesis g to approximate the unknown target function f. If learning is successful, then g should approximate f well, which means $E_{\text{out}}(g) \approx 0$. However, this is not what we get from the probabilistic analysis. What we get instead is $E_{\text{out}}(g) \approx E_{\text{in}}(g)$. We still have to make $E_{\text{in}}(g) \approx 0$ in order to conclude that $E_{\text{out}}(g) \approx 0$.

We cannot guarantee that we will find a hypothesis that achieves $E_{\text{in}}(g) \approx 0$, but at least we will know if we find it. Remember that $E_{\text{out}}(g)$ is an unknown quantity, since f is unknown, but $E_{\text{in}}(g)$ is a quantity that we can evaluate. We have thus traded the condition $E_{\text{out}}(g) \approx 0$, one that we cannot ascertain, for the condition $E_{\text{in}}(g) \approx 0$, which we can ascertain. What enabled this is the Hoeffding Inequality (1.6):

$$\mathbb{P}[|E_{\text{in}}(g) \quad E_{\text{out}}(g)| > \epsilon] \leq 2Me^{\ 2\epsilon^2 N}$$

that assures us that $E_{\text{out}}(g) \approx E_{\text{in}}(g)$ so we can use E_{in} as a proxy for E_{out}.

Exercise 1.12

A friend comes to you with a learning problem. She says the target function f is *completely* unknown, but she has $4,000$ data points. She is willing to pay you to solve her problem and produce for her a g which approximates f. What is the best that you can promise her among the following:

(a) After learning you will provide her with a g that you will guarantee approximates f well out of sample.

(b) After learning you will provide her with a g, and with high probability the g which you produce will approximate f well out of sample.

(c) One of two things will happen.

  (i) You will produce a hypothesis g;

  (ii) You will declare that you failed.

  If you do return a hypothesis g, then with high probability the g which you produce will approximate f well out of sample.

One should note that there are cases where we won't insist that $E_{\text{in}}(g) \approx 0$. Financial forecasting is an example where market unpredictability makes it impossible to get a forecast that has anywhere near zero error. All we hope for is a forecast that gets it right more often than not. If we get that, our bets will win in the long run. This means that a hypothesis that has $E_{\text{in}}(g)$ somewhat below 0.5 will work, provided of course that $E_{\text{out}}(g)$ is close enough to $E_{\text{in}}(g)$.

The feasibility of learning is thus split into two questions:

1. Can we make sure that $E_{\text{out}}(g)$ is close enough to $E_{\text{in}}(g)$?
2. Can we make $E_{\text{in}}(g)$ small enough?

The Hoeffding Inequality (1.6) addresses the first question only. The second question is answered after we run the learning algorithm on the actual data and see how small we can get E_{in} to be.

Breaking down the feasibility of learning into these two questions provides further insight into the role that different components of the learning problem play. One such insight has to do with the 'complexity' of these components.

The complexity of $\mathcal{H}$. If the number of hypotheses M goes up, we run more risk that $E_{\text{in}}(g)$ will be a poor estimator of $E_{\text{out}}(g)$ according to Inequality (1.6). M can be thought of as a measure of the 'complexity' of the

hypothesis set $\mathcal{H}$ that we use. If we want an affirmative answer to the first question, we need to keep the complexity of $\mathcal{H}$ in check. However, if we want an affirmative answer to the second question, we stand a better chance if $\mathcal{H}$ is more complex, since g has to come from $\mathcal{H}$. So, a more complex $\mathcal{H}$ gives us more flexibility in finding some g that fits the data well, leading to small $E_{\text{in}}(g)$. This tradeoff in the complexity of $\mathcal{H}$ is a major theme in learning theory that we will study in detail in Chapter 2.

The complexity of f. Intuitively, a complex target function f should be harder to learn than a simple f. Let us examine if this can be inferred from the two questions above. A close look at Inequality (1.6) reveals that the complexity of f does not affect how well $E_{\text{in}}(g)$ approximates $E_{\text{out}}(g)$. If we fix the hypothesis set and the number of training examples, the inequality provides the same bound whether we are trying to learn a simple f (for instance a constant function) or a complex f (for instance a highly nonlinear function). However, this doesn't mean that we can learn complex functions as easily as we learn simple functions. Remember that (1.6) affects the first question only. If the target function is complex, the second question comes into play since the data from a complex f are harder to fit than the data from a simple f. This means that we will get a worse value for $E_{\text{in}}(g)$ when f is complex. We might try to get around that by making our hypothesis set more complex so that we can fit the data better and get a lower $E_{\text{in}}(g)$, but then E_{out} won't be as close to E_{in} per (1.6). Either way we look at it, a complex f is harder to learn as we expected. In the extreme case, if f is too complex, we may not be able to learn it at all.

Fortunately, most target functions in real life are not too complex; we can learn them from a reasonable $\mathcal{D}$ using a reasonable $\mathcal{H}$. This is obviously a practical observation, not a mathematical statement. Even when we cannot learn a particular f, we will at least be able to tell that we can't. As long as we make sure that the complexity of $\mathcal{H}$ gives us a good Hoeffding bound, our success or failure in learning f can be determined by our success or failure in fitting the training data.

1.4 Error and Noise

We close this chapter by revisiting two notions in the learning problem in order to bring them closer to the real world. The first notion is what approximation means when we say that our hypothesis approximates the target function well. The second notion is about the nature of the target function. In many situations, there is noise that makes the output of f not uniquely determined by the input. What are the ramifications of having such a 'noisy' target on the learning problem?

1.4.1 Error Measures

Learning is not expected to replicate the target function perfectly. The final hypothesis g is only an approximation of f. To quantify how well g approximates f, we need to define an error measure[3] that quantifies how far we are from the target.

The choice of an error measure affects the outcome of the learning process. Different error measures may lead to different choices of the final hypothesis, even if the target and the data are the same, since the value of a particular error measure may be small while the value of another error measure in the same situation is large. Therefore, which error measure we use has consequences for what we learn. What are the criteria for choosing one error measure over another? We address this question here.

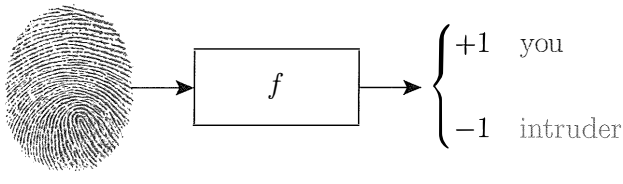
First, let's formalize this notion a bit. An error measure quantifies how well each hypothesis h in the model approximates the target function f,

$$\text{Error} \;=\; E(h, f).$$

While $E(h, f)$ is based on the entirety of h and f, it is almost universally defined based on the errors on individual input points $\mathbf{x}$. If we define a pointwise error measure $\mathsf{e}(h(\mathbf{x}), f(\mathbf{x}))$, the overall error will be the average value of this pointwise error. So far, we have been working with the classification error $\mathsf{e}(h(\mathbf{x}), f(\mathbf{x})) = [\![h(\mathbf{x}) \neq f(\mathbf{x})]\!]$.

In an ideal world, $E(h, f)$ should be user-specified. The same learning task in different contexts may warrant the use of different error measures. One may view $E(h, f)$ as the 'cost' of using h when you should use f. This cost depends on what h is used for, and cannot be dictated just by our learning techniques. Here is a case in point.

Example 1.1 (Fingerprint verification)**.** Consider the problem of verifying that a fingerprint belongs to a particular person. What is the appropriate error measure?



The target function takes as input a fingerprint, and returns $+1$ if it belongs to the right person, and -1 if it belongs to an intruder.

[3]This measure is also called an error *function* in the literature, and sometimes the error is referred to as *cost*, *objective*, or *risk*.

There are two types of error that our hypothesis h can make here. If the correct person is rejected ($h = -1$ but $f = +1$), it is called false reject, and if an incorrect person is accepted ($h = +1$ but $f = -1$), it is called false accept.

		f	
		$+1$	-1
h	$+1$	no error	false accept
	-1	false reject	no error

How should the error measure be defined in this problem? If the right person is accepted or an intruder is rejected, the error is clearly zero. We need to specify the error values for a false accept and for a false reject. The right values depend on the application.

Consider two potential clients of this fingerprint system. One is a supermarket who will use it at the checkout counter to verify that you are a member of a discount program. The other is the CIA who will use it at the entrance to a secure facility to verify that you are authorized to enter that facility.

For the supermarket, a false reject is costly because if a customer gets wrongly rejected, she may be discouraged from patronizing the supermarket in the future. All future revenue from this annoyed customer is lost. On the other hand, the cost of a false accept is minor. You just gave away a discount to someone who didn't deserve it, and that person left their fingerprint in your system they must be bold indeed.

For the CIA, a false accept is a disaster. An unauthorized person will gain access to a highly sensitive facility. This should be reflected in a much higher cost for the false accept. False rejects, on the other hand, can be tolerated since authorized persons are employees (rather than customers as with the supermarket). The inconvenience of retrying when rejected is just part of the job, and they must deal with it.

The costs of the different types of errors can be tabulated in a matrix. For our examples, the matrices might look like:

		f	
		$+1$	-1
h	$+1$	0	1
	-1	10	0

Supermarket

		f	
		$+1$	-1
h	$+1$	0	1000
	-1	1	0

CIA

These matrices should be used to weight the different types of errors when we compute the total error. When the learning algorithm minimizes a cost-weighted error measure, it automatically takes into consideration the utility of the hypothesis that it will produce. In the supermarket and CIA scenarios, this could lead to two completely different final hypotheses. □

The moral of this example is that the choice of the error measure depends on how the system is going to be used, rather than on any inherent criterion

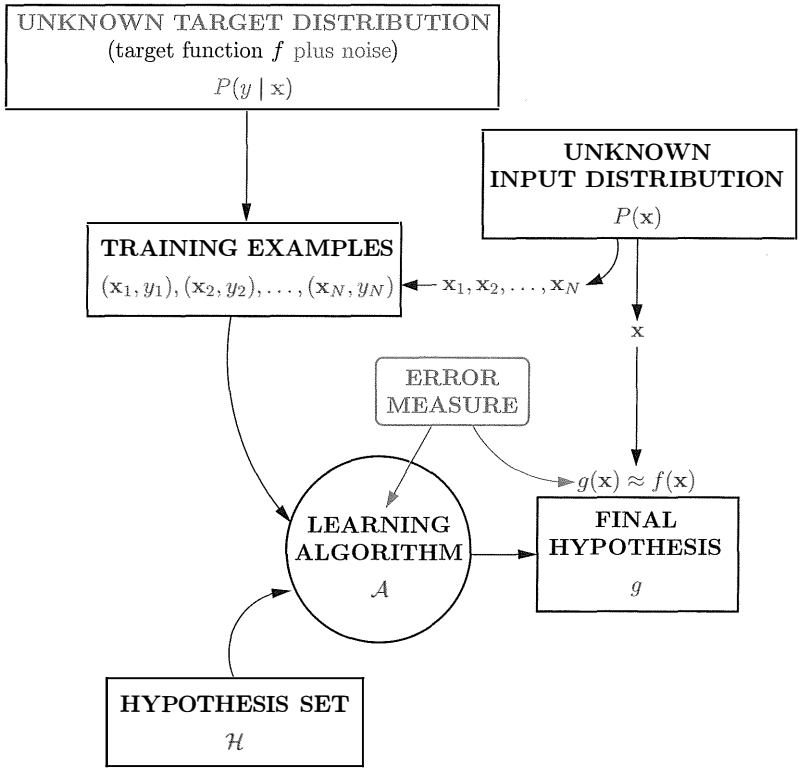


Figure 1.11: The general (supervised) learning problem

that we can independently determine during the learning process. However, this ideal choice may not be possible in practice for two reasons. One is that the user may not provide an error specification, which is not uncommon. The other is that the weighted cost may be a difficult objective function for optimizers to work with. Therefore, we often look for other ways to define the error measure, sometimes with purely practical or analytic considerations in mind. We have already seen an example of this with the simple binary error used in this chapter, and we will see other error measures in later chapters.

1.4.2 Noisy Targets

In many practical applications, the data we learn from are not generated by a deterministic target function. Instead, they are generated in a noisy way such that the output is not uniquely determined by the input. For instance, in the credit-card example we presented in Section 1.1, two customers may have identical salaries, outstanding loans, etc., but end up with different credit behavior. Therefore, the credit 'function' is not really a deterministic function,

but a noisy one.

This situation can be readily modeled within the same framework that we have. Instead of $y = f(\mathbf{x})$, we can take the output y to be a random variable that is affected by, rather than determined by, the input $\mathbf{x}$. Formally, we have a target *distribution* $P(y \mid \mathbf{x})$ instead of a target function $y = f(\mathbf{x})$. A data point $(\mathbf{x}, y)$ is now generated by the joint distribution $P(\mathbf{x}, y) = P(\mathbf{x})P(y \mid \mathbf{x})$.

One can think of a noisy target as a deterministic target plus added noise. If y is real-valued for example, one can take the expected value of y given $\mathbf{x}$ to be the deterministic $f(\mathbf{x})$, and consider $y - f(\mathbf{x})$ as pure noise that is added to f.

This view suggests that a deterministic target function can be considered a special case of a noisy target, just with zero noise. Indeed, we can formally express any function f as a distribution $P(y \mid \mathbf{x})$ by choosing $P(y \mid \mathbf{x})$ to be zero for all y except $y = f(\mathbf{x})$. Therefore, there is no loss of generality if we consider the target to be a distribution rather than a function. Figure 1.11 modifies the previous Figures 1.2 and 1.9 to illustrate the general learning problem, covering both deterministic and noisy targets.

Exercise 1.13

Consider the bin model for a hypothesis h that makes an error with probability μ in approximating a deterministic target function f (both h and f are binary functions). If we use the same h to approximate a noisy version of f given by

$$P(y \mid \mathbf{x}) = \begin{cases} \lambda & y = f(\mathbf{x}), \\ 1 - \lambda & y \neq f(\mathbf{x}). \end{cases}$$

(a) What is the probability of error that h makes in approximating y?

(b) At what value of λ will the performance of h be independent of μ? *[Hint: The noisy target will look completely random.]*

There is a difference between the role of $P(y \mid \mathbf{x})$ and the role of $P(\mathbf{x})$ in the learning problem. While both distributions model probabilistic aspects of $\mathbf{x}$ and y, the target distribution $P(y \mid \mathbf{x})$ is what we are trying to learn, while the input distribution $P(\mathbf{x})$ only quantifies the relative importance of the point $\mathbf{x}$ in gauging how well we have learned.

Our entire analysis of the feasibility of learning applies to noisy target functions as well. Intuitively, this is because the Hoeffding Inequality (1.6) applies to an arbitrary, unknown target function. Assume we randomly picked all the y's according to the distribution $P(y \mid \mathbf{x})$ over the entire input space $\mathcal{X}$. This realization of $P(y \mid \mathbf{x})$ is effectively a target function. Therefore, the inequality will be valid no matter which particular random realization the 'target function' happens to be.

This does not mean that learning a noisy target is as easy as learning a deterministic one. Remember the two questions of learning? With the same learning model, E_{out} may be as close to E_{in} in the noisy case as it is in the

deterministic case, but E_{in} itself will likely be worse in the noisy case since it is hard to fit the noise.

In Chapter 2, where we prove a stronger version of (1.6), we will assume the target to be a probability distribution $P(y \mid \mathbf{x})$, thus covering the general case.