Chapter

2 SUPERVISED LEARNING: RATIONALE AND BASICS

2.1 LEARNING FROM OBSERVATIONS

We consider the following setting in this chapter to present the basics of learning theory.

There is some set S of possible patterns/observations/samples over which various output functions may be defined. The training experience is available in the form of N patterns $s^{(i)} \in S$; $i = 1, 2, \ldots, N$. We specify a pattern by a fixed number n of attributes/features x_j; $j = 1, 2, \ldots, n$; where each feature has real numerical value for the pattern. The domain of x_j is given by a set $V_{x_j} \in \Re$ of its values. A data pattern $s^{(i)}$ has the feature-value set $\{x_1^{(i)}, \ldots, x_n^{(i)}\}$, where $x_j^{(i)} \in V_{x_j}$. We can visualize each pattern with n numerical features as a point in n-dimensional state space $\Re^n$:

$$\mathbf{x} = [x_1\ x_2\ \cdots\ x_n]^T \in \Re^n$$

The set $\mathbf{X}$ is the finite set of feature vectors $\mathbf{x}^{(i)}$ for all the N patterns. We can visualize $\mathbf{X}$ as a region of the state space $\Re^n$ to which the patterns belong, i.e., $\mathbf{X} \subset \Re^n$. Note that $\mathbf{x}^{(i)}$ is the representation of $s^{(i)}$; and $\mathbf{X}$ is the *representation space*.

For supervised learning problems, the decision attribute (output) y is known *a priori* for each pattern in the set S. For multiclass classification (pattern recognition) problems, the domain of y is given by the set $V_y = \{1, 2, \ldots, M\}$. The output $y^{(i)}$ for a pattern $s^{(i)}$ will take the value from the set V_y, which represents the class to which the pattern belongs. Thus, in classification problems, each pattern $s^{(i)}$ is associated with an output $y^{(i)} \in V_y$. For regression (numeric prediction) problems, $V_y \in \Re$, and each pattern $s^{(i)}$ is associated with an output $y^{(i)} \in \Re$. We can visualize the output belonging to one-dimensional region Y of the state space.

We assume that different patterns $\mathbf{x}^{(i)}$ in $\mathbf{X}$ may be encountered with different frequencies. A convenient way to model this is to assume that there is some unknown probability distribution that defines the probability of encountering each pattern $\mathbf{x}^{(i)}$ in $\mathbf{X}$. The training examples are provided to the learning machine by a trainer who draws each pattern independently according to

the distribution, and who then forwards the pattern $\mathbf{x}$ along with its true (observed) output y to the learning machine. The training experience is in the form of data $\mathcal{D}$ that describes how the system behaves over its entire range of operation.

$$\mathcal{D} : \{\mathbf{x}^{(i)}, y^{(i)}\}; i = 1, 2, \ldots, N \tag{2.1}$$

In general, learning is most reliable when the training examples follow a distribution similar to that of future patterns (unseen by the machine during training) which the machine will receive; the machine will be required to give estimated output values $\hat{y}$ for these patterns. If the training experience consists of data that lies in a region of the state space, then that region must be fully representative of situations over which the trained machine will later be used. The most current theory of machine learning rests on the crucial assumption that the distribution of training examples is identical to the distribution of unseen examples.

Assume that the data $\mathcal{D}$ are independently drawn and identically distributed (commonly referred to as *iid* data) samples from some unknown probability distribution represented by the probability density function $p(\mathbf{x}, \mathrm{y})$ (refer to Section 3.2). Assume a machine defined by a function

$$f : \mathbf{X} \rightarrow Y$$

that maps from $\mathbf{x} \in \mathbf{X}$ to $y \in Y$, where $\mathbf{X}$ is the *input space* and Y is the *output space* of the function. When $f(\cdot)$ is selected, the machine is called a *trained machine* that gives estimated output value

$$\hat{y} = f(\mathbf{x})$$

for a given pattern $\mathbf{x}$. To assess the success of learning, we need an evaluation criterion. The evaluation criteria are generally based on the consequences of the decision made by the machine, namely—the errors, profits or losses, penalties or rewards involved. In supervised learning, quite often used criterion is a minimization criterion involving potential loss into a classification/regression decision made.

From the set of possible learning machines (functions), we want to select the *optimal* one that minimizes the loss. We can define the set of learning machines by a function $f(\mathbf{x}, \mathbf{w})$, where $\mathbf{w}$ contains adjustable parameters. Thus, the set of parameters $\mathbf{w}$ becomes the subject of learning. A *loss function* $L(y, f(\mathbf{x}, \mathbf{w}))$ is a measure of the error between the actual output y and estimated output

$$\hat{y} = f(\mathbf{x}, \mathbf{w}) \tag{2.2}$$

Suppose we observe a particular $\mathbf{x}$ and we contemplate taking decision $f(\mathbf{x}, \mathbf{w})$ which yields the output $\hat{y}$ (the estimated output). If the true value of the output is y, by definition, we will incur the loss $L(y, f(\mathbf{x}, \mathbf{w}))$. If $p(\mathbf{x}, y)$ represents the joint probability density function of the data, then expected loss associated with decision $f(\mathbf{x}, \mathbf{w})$ is merely (refer to Section 3.2)

$$\mathbb{E}[L(y, f(\mathbf{x}, \mathbf{w}))] = \int_{\mathbf{X} \times Y} L(y, f(\mathbf{x}, \mathbf{w}))\, p(\mathbf{x}, y)\, d\mathbf{x}\, dy = R(\mathbf{w}) \tag{2.3}$$

$d\mathbf{x}\, dy$ is our notation for the $(n + 1)$-space volume element, and the integral extends over the entire joint space created by feature space (input space) $\mathbf{X}$ and output space Y.

A *risk* refers to a predictable loss in decision-theoretic terminology, and $R(\mathbf{w})$ is known as the *risk function*. On coming across a specific observation $\mathbf{x}$, the expected loss can be minimized by

choosing a decision function that minimizes the risk function. If tractable, this process actually provides optimum performance.

Stated formally, *our problem is to find a decision function* $f(\mathbf{x}, \mathbf{w})$ *against* $p(\mathbf{x}, y)$ *that minimizes the risk function* $R(\mathbf{w})$.

Thus, we could design optimal classifiers/numeric predictors if we know the joint probability density function $p(\mathbf{x}, y)$. In machine learning applications, this type of information pertaining to the probabilistic structure of the problem is hardly found. Typically, we simply have some vague general knowledge regarding the situation together with several *design samples* or training data—specific representatives of the patterns we wish to categorize/regress. The issue, therefore, is to seek a way to employ this information to design or train the learning machine. Different processes that have been established to tackle such problems, will be considered throughout the book.

Empirical-Risk-Minimization

The learning problem defined before is, in fact, intractable, since we do not know $p(\mathbf{x}, y)$ explicitly. The risk function representing *true risk*, given by Eqn (2.3), cannot be calculated. All that is available is a training dataset of examples drawn by independent sampling a $(\mathbf{X} \times Y)$ space according to some unknown probability distribution. Therefore, a learning machine can at best guarantee that the estimated output values $\hat{y}$ fit the true values y over the training data.

With dataset (2.1) being the only source of information, the risk function given by Eqn (2.3) must be approximated by the *empirical risk*, $R_{\text{emp}}(\mathbf{w})$:

$$R_{\text{emp}}(\mathbf{w}) \triangleq \frac{1}{N}\sum_{i=1}^{N} L(y^{(i)}, f(\mathbf{x}^{(i)}, \mathbf{w})) \tag{2.4}$$

The empirical risk given by Eqn (2.4), replaces average over $p(\mathbf{x}, y)$ by an average over the training sample.

In classification problems, probably the simplest and quite often used criterion involves counting the misclassification error (Section 2.8); if a pattern $\mathbf{x}$ is classified wrongly, we incur loss 1, otherwise there is no penalty. The *loss function for classification problems*

$$L(y, f(\mathbf{x}, \mathbf{w})) = \begin{cases} 0 & \text{if } y = f(\mathbf{x}, \mathbf{w}) = \hat{y} \\ 1 & \text{otherwise} \end{cases} \tag{2.5}$$

where $\mathbf{x}$ denotes the pattern, y denotes observation (true output) and $f(\mathbf{x}, \mathbf{w}) = \hat{y}$ is the estimation. The minimum of the loss function is 0.

When estimating the real-valued quantities (regression problems), it is usually the size of the difference $(y - f(\mathbf{x}, \mathbf{w}))$, i.e., the amount of misestimation (misprediction), which is used to determine the quality of the estimate. The popular choice (Section 2.7) is to minimize the sum of squares of the residuals $(y - f(\mathbf{x}, \mathbf{w}))$. In most cases, the *loss function for the regression problems* will be of the type

$$L(y, f(\mathbf{x}, \mathbf{w})) = \tfrac{1}{2}(y - f(\mathbf{x}, \mathbf{w}))^2 \tag{2.6}$$

It appears as if (2.4) is the answer to our problems and all that remains to be done is to find a suitable function $f(\cdot)$ out of the set of all possible functions (it is an infinite set, theoretically) that minimizes R_{emp}. Unfortunately, this problem of finding the minimum of the empirical risk is an *ill-posed problem*. Here, the 'ill-posed' characteristic of the problem is due to the infinite number of possible solutions to the problem. We will show this with the help of some simple examples.

Before we take up the examples, a relook at the *learning task* in hand will be helpful. Given a set of training examples $\{\mathbf{x}^{(i)}, y^{(i)}\}$; $i = 1, 2, \ldots, N$, the learning task requires estimating/predicting outputs $\hat{y} = f(\mathbf{x}, \mathbf{w})$ for the data beyond the training data, i.e., for the data the machine has not seen during its training phase. The only information available about $f(\cdot)$ is its true value over the training examples. Our aim is to use the machine for prediction of the output (required for decision making) for the data for which the true output is not known.

The key assumption for reliable learning is that the training examples and future examples (unseen by the machine during training) are drawn randomly and independently from some population of examples with the same (though unknown) probability distribution (the data is *iid*—independently and identically distributed). If this assumption is met, we can claim that a learning machine giving 100% accuracy on training data will give high accuracy on unseen data (data beyond the training sample). However, this assumption is not guaranteed. In fact, machine learning tasks based on real-world data are unlikely to find the *iid* data assumption tenable. The real-world data tend to be incomplete, noisy, and inconsistent (we will consider these aspects of real-world data in later chapters).

Let us now return to the examples.

Consider a regression problem. Suppose we are given empirical observations:

$$((x^{(1)}, y^{(1)}), \ldots, (x^{(N)}, y^{(N)})) \in \mathbf{X} \times Y$$

where for simplicity, we take $\mathbf{X} \in \Re$, and $Y \in \Re$. Figure 2.1 shows a plot of such a dataset (data points indicated by ○). Note that all functions that *interpolate* these data points will result in zero value for R_{emp}. Figure 2.1 shows two out of infinite many different interpolating functions of training data pairs that are possible. Each of the two interpolants results in $R_{\text{emp}} = 0$, but at the same time, none is a good model of the true underlying dependency between $\mathbf{X}$ and Y, because both the functions perform very poorly outside the training inputs (indicated as ×). Thus, interpolating functions that result in zero empirical risk, can mislead. There are many other approximating functions (learning machines) that will minimize the empirical risk (*training error*) but not necessarily the true (expected) risk.

Consider now a classification problem. In two-class pattern recognition, we seek to infer a function

$$f(\cdot) : \mathbf{X} \rightarrow \{\pm 1\}, \text{ i.e., } Y = \{\pm 1\}$$

Figure 2.2 shows a simple 2-dimensional example of a pattern recognition problem. The task is to separate solid dots from circles by finding a function that takes +1 on the dots and –1 on the circles. The classification function shown by dashed line in Fig. 2.2 separates all training points correctly.

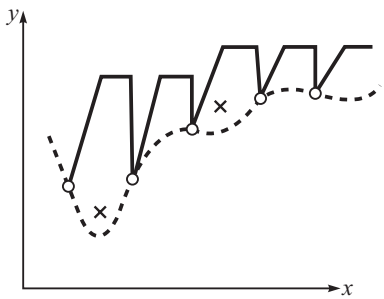


Figure 2.1 A simple regression example

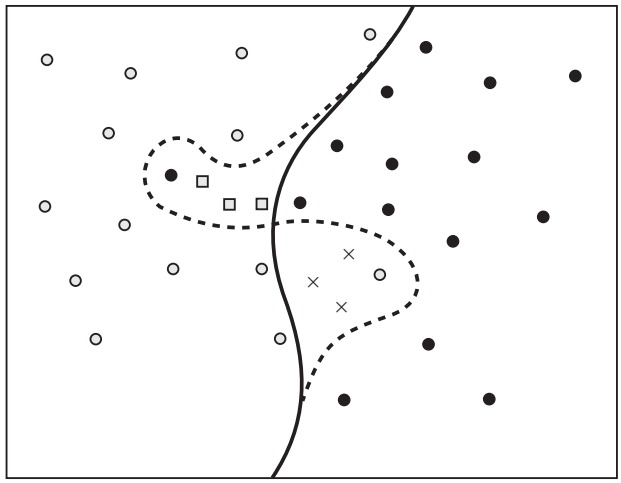


Figure 2.2 A simple classification example

From this figure, it is, however, unclear whether the same would hold true for other points. For instance, we take some *test* points for which the output is known but these points have not been used for training. The test points indicated by × are known to have +1 output and those indicated by □ have –1 output. We see from Fig. 2.2 that a classifier with even zero training error may not be able to get all test points right. We need some compromise (e.g., the decision boundary shown by solid line in Fig. 2.2) which gets most of the test points right; such a classifier may not possess zero training error.

Minimization of training error given by the empirical risk function in (2.4) is, thus, not a solution to our learning task. Minimization of true error (errors of prediction for total data, that includes both training and unseen patterns) given by the risk function in (2.3) is our learning task, but this problem is intractable with $p(\mathbf{x}, y)$ unknown explicitly.

A generic question about machine learning is:

What is it that we need for empirical risk minimization to lead to a successful solution to the learning task?

Lacking any further information except the training samples (which is the situation for most of the complex decision-making problems we face today), the solution lies in *inductive inference*.

Inductive Learning

The task of inductive learning is this:

Given a collection of examples $(\mathbf{x}^{(i)} f(\mathbf{x}^{(i)}))$; $i = 1, 2, \ldots, N$, of a function $f(\mathbf{x})$, return a function $h(\mathbf{x})$ that *approximates* $f(\mathbf{x})$.

In the statistical literature, the approximating function $h(\mathbf{x})$ is called *hypothesis function*.

The *true function* $f(\mathbf{x})$ correctly maps the input space $\mathbf{X}$ (of the entire data) to the output space Y. This function is not known in a real-world decision-making problem. Our satisfaction on an hypothesis function (*machine learning algorithm*) $h(\mathbf{x})$ with high accuracy on the given collection of examples (learning examples for the machine/examples for training the machine) of $f(\mathbf{x})$ could be premature because *central aim* of designing a machine is to suggest decisions when presented with *novel* patterns (unseen by the machine during training).

From a conceptual point of view, it is not easy to tell whether any particular $h(\cdot)$ is a good approximation of $f(\cdot)$. A good approximation will *generalize* well—that is, will predict novel patterns correctly. *Generalization performance is, thus*, *the fundamental problem in inductive learning*.

How do we judge the generalization performance of an algorithm? We can estimate such performance by means of a *test dataset*, sampled independently as is the training set. The *off-training set error*—the error on points *not* in the training set, will be used as a measure of generalization performance. Details will appear later in this chapter.

The assumption in inductive learning is that the ideal hypothesis related to unseen patterns is the one *induced* by the observed training data. Inductive learning techniques obtain a general hypothesis by seeking *empirical regularities* over the training examples. These regularities induce the approximation of the mapping function well over the observed examples. It generalizes from the specific training examples, hypothesizing a general function covering these examples.

When we confront various hypotheses (*hypothesis space*) we also encounter certain generic questions:

If we are only interested in the generalizing performance, is there a reason for the preference of one hypothesis over another? If a specific hypothesis outperforms another over some data sample, will the hypothesis necessarily be more accurate? Is it possible for a hypothesis to be better on the whole?

No Free Lunch Theorem [4] highlights the unpleasant fact that if the aim is to achieve perfect generalization performance, there are no reasons—independent of context or usage—to prefer one learning algorithm over another. If one algorithm appears to outperform another in a specific situation, it is as a result of its fit to the specific learning problem, *not* the general supremacy of the algorithm. Even if the algorithms are widely used and grounded in terms of theory, they will not perform well on certain problems.

It is clear from the No Free Lunch Theorem that in absence of knowledge pertaining to the relevant learning domains, we should not favor any learning algorithm over another. According to the *Ugly Duckling Theorem* [4], in absence of assumptions about the learning domains there is no privileged or 'best' feature representation even the idea of similarity between patterns is dependent

on assumptions we make related to the learning domains. These assumptions may be correct or incorrect.

In certain fields, rigid conservation laws exist—such as conservation of energy, charge, and momentum; and constraint laws—such as the second law of thermodynamics. These apply irrespective of how many forces are at play or their configuration. Do analogous results exist in machine learning? Certainly not. Unfortunately, no rigid results in machine learning exist that can apply to all situations, irrespective of the learning algorithm selected, the number of patterns and their distribution, and the nature of the learning task. Awareness of the learning domains, and experience with a wide range of algorithms is the greatest insurance to solve new learning problems.

Many results that look for ways to quantify the 'match' between a learning algorithm and the problem addressed by it, will appear in the forthcoming discussions. In classical statistics, this problem has been examined in terms of *bias* and *variance*. The two terms are not independent, as is clear from Section 2.2.

Fragments of *statistical learning theory* have begun to emerge, providing answers to generic questions about machine learning within particular problem settings. *VC* (Vapnik-Chervonenkis) model is mainly concerned with *hypothesis space complexity*. PAC (Probably-Accurately-Correct) model answers questions on *sample complexity* (training-data size), and *computational complexity*. We will present these results in Section 2.3.

As we shall see in this chapter, such results are of theoretical interest. Practitioners, by and large, are depending on *heuristic* (trial-and-error) search process. Trial-and-error effort can, of course, be reduced using knowledge of the learning domains, and experience with broad range of learning algorithms. Frequent empirical 'successes' of philosophical principle of *Occam's razor*, and *minimum description length principle*, have given some tools in the hands of practitioners.

Section 2.4 uses intuitive platform of discussion on Occam's razor principle and *overfitting avoidance*. The discussion on minimum description length principle will be taken up in Chapter 3.

The discussion given in this book is without much mathematical rigor or statistical justification. Our aim is to give main insights of *learning theory* in a fairly simple manner.

2.2 BIAS AND VARIANCE

Consider performing the following experiment. We first collect a random sample $\mathcal{D}$ of N independently drawn patterns from the distribution $p(\mathbf{x}, y)$, and then measure the *sample error*/*training error*/*approximation error* from Eqn (2.4), using loss function (2.5) for classification problem or (2.6) for regression problem. Let us denote the approximation error based on data sample $\mathcal{D}$ and hypothesis h as '$\text{error}_{\mathcal{D}}[h]$'. If we repeat the experiment several times, each time drawing a different sample $(\mathbf{x}^{(i)}, y^{(i)})$ of size N, we would expect to see different values of $\text{error}_{\mathcal{D}}[h]$, on the basis of the random differences in the way the different samples of size N are made up. We say in such cases that $\text{error}_{\mathcal{D}_j}[h]$, the result of the j^{th} experiment, is a *random variable*.

Imagine that we were to run K such experiments, measuring the random variables $\text{error}_{\mathcal{D}_j}[h]$; $j = 1, 2, \ldots, K$. The average over the K experiments:

$$\text{error}_{\mathcal{D}}[h] = \mathbb{E}_{\mathcal{D}}\{\text{error}_{\mathcal{D}_j}[h]\} \tag{2.7}$$

where $\mathbb{E}_{\mathcal{D}}\{\cdot\}$ denotes the expectation or ensemble average.

Bias and variance are most easily understood in the context of regression (numeric prediction). It is convenient to consider the particular case of a hypothesis function trained using the risk function (2.4), although our conclusions will be much more general.

Let us suppose that there is a true (yet unknown) function $f(\mathbf{x})$ possessing continuous-valued output y with noise, and we try to estimate this function on the basis of N samples in set $\mathcal{D}_j$ generated from $f(\mathbf{x})$.

The regression function estimated is denoted $h(\mathbf{x}; \mathcal{D}_j)$, and we are interested in the dependence of this approximation on the training set $\mathcal{D}_j$. Due to random variations in the selection of datasets $\mathcal{D}_j$; $j = 1, \ldots, K$, for some datasets, the approximation will be excellent, while for other datasets of the same size, approximation will be poor. The natural measure of the effectiveness of the estimator can be expressed as its mean-square deviation (refer to Eqn (2.6)) from the desired optimal: $[h(\mathbf{x}; \mathcal{D}_j) - f(\mathbf{x})]^2$.

A measure of how close the mapping function $h(\mathbf{x}; \mathcal{D}_j)$ is to the desired one is, therefore, given by the error function,

$$\text{error}_{\mathcal{D}_j}[h] = [h(\mathbf{x}; \mathcal{D}_j) - f(\mathbf{x})]^2 \tag{2.8}$$

The value of this quantity will depend on the particular dataset $\mathcal{D}_j$ on which it is trained. We write the average over the complete ensemble of datasets as,

$$\text{error}_{\mathcal{D}}[h] = \mathbb{E}_{\mathcal{D}}\{[h(\mathbf{x}; \mathcal{D}_j) - f(\mathbf{x})]^2\} \tag{2.9}$$

A non-zero error can arise for essentially two reasons:

1. It may be that the hypothesis function $h(\cdot)$ is on average, different from the regression function $f(\mathbf{x})$. This is called *bias*.
2. It may be that the hypothesis function is very sensitive to the particular dataset $\mathcal{D}_j$, so that for a given $\mathbf{x}$, it is larger than the required value for some datasets, and smaller for other datasets. This is called *variance*.

We can make the decomposition into bias and variance explicit by writing (2.8) in somewhat different, but mathematically equivalent form.

$$\begin{aligned}
\text{error}_{\mathcal{D}_j}[h] &= [h(\mathbf{x}; \mathcal{D}_j) - f(\mathbf{x})]^2 \\
&= [h(\mathbf{x}; \mathcal{D}_j) - \mathbb{E}_{\mathcal{D}}\{h(\mathbf{x}; \mathcal{D}_j)\} + \mathbb{E}_{\mathcal{D}}\{h(\mathbf{x}; \mathcal{D}_j\} - f(\mathbf{x})]^2 \\
&= [h(\mathbf{x}; \mathcal{D}_j) - \mathbb{E}_{\mathcal{D}}\{h(\mathbf{x}; \mathcal{D}_j)\}]^2 + [\mathbb{E}_{\mathcal{D}}\{h(\mathbf{x}; \mathcal{D}_j)\} - f(\mathbf{x})]^2 + \\
&\quad 2[h(\mathbf{x}; \mathcal{D}_j) - \mathbb{E}_{\mathcal{D}}\{h(\mathbf{x}; \mathcal{D}_j)\}]\,[\mathbb{E}_{\mathcal{D}}\{h(\mathbf{x}; \mathcal{D}_j)\} - f(\mathbf{x})]
\end{aligned} \tag{2.10}$$

In order to compute the expression in (2.10), we take the expectation of both sides over the ensemble of datasets $\mathcal{D}$.

$$\begin{aligned}
\text{error}_{\mathcal{D}}[h] &= \mathbb{E}_{\mathcal{D}}\{[h(\mathbf{x}; \mathcal{D}_j) - f(\mathbf{x})]^2\} \\
&= \mathbb{E}_{\mathcal{D}}\{[h(\mathbf{x}; \mathcal{D}_j) - \mathbb{E}_{\mathcal{D}}\{h(\mathbf{x}; \mathcal{D}_j)\}]^2\} + \mathbb{E}_{\mathcal{D}}\{[\mathbb{E}_{\mathcal{D}}\{h(\mathbf{x}; \mathcal{D}_j)\} - f(\mathbf{x})]^2\} + \\
&\quad 2\mathbb{E}_{\mathcal{D}}\{[h(\mathbf{x}; \mathcal{D}_j) - \mathbb{E}_{\mathcal{D}}\{h(\mathbf{x}; \mathcal{D}_j)\}]\,[\mathbb{E}_{\mathcal{D}}\{h(\mathbf{x}; \mathcal{D}_j)\} - f(\mathbf{x})]\}
\end{aligned} \tag{2.11}$$

Note that the third term on the right hand side of (2.11) vanishes (Help: $\mathbb{E}_{\mathcal{D}}\{f(\mathbf{x})\} = f(\mathbf{x})$; $\mathbb{E}_{\mathcal{D}}\{\mathbb{E}_{\mathcal{D}}\{h(\mathbf{x}; \mathcal{D}_j)\}\} = \mathbb{E}_{\mathcal{D}}\{h(\mathbf{x}; \mathcal{D}_j)\}$) and we are left with,

$$\begin{aligned}\text{error}_{\mathcal{D}}[h] &= \mathbb{E}_{\mathcal{D}}\{[h(\mathbf{x};\,\mathcal{D}_j) - f(\mathbf{x})]^2\} \\ &= \underbrace{[\mathbb{E}_{\mathcal{D}}\{h(\mathbf{x};\,\mathcal{D}_j)\} - f(\mathbf{x})]^2}_{(\text{bias})^2} + \underbrace{\mathbb{E}_{\mathcal{D}}\{[h(\mathbf{x};\,\mathcal{D}_j) - \mathbb{E}_{\mathcal{D}}\{h(\mathbf{x};\,\mathcal{D}_j)]^2\}}_{\text{variance}}\end{aligned} \tag{2.12}$$

The *bias* measures the level to which the average (over all datasets) of the hypothesis function is different from the desired function $f(\mathbf{x})$. The *variance* is a measure of the level to which the hypothesis function $h(\mathbf{x};\,\mathcal{D}_j)$ is sensitive to the specific selection of the dataset.

The true (but unknown) function we wish to approximate is $f(\mathbf{x})$. We have dataset $\mathcal{D}$ of N samples having continuous-valued output y with noise: $y = f(\mathbf{x}) + \varepsilon$ where ε represents noise. We seek to find an approximate function. $\hat{y} = h(\mathbf{x};\,\mathbf{w})$ such that mean-square-error is minimized ($\mathbf{w}$ represents adjustable parameters of hypothesis function $h(\cdot)$). To assess the effectiveness of the learned model $h(\cdot)$, we consider $\mathcal{D}_j$; $j = 1, \ldots, K$, training sets.

From Eqn (2.12), we observe that the bias term reflects the approximation error that the hypothesis $h(\cdot)$ is expected to have on average when trained on datasets of same finite size. In practice, it is often necessary to iterate in order to adjust the parameters of the hypothesis function to the problem specifics, so as to reduce the approximation error. In general, *higher the complexity* of the hypothesis function (more flexible function with large number of adjustable parameters), the *lower is the approximation error*.

From Eqn (2.12), we also observe that the variance term reflects the capability of the trained model on a data sample to *generalize* to other data samples. Low variance means that the estimate of $f(\mathbf{x})$ based on a data sample does not change much on the average as the data sample varies. Unfortunately, the *higher the complexity* of the hypothesis function (which results in low bias/low approximation error), the *higher is the variance*.

We see that there is a natural trade-off between bias and variance. Procedures with increased flexibility to adopt to the training data tend to have lower bias but higher variance. Simpler (inflexible; less number of free parameters) models tend to have higher bias but lower variance. To minimize the overall mean-square-error, we need a hypothesis that results in *low bias* and *low variance*. This is known as *bias-variance dilemma* or *bias-variance trade-off* [35].

The design of a good hypothesis through the study of trade-off between bias and variance in the context of a finite data-sample size, is one of the main themes of modern automatic learning theory. In general, finding an optimal bias-variance trade-off is hard, but acceptable solutions can be found. New statistical learning techniques are being developed with promising results. The next section introduces this approach. This is then followed by procedures for avoiding overfitting, and heuristic search methods.

Example 2.1

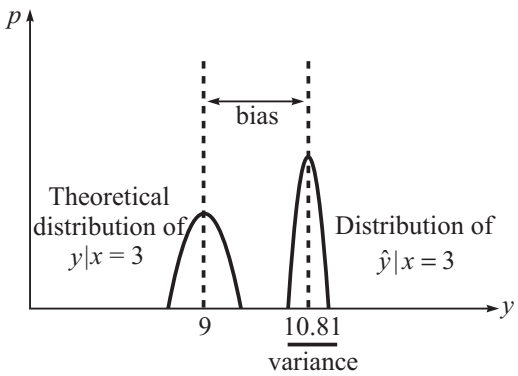
Suppose that the true target function is of quadratic form in one variable with noise: $y = x^2 + \varepsilon$; ε is noise term; x takes on values on the interval $(0, 5)$ [3]. Assume that a data sample is generated from this model.

Consider a linear hypothesis function; slope and intercept are the two adjustable parameters of the linear curve that we fit to the training data. If the experiment is repeated many times, it can be

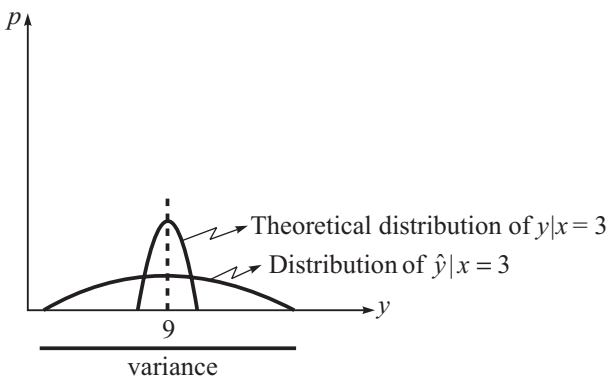
seen that estimates are scattered around $\hat{y} = 10.81$, given $x = 3$. According to the original model, the value of y, given $x = 3$, is scattered around $y = 9$. This difference is the *bias* (a sort of persistent error because of our choice of a simpler model), i.e., a linear model instead of a more complex polynomial model. The bias is depicted in Fig. 2.3(a).

Now, let us fit a complex model, say a 5^{th} order polynomial, having many adjustable parameters. The distribution of the estimate $\hat{y}$ at $x = 3$ from different random samples will look like the one shown in Fig. 2.3(b). We note that the bias is now zero but variability in prediction is very high. We see from Fig. 2.3(a) that when we fit a model which is simpler (compared to the model which generated the data) bias is high but variability is less, whereas when we fitted a model of higher order, bias is negligible but variance is very high. This is bias-variance dielemma.

Figure 2.4 shows how bias and variance vary with the complexity of the model. For increasing complexity of the model, the bias component of the error decreases, but the variance component increases. For a certain model complexity, the total error is minimal. Different models result in different bias-variance trade-off curves.



(a) Linear curve fitting



(b) Higher-order polynomial fitting

Figure 2.3 Illustration of bias and variance in the domain of regression

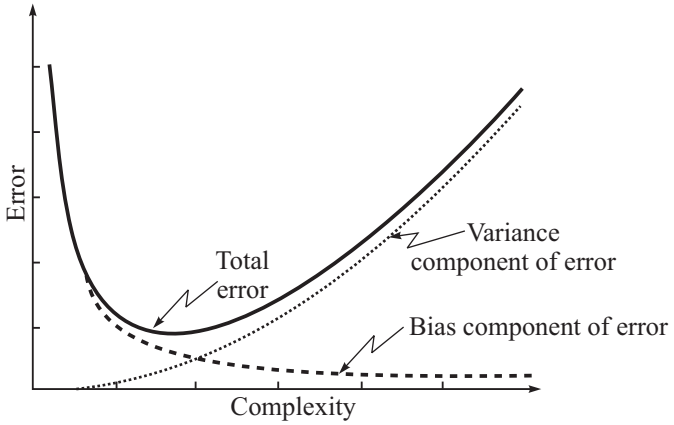


Figure 2.4 Bias-variance trade-off

2.3 WHY LEARNING WORKS: COMPUTATIONAL LEARNING THEORY

The main question posed in Section 2.1 has not yet been answered:

Is it possible to say with certainty that the learning algorithm has given rise to a theory, which can predict the future with accuracy? Formally, how is it possible to know that the hypothesis $h(\cdot)$ is close to the true function $f(\cdot)$ if we do not know what f is?

Till we know the answers to these questions, machine learning's own success will be baffling.

The approach taken in this section is based on the *statistical learning theory* (also known as *computational learning theory*). Instead of a complete treatment, we attempt to provide the main insights in a non-technical way, to offer the reader with certain intuition as to how various pieces of the puzzle fit together [3].

What is required for empirical risk minimization over a given dataset $(\mathbf{x}^{(i)}, y^{(i)})$; $i = 1, \ldots, N$, to work (bias and variance low at the same time)? Let us examine the possibility of success as $N \to \infty$ [53]. The classical *Law of Large Numbers* ensures that empirical risk (refer to Eqn (2.4))

$$R_{\text{emp}}[f] = \frac{1}{N} \sum_{i=1}^{N} L(y^{(i)}, f(\mathbf{x}^{(i)}))$$

converges probabilistically to the true expected risk (refer to Eqn (2.3))

$$R[f] = \int_{\mathbf{X} \times Y} L(y, f(\mathbf{x}))\, p(\mathbf{x}, y) d\mathbf{x}\, dy$$

as $N \to \infty$:

$$|R_{\text{emp}}[f] - R[f]| \xrightarrow{P} 0 \text{ as } N \to \infty, \text{ for each fixed } f. \tag{2.13}$$

This means that $\forall\ \delta > 0$, we have (P denotes the probability)

$$\lim_{N \to \infty} P\{|R_{\text{emp}}[f] - R[f]| > \delta\} = 0 \tag{2.14}$$

This statement applies as well for the parameters **w** which define function f (f: $f(\mathbf{x}, \mathbf{w})$).

$$\lim_{N\to\infty} P\{|R_{\text{emp}}[\mathbf{w}] - R[\mathbf{w}]| > \delta\} = 0 \ \forall\ \delta > 0 \tag{2.15}$$

At first sight, it seems that empirical risk minimization should work (as $N\to\infty$) in contradiction to our fears expressed in bias-variance dilemma. Unfortunately, it does not work. The convergence in probability does not guarantee that the function f^{emp} that minimizes $R_{\text{emp}}[f]$ converges to the true function f^{opt} that minimizes $R[f]$.

Simplified depiction of the convergence of empirical risk to true risk is given in Fig. 2.5. x-axis gives one-dimensional representation of function class, and y-axis gives risk (error). Downward arrow indicates that $R_{\text{emp}}[f]$ converges to $R[f]$ for fixed f as $N \to \infty$. f^{opt} is the function that gives best attainable true risk, $R[f^{\text{opt}}]$. There is no guarantee that as $N \to \infty, f \to f^{\text{opt}}$.

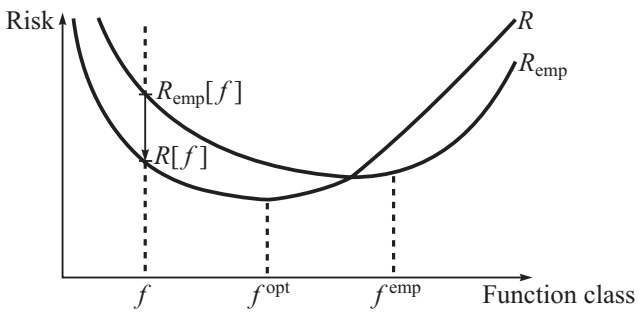


Figure 2.5 Convergence of empirical risk to true risk

We have to do more than just solving empirical risk minimization problem. What is the catch?

What is needed is *asymptotic consistency* or *uniform convergence*. This property of consistency is defined in the key learning theorem for bounded loss functions [37, 38], which states that empirical risk minimization principle is consistent if and only if *empirical risk converges uniformly to the true risk* in the following probabilistic sense:

$$\lim_{N\to\infty} P\{\sup_{f\in\mathcal{F}} |R[f] - R_{\text{emp}}[f]| > \delta\} = 0 \ \forall\ \delta > 0 \tag{2.16}$$

$\mathcal{F}$ is the set of functions that the learning machine can implement.

Equivalently, in terms of parameters of function f, we can express asymptotic consistency condition as,

$$\lim_{N\to\infty} P\{\sup_{\mathbf{w}} |R[\mathbf{w}] - R_{\text{emp}}[\mathbf{w}]| > \delta\} = 0 \ \forall\ \delta > 0 \tag{2.17}$$

sup denotes the supremum (least upper bound) of the set $S = |R(\mathbf{w}) - R_{\text{emp}}(\mathbf{w})|$; it is defined by the smallest element b such that $b \geq s$ for all $s \in S$.

Equation (2.17) asserts that the consistency is determined by the function from the set of approximating functions $\mathcal{F}$ that gives a worst-case measure (maximum value of $(R - R_{\text{emp}})$), i.e., that provides the largest error between the empirical risk and the true risk. Consistency, and thus learning, crucially depends on the set of functions $\mathcal{F}$. If we consider the set of all possible functions, learning is not possible.

It turns out that *without restricting the set of admissible functions, empirical risk minimization is not consistent*.

We now address whether there are properties of learning machines, i.e., of sets of functions, which *ensure* uniform convergence of risks.

VC (Vapnik and Chervonenkis) Theory [38] shows that it is imperative to restrict the set of functions $\mathcal{F}$ from which f is chosen to one that has a *capacity* suitable for the amount of available training data. The best-known capacity concept of VC theory is *VC Dimension*; it restricts the function class $\mathcal{F}$ to a finite one for the purpose of bounding (2.16).

Despite the fact that the VC dimension is very important, the unfortunate reality is that its analytic estimations can be used only for the simplest sets of functions. The calculation of VC dimension for nonlinear function classes is very difficult task, if possible at all. Also, the nature of learning problem (classification/regression) affects estimation complexity. But even when the VC dimension cannot be calculated directly, the results of VC theory are relevant for an introduction of *structure* on the class of approximating functions.

The VC theory offers *bounds* on the generalization error. Minimization of these bounds is dependent on the empirical risk and the capacity of the function class. A result from VC theory for binary classification states that with a probability at least $(1 - \delta)$,

$$R[f] \leq R_{\text{emp}}[f] + \sqrt{\frac{1}{N}\left(h_c\left(\ln\frac{2N}{h_c}+1\right)+\ln\frac{4}{\delta}\right)} \tag{2.18}$$

In case of binary classification, the VC dimension (capacity) h_c of a set $\mathcal{F}$ of functions is defined as the number of points that can be separated (shattered) in all possible ways.

Bounds like (2.18) can be used to justify induction principle different from the empirical risk minimization principle. Vapnik and Chervonenkis proposed minimizing the right hand side of the bound (2.18), rather than the empirical risk. The confidence term in the present case, $\sqrt{\frac{1}{N}\left(h_c\left(\ln\frac{2N}{h_c}+1\right)+\ln\frac{4}{\delta}\right)}$, then ensures that the chosen function not only leads to small risk, but also comes from a function class with small capacity. The capacity term is a property of the function class $\mathcal{F}$, and not of any individual function f. Thus, the bound cannot simply be minimized over choices of f. Instead, we introduce a so-called *structure* on $\mathcal{F}$, and minimize over elements of the structure. This leads to an induction principle called *structural risk minimization*. We leave out the technicalities involved; the main idea is as follows:

The function class $\mathcal{F}$ is decomposed into a nested sequence of subsets of increasing capacity. The structural risk minimization principle picks a function f^* which has small training error (R_{emp}), and comes from an element of the structure that has low capacity h_c; thus minimizing a risk bound of the type (2.18). This is graphically depicted in Fig. 2.6.

Structural risk minimization is a novel inductive principle for learning from finite training data sets. It is very useful for dealing with small samples. The basic idea is to choose from a large number of candidate learning machines, a machine of the right complexity to describe training data pairs. This is done by restricting the hypothesis space of approximating functions and simultaneously controlling their complexity.

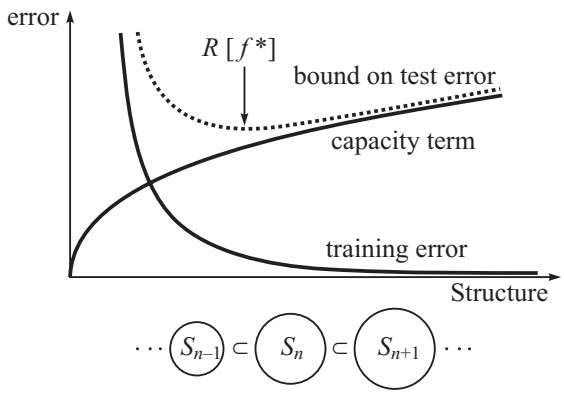


Figure 2.6 Structural risk minimization

Another popular formalism of computational learning theory is the *Probably-Approximately-Correct* (PAC) learning. Let us examine *sample complexity* in PAC framework.

The sample complexity of a learning algorithm is defined as the smallest number of samples required for learning by $\mathcal{F}$, that achieves a given approximation accuracy ε with a probability $(1 - \delta)$. Accuracy ε is defined as follows:

$$\text{error}_P[f^{\text{emp}}] \leq \varepsilon + \text{error}_{\mathcal{D}}[f^{\text{emp}}] \tag{2.19}$$

where $\mathcal{D}$ is the set of examples available to the machine, and P represents the probability distribution over the entire set of patterns.

In terms of cardinality of $\mathcal{F}$, denoted $|\mathcal{F}|$, the sample complexity of a learning algorithm N_{PAC} (smallest number of training examples required for learning), is at most

$$N_{\text{PAC}} = \frac{1}{2\varepsilon^2}\left(\ln|\mathcal{F}| + \ln\left(\frac{1}{\delta}\right)\right) \tag{2.20}$$

For most hypothesis spaces for binary classification, $N \geq N_{\text{PAC}}$ gives a good result. Bounds for other problems have also been found.

Note that the bound (2.20) can be a substantial overestimate. The weakness of this bound is mainly due to $|\mathcal{F}|$ term. In fact, much tighter bound is possible using VC dimension:

$$N \geq \frac{1}{\varepsilon}\left[4\log_2\left(\frac{2}{\delta}\right) + 8h_c\log_2\left(\frac{13}{\varepsilon}\right)\right] \tag{2.21}$$

How about the overall *computational complexity*? That will depend, of course, on specific learning algorithm.

2.4 OCCAM'S RAZOR PRINCIPLE AND OVERFITTING AVOIDANCE

Occam's Razor Principle

The Franciscan monk, Willam of Occam, was born in 1280, much before the invention of modern statistics. His name is linked with machine learning through the basic idea 'The *simpler*

explanations are more reasonable, and any unnecessary complexity should be shaved off'. This line of reasoning—Occam's razor principle—is his contribution to the domain of machine learning.

If there are two algorithms and both of them perform equally well on the training set, then according to Occam's razor principle, the *simpler* algorithm can be expected to do better on a test set; 'simpler' may imply needing lesser parameters, lesser training time, fewer attributes for data representation, and lesser computational complexity. Our design methodology itself imposes a bias toward 'simple' solutions in applied machine learning. We usually stop looking for a design when the solution is 'good enough', not necessarily the optimal one.

Occam's razor principle suggests hypothesis functions that avoid *overfitting* of the training data.

Overfitting

In machine learning jargon, a learning machine is said to *overfit* the training examples if certain other learning machine that fits the training examples less well, actually performs better over the total distribution of patterns (i.e., including patterns beyond the training set).

Figure 2.7 depicts the effect of overfitting in a typical application of machine learning. The horizontal axis of the plot shows how *complex* the classifier is (more number of weights in the neural network, large number of nodes in the decision free, more number of rules in a fuzzy logic model, etc.). The vertical axis is an indicator of how *accurate* the forecasts made by the classifier are. The solid line depicts how accurate the classifier is over the training examples, while the broken line shows the accuracy measured over an independent set of test examples (not included in the training set, but class labels known). Predictably, the accuracy of the classifier over training examples increases monotonically as the classifier becomes more complex. But, the accuracy over the independent test examples first grows, then reduces.

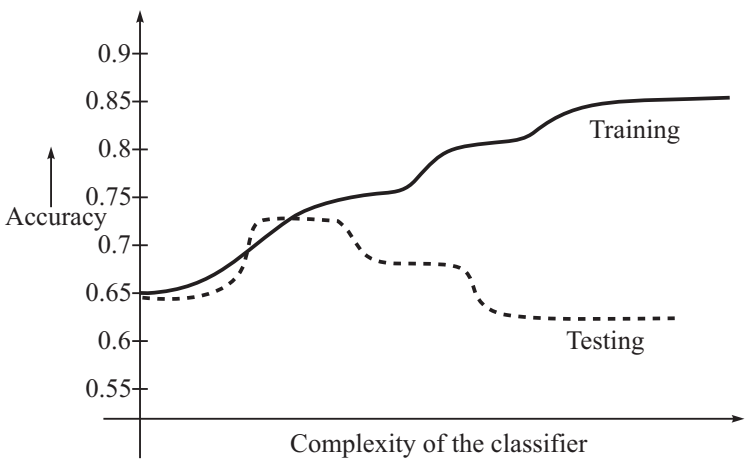


Figure 2.7

Learning is most reliable when the training examples follow a distribution similar to that of future instances unseen during training. If training experience consists of data that lies in a region of state space, then that region must be fully representative of situations over which our learning

machine will later be used. Machine learning tasks based on real-world data are unlikely to find this assumption tenable. The real-world data tend to be incomplete, noisy, and inconsistent. Classifier design for such a data results in more complex classifier, if a near-perfect fit on the training data is the goal. To illustrate, consider the effect of a training example with incorrect class label. A classifier that fits the data perfectly will be more complex; complexity in design is simply a consequence of fitting the noisy training example. We expect another classifier that is simpler but does not fit training data perfectly, to outperform this classifier over subsequent data drawn from the same pattern distribution.

Even in noise-free training data, overfitting can occur due to insufficient number of training examples. Coincidental regularities in this dataset lead to a risk of inducing an approximation function with poor generalization performance.

Section 2.1 clearly states that there exist no problem-independent reasons to favor one algorithm over another. Then why is it required to avoid overfitting? Why do we usually promote simpler algorithms which have lesser features and parameters?

In fact, methods that help avoid overfitting are not intrinsically advantageous. They can provide benefit only if they are able to resolve the problems for which they are used. Empirical success depends on the match of the learning algorithm with the problem—not only the imposition of avoiding overfitting. The frequent empirical successes of overfitting avoidance indicate that the classes of real-world problems that have been addressed till now have particular properties that offer a match.

2.5 HEURISTIC SEARCH IN INDUCTIVE LEARNING

Before we move forward, a review of our discussion so far in this chapter, will be helpful.

Review

Machine learning is aimed at building a statistical model of the process which generates the data, and not to learn an accurate representation of the training data itself. This aim is important for the machine to make good predictions for new inputs, i.e., to demonstrate good generalization.

The success of learning very much depends on hypothesis space complexity and sample complexity. The two are interdependent. The goal is to find a hypothesis (model) simplest in terms of complexity and best in terms of empirical error on the data. Such a choice is expected to give good generalization performance. Basically, *finding a hypothesis function of complexity consistent with the given training data, is the search problem in hand.*

Substantial insight into this phenomenon was given in Section 2.2, which introduced the idea of the *bias-variance trade-off*, wherein the mean-square error is decomposed into the sum of the bias-squared plus variance. A function, which is too simple or too inflexible (number of free parameters too less) will have a huge bias, whereas one which is highly complex or too flexible (too many parameters will help fit too much of noise present in the training data) will possess a huge variance. Best generalization is achieved through the best compromise between the contradictory requirements of small bias and small variance.

In order to find the optimum balance between bias and variance, we need to have a way of controlling the effective complexity of the hypothesis functions. Section 2.3 highlighted the *structural-risk minimization principle* to achieve this objective. *VC dimension* was introduced as an important measure of function complexity. Structural-risk minimization uses a set of hypothesis functions ordered in terms of their complexities; hypothesis selection then corresponds to finding the function simplest in terms of complexity and best in terms of empirical error on the data. Relationship between the hypothesis function complexity and sample complexity given by PAC (Probably-Accurately-Correct) model of learning was also highlighted in this section.

What has been the conclusion?

The mathematical models of learning (VC model, PAC model) provide a platform for theoretical characterization of machine learning but lack (at least today) practical tools for real-life problems of varying complexities. Bias-variance trade-off is also a theoretical result because we lack practical tools to obtain optimum values of bias and variance. Machine learning community is, by and large, not using these models to build a learning machine for classification/regression problems; they depend on certain tools which appear to be *heuristic*, trial-and-error tools. In fact, the motivation for these tools comes from classical statistics and the computational learning theory. Practitioners today, more or less rely on the knowledge of learning domains and their experience with broad range of hypothesis functions. Occam's razor principle and overfitting avoidance, described earlier in Section 2.4, are extensively exploited. Heuristic strategies for overfitting avoidance: regularization, early stopping, pruning (appearing later in this section); and minimum description length principle (detailed in Section 3.9), are some of the heuristic tools explored by practitioners today.

An optimistic view point: the mathematical models give an upper bound on generalization error. We might hope that by our practical heuristic approach (or even mere luck), we would achieve good generalization with fewer training patterns than the number predicted using mathematical learning models developed in classical statistics and computational learning theory.

2.5.1 Search through Hypothesis Space

In this sub-section, we discuss heuristic search strategies. The word 'heuristic' indicates that strategies for the search problem cannot be precisely pre-defined. *Trial-and-error* is the approach of searching for a 'good enough' solution. This approach, however, gets more organized if we exploit the prior knowledge about the learning domains and experience with a broad range of algorithms.

Machine learning entails looking through a space of probable hypotheses to decide on one that fits the data observed. and any previous knowledge held by the learner. Space of probable hypotheses is infinite in theoretical terms. The learning task, therefore, is to explore a widespread space to find the hypothesis that is most consistent with the existing training examples. Luckily, there are merely a few possible candidate hypotheses in use today based on empirical experience (refer to Section 3.5.3).

Applied machine learning organizes the search as per the following two-step procedure (refer to Section 3.5).

(1) The search is first focused on a *class* of the possible hypotheses, chosen for the learning task in hand. Prior knowledge and experience are helpful in this selection. Different classes are appropriate for different kinds of learning tasks.

(2) For each of the classes, the corresponding learning algorithm organizes the search through all possible structures of the learning machine.

In the following paragraphs, we describe principal techniques used in heuristic search to optimize hypothesis complexity for a given training dataset.

Regularization: The regularization model promotes smoother functions by creating a new criterion function that relies not only on the training error, but also on algorithmic intricacy. Particularly, the new criterion function punishes extremely complex hypotheses; looking for the minimum in this criterion is to balance error on the training set with complexity. Formally, it is possible to write the new criterion as a sum of the error on the training set plus a *regularization term*, which depicts constraints or sought after properties of solutions:

$$\bar{E} = E + \lambda \Omega \tag{2.22}$$

= error on data + $\lambda \times$ hypothesis complexity where λ gives the weight of penalty

The second term penalizes complex hypotheses with large variance. When we minimize augmented error function instead of the error on data only, we penalize complex hypotheses and thus decrease variance. When λ is taken too large, only very simple functions are allowed and we risk introducing bias. λ is optimized using cross-validation (discussed later in this chapter).

We consider here the example of neural network hypotheses class (discussion coming up in Chapter 5). The hypothesis complexity may be expressed as,

$$\Omega = \frac{1}{2} \sum_{l,j} w_{lj}^2 \tag{2.23}$$

The regularizer encourages smaller weights w_{lj}. For small values of weights, the network mapping is approximately linear. Relatively large values of weights lead to overfitted mapping with regions of large curvature.

Early Stopping: The training of a learning machine corresponds to iterative decrease in the error function defined as per the training data. During a specific training session, this error generally reduces as a function of the number of iterations in the algorithm. Stopping the training before attaining a minimum training error, represents a technique of restricting the effective hypothesis complexity.

Pruning: An alternative solution that sometimes is more successful than early stopping the growth (complexity) of the hypothesis is pruning the full-grown hypothesis that is likely to be overfitting the training data. Pruning is the basis of search in many decision-tree algorithms (discussion coming up in Chapter 8); weakest branches of large tree overfitting the training data, which hardly reduce the error rate, are removed.

2.5.2 Ensemble Learning

For the given amount and quality of training data, the output of one hypothesis function may be inappropriate for the problem at hand. The ideal model to make more reliable decisions is to create

a combination of outputs of many different hypotheses. Many machine learning algorithms do this by learning an *ensemble* of hypothesis and employing them in a combined form [39–41]. *Bagging* and *boosting* are the most frequently used among these schemes. These general methods may be applied to classification (categorization) and regression (numeric prediction) problems; and, they frequently increase predictive performance over a single hypothesis.

By combining the decisions of various hypotheses, we amalgamate the different outputs into a single prediction. For classification problems, it is done through voting (may be a weighted vote), whereas in case of regression problems, the average is computed (may be a weighted average). Conventionally, various hypotheses in the ensemble possess the same general form—for instance, all neural networks, or all decision trees—simply the final parameter values are different. The difference in parameter values is because of the fact that their training patterns differ—each one handles a certain percentage of data accurately. If the hypotheses complement each other, it would be ideal, as each is a specialist for a part of the domain wherein the other hypotheses fail to do well.

In the **bagging technique**, individual approaches are constructed separately, whereas in **boosting**, each new model is impacted by the performance of those built earlier. In boosting, we first make a model with accuracy on the training set greater than average, and then add new component classifiers to make an ensemble whose joint decision rule possesses a high level of accuracy on the training set. In a scenario such as this, we claim that performance of the learning algorithm has been 'boosted'. The method is capable of training successive classifiers with a subset of 'very informative' training data, considering the present set of component classifiers. **Ada Boost** (an abbreviation for Adaptive Boosting) is a widely used algorithm for boosting.

Bagging is the most straightforward and basic technique of pooling or integrating the outputs of component classifiers.

Yet another ensemble technique is known as **random forests**. Suppose each classifier in the ensemble is a *decision tree* classifier. Therefore, the set of such classifiers gives rise to a 'forest'. The individual decision trees are created with the help of a random choice of attributes at each node to identify the split (Chapter 8). More formally, each tree is dependent on the values of a random vector independently sampled and with the same distribution for all trees in the forest. During classification, each tree votes and the class that is highly popular is returned. You can build random forests with the help of bagging in tandem with the random attribute selection.

Class-Imbalanced Problems: Ensemble methods have been used to solve *class-imbalanced* problems. What are class-imbalanced problems?

Two-class data are *class-imbalanced* if the primary class of interest is represented by just a few samples in the dataset, whereas the majority of the samples represent the other class.

For multiclass-imbalanced data, the data distribution of each class is substantially different, where again, the primary class or classes of interest are represented by merely a few samples. The class-imbalanced problem is very similar to *cost-sensitive learning, wherein the costs of errors per class, are unequal*. Metrics for assessment of accuracy of learning machines on the basis of cost-sensitive datasets will be presented in Section 2.8.

The general approaches for improvement of the classification performance of class-imbalanced data are:

(i) Oversampling
(ii) Undersampling
(iii) Threshold moving
(iv) Ensemble methods

Oversampling and undersampling alter sample distribution in the training set; threshold-moving impacts the manner in which the model makes decisions while classifying new data, and ensemble techniques follow the methods discussed earlier.

2.5.3 Evaluation of a Learning System

Before being used, a machine learning system should be evaluated in many aspects, intrinsically linked to what the system is meant for. Some important aspects of evaluation are:

Accuracy: Inductive learning is based on empirical data. The learning system extracts knowledge from the training data. The learned knowledge should be general enough to deal with unknown data. The *generalization* capability of a learning system is an index of *accuracy* of the learning machine.

A machine learning system will not be accepted by its intended users unless it has demonstrated its accuracy in problem solving. This is the single most important objective of learning. We will shortly describe the *accuracy and error measures* and the methods of *error estimation* to quantitatively measure the generalization capability of a learning machine.

Robustness: 'Robustness' means that the machine can perform adequately under all circumstances, including the cases when information is corrupted by noise, is incomplete, and is interfered with irrelevant data. All these conditions seem to be part of the real world (discussed in Chapter 7), and must be considered while evaluating a learning system. Robustness is typically assessed with a series of synthetic datasets representing increasing degrees of inconsistencies in data.

Computational Complexity and Speed: Computational complexity of a learning algorithm and learning speed determine the *efficiency* of a learning system: how fast the systems can arrive at a correct answer, and how much computer memory is required. We know how important speed is in real-time situations.

Online Learning: An online learning system can continue to assimilate new data. This feature is essential for a learning system which continues to receive inputs from a real-time environment.

Interpretability: This is the level of understanding and insight offered by a learning algorithm. Interpretability is subjective and hence, tougher to evaluate. Interpretation is easy in fuzzy logic systems and decision trees, but still their interpretability may decrease with an increase in their complexity.

Scalability: This is the capability to build the learning machine considering the huge amounts of data. Typically, the assessment of scalability is done with a series of datasets of ascending size.

2.6 ESTIMATING GENERALIZATION ERRORS

The success of learning depends on the hypothesis space complexity and sample complexity. The two are interdependent. The goal is to find a function simplest in terms of complexity and best in terms of empirical error on the data. Such a choice is expected to give good generalization performance. Basically, *finding a hypothesis function of complexity consistent with the given training data* is the problem in hand.

So how is the learning scheme likely to perform in the future on new (unseen) data? We are not interested in the past performance on old (training) data. We are already aware of the output of each instance in the *training set*, which is the reason why it can be used for training. The actual question is: Is the performance on training data likely to be a proper indicator of the performance on unseen data, which is not employed in training? The answer is in the negative. To predict the performance of a learning scheme on new data, the assessment of the success rate (error rate) on a dataset— that had no role in forming the leaning model for classification or numeric prediction (metrics for *error rate* are given in the next two sections)—is required. This independent dataset is known as the *test set*. The assumption is that both the training set and the test set comprise representative samples of the underlying distribution.

Importantly, the test data is in no way employed to build the learning model. Some learning schemes include two stages for constructing a model—one to come up with the elementary structure, and the second to optimize parameters involved in that structure. Separate sets of data may be required in the two stages. In such scenarios, we frequently consider three datasets: the *training data*, the *validation data*, and the *test data*. The training data is employed to create the structure of the learning model. The validation data is employed to optimize parameters of that model, or to choose a specific model if the training data has been made use of to make several learning models. Then the test data is employed to compute the error rate of the final optimized selected model. Each of the three sets has to be selected independently: the validation set should be different from the training set to get perfect performance in the optimization or selection stage, and the test data has to be different from both to get a reliable estimate of the true error rate.

If a lot of data is available, a reasonable large sample is taken up for training, then another independent reasonable large sample is taken up for testing. What to do when vast supply of data is not available. The question of predicting performance on the basis of *limited data* is interesting, yet controversial.

We will take a look at various methods—the probable techniques for success in most practical limited-data situations.

2.6.1 Holdout Method and Random Subsampling

In the *holdout* technique, some amount of data is earmarked for the purpose of testing, while the remainder is employed for training (and a portion of it is kept aside for validation, if needed). Practically speaking, it is common to hold out one-third of the data for testing and use the remainder for training.

In order to divide dataset $\mathcal{D}$ into training and test sets, we can *arbitrarily* sample a set of training examples from $\mathcal{D}$, and keep aside the remaining for testing. If the data is collected over time (*time-series data*), then we can make use of the earlier part to train and the latter part of the data

for the purpose of testing. In various applications, this process of dividing time-series data is more suited as the learning machine is made use of in the real world; the unseen data belongs to the future.

The samples used to train and test have to represent the underlying distribution for the problem area. Generally speaking, it is not possible to tell whether a sample is representative or not because we do not know the distribution. However, there is a simple check that may be worth while. In case of classification problems, each class in the full dataset needs to be represented in about the right proportion in the training and test sets. The proportion of class-data in training, testing, and full datasets should more or less be same. To make sure this happens, random sampling should be performed in a manner that will guarantee that each class is properly represented in training as well as test sets. This process is known as *stratification*.

Even though *stratified holdout* is generally well worth doing, it offers merely a basic safeguard against irregular representation in training and test sets.

A more general way to alleviate any bias resulting from the specific sample selected for holdout is *random subsampling*, wherein the holdout technique is iterated K times with various arbitrary samples. The accuracy estimate on the whole is considered as the average of the accuracies got from each repetition.

In a single holdout process, we may look at considering swapping the roles of training and test data—that is, train the machine on the test data, and estimate the success rate making use of the training data—and average the two results, thereby decreasing the effect of uneven distribution in training and test sets. This is, however, useful with a 50:50 split of the full data between training and test sets, which is usually not ideal—it is ideal to employ more than 50 per cent of the data for training even at the cost of test data. But, a simple variant becomes the basis of a powerful statistical method, known as *cross-validation*.

2.6.2 Cross-validation

A commonly used technique for forecasting the success rate of a learning method, taking into account a fixed data sample, is the *K-fold cross-validation*. Another estimate prevalent is the *leave-one-out* cross-validation. The description of these two cross-validation techniques follows.

K-Fold Cross-validation

In K-fold cross-validation, the given data $\mathcal{D}$ is randomly divided into K mutually exclusive subsets or 'folds', $\mathcal{D}_k$; $k = 1, \ldots, K$, each of about equal size. Training and testing is done K times. In iteration k, partition $\mathcal{D}_k$ is set aside for testing, and the remainder of the divisions are collectively employed to train the model. That is, in the first iteration, the set $\mathcal{D}_2 \cup \mathcal{D}_3 \cup \cdots \cup \mathcal{D}_K$ serves as the training set to attain the first model, which is tested on $\mathcal{D}_1$; the second iteration is trained on $\mathcal{D}_1 \cup \mathcal{D}_3 \cup \cdots \cup \mathcal{D}_K$ and tested on $\mathcal{D}_2$; and so on.

If stratification is also used, it is known as *stratified K-fold cross-validation* for classification.

Ultimately, the K error estimates received from K iterations are averaged to give rise to an overall error estimate. Out of the 10 machines, the one with lowest error may be deployed.

$K = 10$ folds is the standard number employed to predict the error rate of a learning method. Why 10? Extensive tests on numerous datasets with various learning methods have revealed that

10 is about the right number of folds to achieve the best estimate of error. There is some theoretical proof also that backs up *10-fold cross-validation*. These arguments, although, cannot be said to be conclusive, 10-fold cross-validation is now the standard technique in practical terms.

When you look for a precise estimate, the normal process is to repeat the 10-fold cross-validation procedure 10 times and average the outcomes. This requires applying the learning algorithm 100 times on datasets that are all nine-tenths the size of the original.

Leave-One-Out Cross-Validation

This is an exceptional case of *K*-fold cross-validation wherein *K* is set to the number *N* of initial tuples. In other words, only a single sample is 'left out' for the test set in each iteration. The learning machine is trained on the remainder of the samples. It is judged by its accuracy on the left-out sample. The average of all outcomes of all *N* judgements in *N* iterations is taken, and this is the average which is representative of the error estimate.

The computational expense of this process is quite high as the whole learning process has to be iterated *N* times, and this is generally not feasible for big datasets. Nevertheless, leave-one-out seems to present an opportunity to squeeze the maximum out of a small dataset and obtain an estimate that is as precise as possible. This process disallows stratification.

2.6.3 Bootstrapping

The *bootstrap technique* is based on the process of *sampling with replacement*. In the earlier techniques, whenever a sample was used from the dataset to form a training or test set, it was never replaced. In other words, the same instance, which was once chosen could not be chosen again. However, most learning techniques can employ an instance several times, and it affects the learning outcome if it is available in the training set more than once. The concept of *bootstrapping* aims to sample the dataset by replacement, so as to form a training set and a test set.

There are many bootstrap techniques. The most popular one is the *0.632 bootstrap*, which works as follows:

A dataset of *N* instances is sampled *N* times, with replacements, to give rise to another new dataset of *N* instances, which is a *bootstrap sample*—a training set of *N* samples. As certain elements in the bootstrap sample will (almost certainly) be repeated, there will be certain instances in the original dataset $\mathcal{D}$ that have not been selected—these will be used as test instances. If we attempt this many times, on an average, 63.2% of the original data instances will result in the bootstrap sample and the remaining 36.8% will give rise to the test set (therefore, the name, 0.632 bootstrap).

Where does the figure, 63.2%, come from? The probability that a particular instance will be picked is 1/*N*; so the probability of not being picked is $\left(1-\frac{1}{N}\right)$. Number of picking opportunities is *N*, so the probability that an instance will not be picked during the whole sampling cycle is $\left(1-\frac{1}{N}\right)^N$. If *N* is large, the probability approaches $e^{-1} = 0.368$ (*e* is the base of natural algorithms that is, $e = 2.718$; and not the error rate). Thus, for a reasonably large dataset, the test set will

contain about 36.8% of the instances, and training set will contain about 63.2% of them. Some instances will be repeated in the training set, bringing it up to a total size of N.

Training a learning system on the training set and calculating the error over the test set will give a pessimistic estimate of the true error because the training set, although its size is N, nevertheless contains only 63.2% of the instances (In 10-fold cross-validation, 90% of the instances are used for training). To compensate for this, the bootstrap procedure combines the training error with the test error to give a final error estimate as follows:

$$\text{Error estimate} = 0.632 \times \text{Error given by test instances} + 0.368 \times \text{Error given by training instances} \quad (2.24)$$

Then, the whole bootstrap procedure is repeated several times, with different replacement samples for the training set, and the results are averaged.

Bootstrapping tends to be overly optimistic. It works best with small datasets.

2.7 METRICS FOR ASSESSING REGRESSION (NUMERIC PREDICTION) ACCURACY

A function,

$$f = \mathbf{X} \rightarrow Y; f(\mathbf{x}) = y$$

maps from $\mathbf{x} \in \mathbf{X}$ to $y \in Y$, where $\mathbf{X}$ is the *input space* and Y is the *output space* of the function. We assume here that $\mathbf{X} \subset \Re^n$, and $Y \subset \Re$. The data is available as samples $(\mathbf{x}, y)$ where the distribution of inputs $\mathbf{x}$ and the function f are both unknown.

The task is to find the model $h(\mathbf{x})$ that explains the underlying data, i.e., $h(\mathbf{x}) \simeq y$ for all samples $(\mathbf{x}, y)$. Equivalently, the task is to approximate function $f(\mathbf{x})$ with unknown properties by $h(\mathbf{x})$.

The term used in statistics for function description of data is *regression*. Also, since $h(\mathbf{x})$ predicts $y \in \Re$ for a given $\mathbf{x}$, the term *numeric prediction* is also in use. Throughout this book, the three terms: function approximation, numeric prediction, and regression, are used interchangeably without any discrimination.

The classic problem of approximation of multivariate function $f(\mathbf{x})$ is the determination of an approximating function $h(\mathbf{x}, \mathbf{w})$, having a fixed finite number of parameters w_j that are entries of the *weight vector* $\mathbf{w}$.

This section presents measures (metrics) for assessing how good or how accurate our regressor (i.e., approximating function $h(\mathbf{x}, \mathbf{w})$) is at predicting the continuous (numeric) response variable. Classical measures of performance are aimed at finding a model $h(\mathbf{x}, \mathbf{w})$ that fits the data well. However, those measures do not tell us much about the ability of the model to predict new cases. Using training data to derive a regressor and then to estimate the accuracy of the resulting learned model, can result in misleading overoptimistic estimates due to overspecialization of the learning algorithm to the data. Instead, it is better to measure the regressor's accuracy on a test set/validation set (refer to previous section) consisting of data not used to train the model.

Estimating the *error* in prediction using holdout and random subsampling, cross-validation and bootstrap methods (discussed in the previous section) are common techniques for assessing accuracy of the predictor. Several alternative metrics can be used to assess the accuracy of numeric prediction.

Chapter

3 STATISTICAL LEARNING

3.1 MACHINE LEARNING AND INFERENTIAL STATISTICAL ANALYSIS

Statistical analysis saw the development of two branches in the 18th century—*Bayesian* and *classical statistics*. In case of the approach that originated from the mathematical works of Thomas Bayes, analysis is based on the concept of *conditional probability*: the probability of an event taking place considering that another event has already taken place. The quantification of the investigator's present state of beliefs, knowledge and assumptions marks the beginning of the Bayesian analysis. These *subjective priors*, in combination with observed data, are quantified probabilistically using an appropriate objective function.

Regression and correlation were concepts that were developed in the later part of the 19th century, for generic data analysis. A system developed for inference testing in medical sciences, by RA Fisher in the 1920s was based on the concept of *standard deviation*. Bayesian model for inference testing could result in extremely different conclusions by different medical investigators as they used various sets of subjective priors. Fisher developed his system with the objective of providing medical investigators with a common set of tools based on data alone (no subjective priors). To make his system work even for big samples, Fisher had to assume a number of things, including *linear regression*, to define his '*Parametric Model*'.

Mathematical research dominated the 1980s on lines similar to Fisher's statistical inference through the development of nonlinear versions of parametric techniques. Bayesians kept on researching to promote their approach. The line of thinking called *machine learning* emerged from Artificial Intelligence community (1990s) in search of intelligent machines. Machine learning methods allowed the analysis of extremely nonlinear relationships in big datasets which have no known distribution.

Conventional statistical analysis adopts the *deductive* technique to look for relationships in datasets. It employs past knowledge (domain theory) along with training examples to create a model. Machine learning methods, on the other hand, adopt the *inductive* technique to discover feeble patterns of relationships in datasets. In the absence of any more information, it is assumed that the best hypothesis pertaining to unseen patterns is the one *induced* by the training data observed.

However, now, there has been a convergence of the two perspectives. Machine learning methods integrate a lot of statistical thinking. Statistical tests are used by most learning algorithms to construct models, and to prune 'overfitting' in these models. *Stastical learning theory* (PAC framework, VC dimension framework; Section 2.3) tries to provide answers to questions such as what are the conditions necessary for successful learning? What are the conditions under which a specific learning algorithm can be certain of successful learning? This theory provided new directions for complex problems with nonlinear relationships in datasets by 'mapping' data points to high-dimensional spaces with 'kernels'.

Data mining developed much recently, in the 1990s, and grew to become a significant field in the initial years of the 21st century. It is representative of a union of many well-established areas of interest—classical Bayesian statistics, classical parametric statistics, machine learning and database technology. Many off-the-shelf data mining system products and domain-specific data mining application software are on offer in the market, even though several data mining problems still require to be examined in detail.

We do not intend to describe in detail any specific commercial data mining system. Instead, we briefly outline popular data mining techniques for predictive modeling, that form part of most of the commercial data mining systems.

We begin this chapter by presenting a handful of key concepts from *descriptive statistics*. Descriptive statistics is only solely concerned with properties of the observed data. It has proven to be useful for data exploration (Section 7.2). It also provides useful tools for inferential statistical analysis/machine learning/data mining. Section 3.2 describes key concepts of descriptive statistics, used later in this book.

Inferential statistics is concerned with making predictions from data. Mathematical methods of inferential statistics employ probability theory for inferring the properties of an underlying distribution from the analysis of properties of a data sample drawn from it. It is concerned also with the testing of the inferences made for precision and reliability (hypotheses testing). A handful of widely used results from inferential statistical analysis are covered in this chapter.

The widely used technique based on classical Bayesian statistics is *naive Bayes classifier*. The Bayesian learning methods (statistical inference methods) have been found to be competitive with other machine learning algorithms in many cases, and in some cases they outperform. In this chapter, we introduce Bayesian learning, giving detailed coverage of *naive Bayes classifier*, and *k-Nearest Neighbor (k-NN) classifier*.

The inference techniques based on classical parametric statistics: *linear regression, logistic regression, discriminant analysis*; are also covered in this chapter. The other topic based on statistical techniques included in this chapter is *Minimum Description Length (MDL) principle*.

3.2 DESCRIPTIVE STATISTICS IN LEARNING TECHNIQUES

The standard 'hard-computing' paradigm is based on analytical closed-form models using a reasonable number of equations that can solve the given problem in a reasonable time, at reasonable cost, and with reasonable accuracy. It is a mathematically well-established discipline.

The field of machine learning is also mathematically well-founded; it uses 'soft-models'—modern computer-based applications of standard and novel mathematical and statistical techniques. Each of these techniques is a broad subject by itself with inputs from linear algebra and analytical geometry, vector calculus, unconstrained optimization, constrained optimization, probability theory and information theory. Our focus in this book is on 'applied' nature of machine learning; in-depth knowledge of these and related topics to understand the content of this book is not essential. However, in-depth knowledge will be essential for advanced study and research in machine learning.

We have assumed that the reader has some knowledge of these techniques. The summary of the properties of the techniques and the notation we use are described as and when a technique appears in our presentation.

Statistics developed as a discipline markedly different from mathematics over the past century and a half, to help scientists derive knowledge from observations, and come up with experiments that give rise to the reproducible and correct outcomes pertaining to the scientific technique. The methods established in the past on small amounts of data in a world of hand calculations, have managed to survive and continue to prove how useful they are. These methods have proved how worthwhile they are, not merely in the original spheres but also virtually in all areas wherein data is collected.

This section is aimed at presenting a few primary ideas from descriptive statistics that have confirmed their utility as tools for inferential statistical analysis/machine learning/data mining [46, 47].

3.2.1 Representing Uncertainties in Data: Probability Distributions

One of the common features of existing information for machine learning is the uncertainty associated with it. Real-world data tend to remain incomplete, noisy, and inconsistent. Noise, missing values, and inconsistencies add to the inaccuracy of data. Although *data cleansing* (Chapter 7) procedures try to approximately fill-in the missing values, smooth out noise while identifying outliers, and rectify inconsistencies in the data, inaccuracies do crop up and bring in the factor of uncertainty. Information, in other words, is not always suitable for solving problems. However, machine intelligence can tackle these defects and can usually make the right judgments and decisions. Intelligent systems should possess the capability to deal with uncertainties and derive conclusions.

Most popular uncertainty management paradigms are based on probability theory. Probability can be viewed as a numerical measure of the liklihood of occurrence of an outcome relative to the set of other alternatives. The set of all possible outcomes is the *sample space* and each of the individual outcomes is a *sample point*. Since the outcomes are uncertain, they are termed the *random variables*.

For problems of interest to us, the data matrix $\mathcal{D}$ is the sample space; the features/attributes x_j; $j = 1,\ldots, n$, are scalar random variables, and each of N n-dimesional vector random variable $\mathbf{x} \in \Re^n$ is a sample point. Initially, let us focus on features x_j in the data matrix defining random variables; we will add the output column y in our data space later and consider the total sample space.

Probability Mass Function

Assume $n = 1$, and the feature x is a random variable representing a sample point. Random variable x can be *discrete* or *continuous*. When x is discrete, it can possess finite number of discrete values v_{lx}; $l = 1, 2,\ldots, d$. The occurrence of discrete value v_{lx} of the random variable x is expressed by the probability $P(x = v_{lx})$.

From the frequency point of view, $P(x = v_{lx}) \triangleq P(v_{lx})$ can be interpreted as,

$$P(v_{lx}) = \lim_{N \to \infty} \frac{\Delta N}{N} \tag{3.1}$$

where

N = number of sample points, and
ΔN = number of times $x = v_{lx}$

The probabilities of all possible values of x are expressed as a *probability mass function:*

$$\begin{aligned} P(x) &= \langle P(v_{1x}), \ldots, P(v_{dx}) \rangle \\ P(v_{lx}) &\geq 0; l = 1, 2, \ldots, d \\ P(v_{1x}) + P(v_{2x}) + \cdots + P(v_{dx}) &= 1 \end{aligned} \tag{3.2}$$

Figure 3.1 graphically displays a probability mass function.

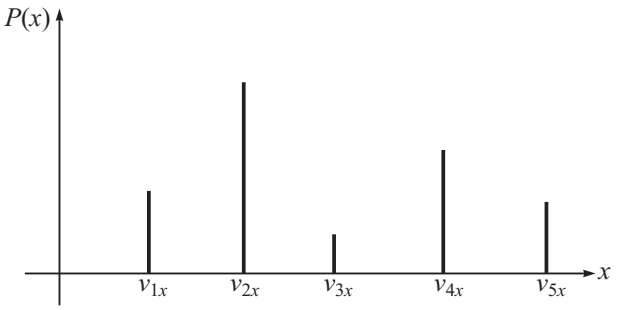


Figure 3.1 A probability mass function

Probability Density Function

A continuous random variable can take infinite values within its domain. In this case, the probability of a particular value within the domain is zero. Thus, we describe a continuous random variable not by the probability of taking on a certain value but by probability of being within a range of values.

For a continuous random variable x, probabilities are associated with the ranges of values of the variable, and consequently, at the specific value of x, only the density of probability is defined. If $p(x)$ is the *probability density function*[1] of x, the probability of x being in the interval (v_{1x}, v_{2x}) is:

$$\begin{aligned} P(v_{1x} \leq x < v_{2x}) &= \int_{v_{1x}}^{v_{2x}} p(x)\,dx \\ p(x) \geq 0, \text{ and } &\int_{-\infty}^{\infty} p(x)\,dx = 1 \end{aligned} \tag{3.3}$$

[1] We generally use an uppercase $P(\cdot)$ to denote a probability mass function, and a lowercase $p(\cdot)$ to denote a probability density function.

Figure 3.2 graphically displays a probability density function.

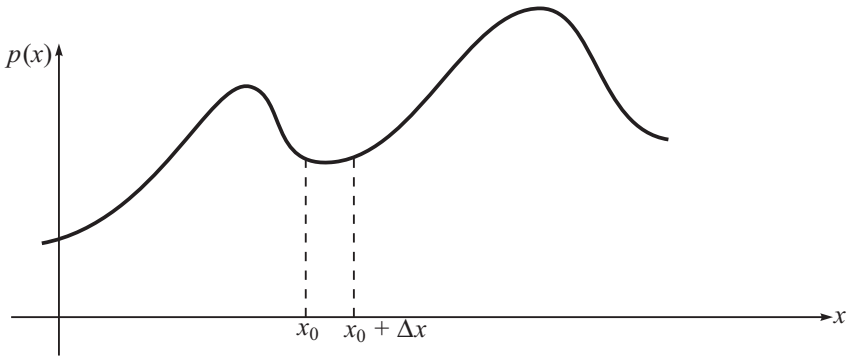


Figure 3.2 A probability density function

Probability that x has a value between x_0 and $x_0 + \Delta x$ is:

$$P(x_0 \le x < x_0 + \Delta x) = \int_{x_0}^{x_0 + \Delta x} p(x)\, dx \tag{3.4a}$$

If Δx is incrementally small,

$$P(x_0 \le x < x_0 + \Delta x) = p(x = x_0)\, \Delta x \tag{3.4b}$$

From the frequency point of view,

$$p(x) = \lim_{\Delta x \to 0} \left[\frac{P(x_0 \le x < x_0 + \Delta x)}{\Delta x} \right] = \lim_{\substack{N \to \infty \\ \Delta x \to 0}} \left[\frac{\Delta N / N}{\Delta x} \right] \tag{3.5}$$

where N = number of sample points, and

ΔN = number of times the random variable x lies in the range $x_0 \le x < x_0 + \Delta x$

Vector Random Variables

The results for a scalar random variable x can easily be extended to a vector random variable $\mathbf{x}$, with components x_j: $j = 1,\ldots, n$. The random variable $\mathbf{x}$ can take on values in discrete sample space $\Re_d^n : V_{\mathbf{x}} = \{v_{lx_1}, v_{lx_2}, \ldots, v_{lx_n}\}; l = 1, 2, \ldots, d_j$. For each possible value of $\mathbf{x}$ in $\Re_d^n$, we have a *joint probability*,

$$\begin{gathered} P(\mathbf{x}) = P\,(x_1 = v_{lx_1}, x_2 = v_{lx_2}, \ldots, x_n = v_{lx_n}) \\ P(\mathbf{x}) \ge 0, \text{ and } \sum_{\mathbf{x} \in \Re_d^n} P(\mathbf{x}) = 1 \end{gathered} \tag{3.6}$$

Note that $P(\mathbf{x})$ is a function of n variables and can be very complicated multidimensional function. However, if the random components x_j ; $j = 1, \ldots, n$, are *statistically independent*, then

$$P(\mathbf{x}) = P(x_1)\, P(x_2) \cdots P(x_n) = \prod_{j=1}^{n} P(x_j) \tag{3.7}$$

The *multivariate* situation is similarly handled with continuous random vectors $\mathbf{x}$. The *probability density function* $p(\mathbf{x})$ must satisfy

$$p(\mathbf{x}) \geq 0, \text{ and } \int_{-\infty}^{\infty} p(\mathbf{x})\, d\mathbf{x} = 1 \tag{3.8}$$

where the integral is understood to be an n-fold multiple integral and the element of n-dimensional volume $d\mathbf{x} = dx_1\, dx_2 \cdots dx_n$.

If the components of $\mathbf{x}$ are statistically independent, then the joint probability density function factors as,

$$p(\mathbf{x}) = \prod_{j=1}^{n} p(x_j) \tag{3.9}$$

Class-conditional Probability Density Function

We now consider total sample space (data matrix) wherein the continuous vector random variable $\mathbf{x}$ is an n-dimensional vector that corresponds to feature vector for each sample point (each row in data matrix) and y is the discrete scalar random variable that corresponds to the class to which $\mathbf{x}$ belongs.

$$y \in \{y_1, y_2, \ldots, y_M\} = \{y_q\,;\, q = 1, \ldots, M\}$$

The variable y has M discrete values: $\{y_1, y_2, \ldots, y_M\} = \{1, 2, \ldots, M\}$. We consider $\mathbf{x}$ to be a continuous random variable whose density function depends on the class y, expressed as $p(\mathbf{x}|y)$. This is the *class-conditional probability density function*—the probability density function for $\mathbf{x}$ given that the class is y. The difference between $p(\mathbf{x}|y_q)$ and $p(\mathbf{x}|y_k)$ describes the difference in the feature vector $\mathbf{x}$ between data of classes y_q and y_k ; $q, k \in 1, 2, \ldots, M$.

If the attribute values x_j ; $j = 1, \ldots, n$, are statistically independent, then

$$p(\mathbf{x}|y) = \prod_{j=1}^{n} p(x_j|y) \tag{3.10}$$

Figure 3.3 illustrates the class-conditional probability density function under the assumption of statistical independence.

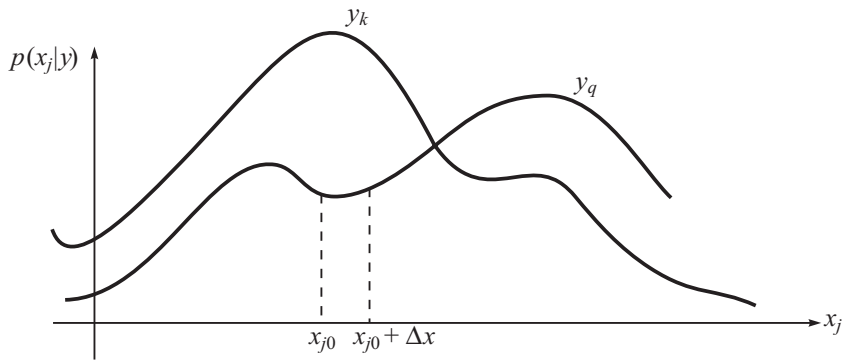


Figure 3.3 Class-conditional probability density function

Probability that x_j is between x_{j0} and $x_{j0} + \Delta x$,

$$P(x_{j0} \leq x_j < x_{j0} + \Delta x | y_q) = \int_{x_{j0}}^{x_{j0}+\Delta x} p(x_j | y_q)\, dx_j \tag{3.11a}$$

If Δx is incrementally small,

$$P(x_{j0} \leq x_j < x_{j0} + \Delta x | y_q) = p(x_j = x_{j0} | y_q)\, \Delta x \tag{3.11b}$$

Estimation of Probability Distributions

The probabilistic structure of a random variable can be characterized completely if its *distribution function* (probability mass function (pmf)/probability density function (pdf)) is known. Therefore pmf/pdf constitutes a mathematical model of the random variable.

Once mathematical models of random variables are specified, it is possible to design optimal classifiers. *Bayesian decision theory* [1, 4] gives a basic statistical model for designing optimal classifiers. This model is based on quantifying the trade-offs between various classification decisions and the risks associated with them. It assumes that the decision problem is posed in probabilistic terms, and that all the relevant pmfs/pdfs; specifically, probability mass function $P(y_q)$ and class-conditional densities $p(\mathbf{x}|y_q)$, $q = 1, \ldots, M$, are known.

Sadly, in pattern recognition applications, we hardly ever have this kind of total knowledge of the probabilistic structure of the problem. Typically, we only have some rough, general knowledge about the situation, along with several *design samples* or *training data*—specific representatives of the patterns we wish to categorize. The problem, therefore, is to discover some method of using this information for designing or training the classifier.

One model for this problem is to make use of the samples to make an estimation of unknown probabilities, and probability densities, and then employ the resulting estimates as if they were the true values. Typically, for supervised pattern classification, the probabilities $P(y_q)$ are estimated quite easily without any problems (refer to Eqn (3.1)). But then, estimation of class-conditional densities $p(\mathbf{x}|y_q)$ is not practical (refer to Eqn (3.5)). The available samples turn out to be small in number for feasibility of this estimation, and serious issues emerge as a result of the large dimensionality of the vector $\mathbf{x}$.

If our general knowledge pertaining to the problem allows parameterization of the conditional densities, then the difficulties associated with these issues can be brought down. Parametric forms of models of certain commonly observed density functions are known; once the form of underlying density function is chosen, we need to simply estimate its parameters.

Parameter estimation is a classical problem in statistics, which can be approached in many ways. Two widely used and feasible methods are *maximum likelihood estimation* and *Bayesian estimation*. Even though the results obtained from these two methods are often almost same, the models differ in terms of concept.

Maximum-likelihood technique looks at the parameters as quantities wherein values are fixed but not known. The ideal estimate of their value is one that maximizes the probability of obtaining the samples actually observed. Bayesian techniques consider the parameters to be random variables with certain known prior distribution. On observation, the samples convert this to a posterior density, which helps us tune our earlier estimate of the actual values of the parameters. Typically, the effect of observing additional samples is to improve *a posteriori* density function, sharpening it to peak near the true values of the parameters (refer to [4] for details on both the methods).

3.2.2 Descriptive Measures of Probability Distributions

Expected (mean or average) Value

One of the most important descriptive values of a random variable is the point around which distribution is centered, known as measure of central tendency. The *expected (mean* or *average) value* is the best measure for central tendency.

For a scalar random variable x, the expected value, denoted as $\mathbb{E}[x]$ (or μ), is

$$\mathbb{E}[x] = \mu = \int_{-\infty}^{\infty} xp(x)\,dx \tag{3.12a}$$

where $p(x)$ is the probability density function.

For a discrete random variable described by a large random sample S,

$$\mathbb{E}[x] = \mu = \sum_{x \in S} xP(x) \tag{3.12b}$$

where $P(x)$ is probability mass function.

The following results immediately follow:

1. $\mathbb{E}[C] = C$; C = constant (3.13a)
2. $\mathbb{E}[Cx] = C\,\mathbb{E}[x]$ (3.13b)
3. $\mathbb{E}\left[\sum_{j=1}^{n} x_j\right] = \sum_{j=1}^{n} \mathbb{E}[x_j]$ (3.13c)
4. If there are two random variables x_1 and x_2 defined on a sample space, then

$$\mathbb{E}[x_i x_2] = \int_{-\infty}^{\infty} \int_{-\infty}^{\infty} x_1 x_2 p(x_1, x_2) dx_1 dx_2 \tag{3.13d}$$

where $p(x_1, x_2)$ is *joint probability density function*.

5. If $f(x)$ is any function of x, the expected value of f is defined as,

$$\mathbb{E}[f(x)] = \int_{-\infty}^{\infty} f(x) p(x) dx \tag{3.13e}$$

6. For a vector random variable $\mathbf{x}$,

$$\mathbb{E}[\mathbf{x}] = \boldsymbol{\mu} = \int_{-\infty}^{\infty} \mathbf{x} p(\mathbf{x}) d\mathbf{x} \tag{3.13f}$$

In discrete case,

$$\mathbb{E}[\mathbf{x}] = \boldsymbol{\mu} = \sum_{\mathbf{x} \in S} \mathbf{x} P(\mathbf{x}) \tag{3.13g}$$

Variance

The expected value or mean is used more than any other single number to describe a random variable. Usually it is also desired to know the 'spread' of the random variable about the mean. The most convenient measure used for this purpose is variance, denoted as Var[x] or σ^2, defined below:

$$\text{Var}[x] = \sigma^2 = \mathbb{E}[(x - \mu)^2] = \int_{-\infty}^{\infty} (x - \mu)^2 p(x) dx \tag{3.14a}$$

It may be noted that variance is simply the weighted average of squared deviations from the mean value.

In discrete case,

$$\text{Var}[x] = \sigma^2 = \mathbb{E}[(x - \mu)^2] = \sum_{x \in S} (x - \mu)^2 P(x) \tag{3.14b}$$

A more convenient measure of dispersion is the square root of variance, called the *standard deviation*, σ, i.e.,

$$\sigma = \sqrt{\text{Var}[x]} \tag{3.14c}$$

The following results immediately follow:

1. $$\begin{aligned} \text{Var}[x] &= \mathbb{E}[(x - \mathbb{E}[x])^2] \\ &= \mathbb{E}[x^2 - 2x\, \mathbb{E}[x] + (\mathbb{E}[x])^2 \\ &= \mathbb{E}[x^2] - (\mathbb{E}[x])^2 \end{aligned} \tag{3.14d}$$

   $\mathbb{E}[x^2]$ is called the *second moment* ($\mathbb{E}[x]$ is the *first moment*) of the random variable.
2. The moments of difference between a random variable and its mean value are called *central moments*. Variance is thus a *second central moment*.
3. For two random variables x_1 and x_2 defined on a sample space, $\mathbb{E}[x_1 x_2]$ is the *joint second moment* of x_1 and x_2.

Covariance

The joint second moment of x_1 and x_2 about their respective means μ_1 and μ_2, is the *covariance* of x_1 and x_2, i.e.,

$$\text{Cov}[x_1, x_2] = \sigma_{12} = \mathbb{E}[(x_1 - \mu_1)(x_2 - \mu_2)] \tag{3.15a}$$

$$= \mathbb{E}[x_1 x_2] - \mathbb{E}[x_1]\,\mathbb{E}[x_2] \tag{3.15b}$$

The covariance is an important measure of the degree of statistical dependence between x_1 and x_2. If x_1 and x_2 are statistically independent, $\mathbb{E}[x_1 x_2] = \mathbb{E}[x_1]\,\mathbb{E}[x_2]$, and $\text{Cov}[x_1, x_2] = \sigma_{12} = 0$. If $\text{Cov}[x_1, x_2] = 0$, then x_1 and x_2 are said to be *uncorrelated*.

Sometimes *correlation coefficient*, denoted as ρ, is more convenient to use. It is defined as,

$$\rho = \frac{\text{Cov}[x_1, x_2]}{\text{Var}[x_1]\,\text{Var}[x_2]} = \frac{\sigma_{12}}{\sigma_1^2\,\sigma_2^2} \tag{3.16}$$

ρ is *normalized covariance* and must always be between –1 and +1. If $\rho = +1$, then x_1 and x_2 are maximally positively correlated (the values of x_1 and x_2 tend to be both large or small relative to their respective means); while if $\rho = -1$, they are maximally negatively correlated (the values of x_1 tend to be large when values of x_2 are small and vice versa). If $\rho = 0$, the variables are uncorrelated.

The following results immediately follow:

1. For an n-dimensional random vector $\mathbf{x}$, covariance matrix $\mathbf{\Sigma}$ describes the correlations among its n components $x_1, x_2, \ldots, x_n$; $\boldsymbol{\mu} = [\mu_1\ \mu_2 \cdots\ \mu_n]^T$ represents the mean vector.

$$\mathbf{\Sigma} = \mathbb{E}[(\mathbf{x} - \boldsymbol{\mu})(\mathbf{x} - \boldsymbol{\mu})^T] \tag{3.17a}$$

$$= \int_{-\infty}^{\infty} (\mathbf{x} - \boldsymbol{\mu})(\mathbf{x} - \boldsymbol{\mu})^T\, p(\mathbf{x})\, d\mathbf{x} \tag{3.17b}$$

2. The covariance matrix $\mathbf{\Sigma}$ is defined as the (square) matrix whose jk^{th} element σ_{jk} is the covariance of x_j and x_k :

$$\sigma_{jk} = \sigma_{kj} = \mathbb{E}[(x_j - \mu_j)(x_k - \mu_k)];\ j, k = 1, \ldots, n$$

$$\mathbf{\Sigma} = \begin{bmatrix} \mathbb{E}[(x_1 - \mu_1)(x_1 - \mu_1)] & \mathbb{E}[(x_1 - \mu_1)(x_2 - \mu_2)] & \cdots & \mathbb{E}[(x_1 - \mu_1)(x_n - \mu_n)] \\ \mathbb{E}[(x_2 - \mu_2)(x_1 - \mu_1)] & \mathbb{E}[(x_2 - \mu_2)(x_2 - \mu_2)] & \cdots & \mathbb{E}[(x_2 - \mu_2)(x_n - \mu_n)] \\ \vdots & \vdots & & \vdots \\ \mathbb{E}[(x_n - \mu_n)(x_1 - \mu_1)] & \mathbb{E}[(x_n - \mu_n)(x_2 - \mu_2)] & \cdots & \mathbb{E}[(x_n - \mu_n)(x_n - \mu_n)] \end{bmatrix}$$

$$= \begin{bmatrix} \sigma_{11} & \sigma_{12} & \cdots & \sigma_{1n} \\ \sigma_{21} & \sigma_{22} & \cdots & \sigma_{2n} \\ \vdots & \vdots & & \vdots \\ \sigma_{n1} & \sigma_{n2} & \cdots & \sigma_{nn} \end{bmatrix} = \begin{bmatrix} \sigma_1^2 & \sigma_{12} & \cdots & \sigma_{1n} \\ \sigma_{21} & \sigma_2^2 & \cdots & \sigma_{2n} \\ \vdots & \vdots & & \vdots \\ \sigma_{n1} & \sigma_{n2} & \cdots & \sigma_n^2 \end{bmatrix} \tag{3.18}$$

3. $\boldsymbol{\Sigma}$ is symmetric and its diagonal elements are simply the variances of the individual elements of $\mathbf{x}$, which can never be negative; the off-diagonal elements are the covariances, which can be positive or negative. If the variables are statistically independent, the covariances are zero, and the covariance matrix is diagonal.

3.2.3 Descriptive Measures from Data Sample

A *statistic* is a measure of a sample of data. Statistics is the study of these measures and the samples they are measured on. Let us summarize the simplest statistical measures employed in data exploration.

1. **Range:** It is the difference between the smallest and the largest observation in the sample. It is frequently examined along with the minimum and maximum values themselves.
2. **Mean:** It is the arithmetic average value, that is, the sum of all the values divided by the number of values.
3. **Median:** The median value divides the observations into two groups of equal size—one possessing values smaller than the median and another one possessing values bigger than the median
4. **Mode:** The value that occurs most often, is called the *mode* of data sample.
5. **Variance and Standard Deviation:** The difference between a given observation and the arithmatic average of the sample is called its *deviation*. The variance is defined as the arithmatic average of the squares of the deviations. It is a measure of dispersion of data values; measures how closely the values cluster around their arithmatic average value. A low variance means that the values stay near the arithmatic average; a high variance means the opposite.

   Standard deviation is the square root of the variance and is commonly employed in measuring dispersion. It is expressed in units similar to the values themselves while variance is expressed in terms of those units squared.
6. **Covariance matrix:** Arithmatic average of the matrix $(\mathbf{x}^{(i)} - \boldsymbol{\mu})\,(\mathbf{x}^{(i)} - \boldsymbol{\mu})^T$ gives us *sample covariance matrix*; $i = 1,\ldots, N$ are the N observations in the data sample, and $\boldsymbol{\mu}$ is the arithmatic average of the sample. $\mathbf{x}$ and $\boldsymbol{\mu}$ are both n-dimensional vectors.

$$\boldsymbol{\mu} = \frac{1}{N}\sum_{i=1}^{N} \mathbf{x}^{(i)} \tag{3.19}$$

$$\boldsymbol{\Sigma} = \frac{1}{N}\sum_{i=1}^{N} (\mathbf{x}^{(i)} - \boldsymbol{\mu})\,(\mathbf{x}^{(i)} - \boldsymbol{\mu})^T \tag{3.20}$$

Correlation coefficient ρ between two attributes can be calculated from data samples using Eqn (3.16).

3.2.4 Normal Distributions

Of the different density functions explored, the maximum attention has been received by the *multivariate Gaussian (normal) density*. This is primarily because of its analytical tractability. It has a simple parameterized form, and is entirely controlled by numerical values of the two parameters, the *mean* and the *variance/covariance*. Multivariate normal density is, for many applications, an appropriate model for the given data.

According to the *central limit theorem* in statistics:

As random samples are increasingly taken from a population, the distribution of the averages (or any similar statistic) of the sample follows the normal distribution. With an increase in the number of samples, the average of the samples comes closer to the average of the entire population.

The contrapositive is true as well, that is, if the distribution of the values does *not* follow a normal distribution, then it is highly unlikely that the original values were drawn randomly. And, if they were not drawn randomly, some other process is at work.

The *normal* or *Gaussian* probability density function is significant for not only theoretical but also practical reasons. In one dimension, it is defined by (univariate normal probability density function):

$$p(x) = \frac{1}{\sigma\sqrt{2\pi}} \exp\left[-\tfrac{1}{2}\left(\frac{x-\mu}{\sigma}\right)^2\right] \tag{3.21}$$

The conventional description of normal density is a bell-shaped curve, totally controlled by the numerical values of the two parameters, the mean μ and the variance σ^2. The distribution is symmetrical about the mean; the peak occurring at $x = \mu$, and the width of the bell is proportional to the standard deviation σ (Fig. 3.4).

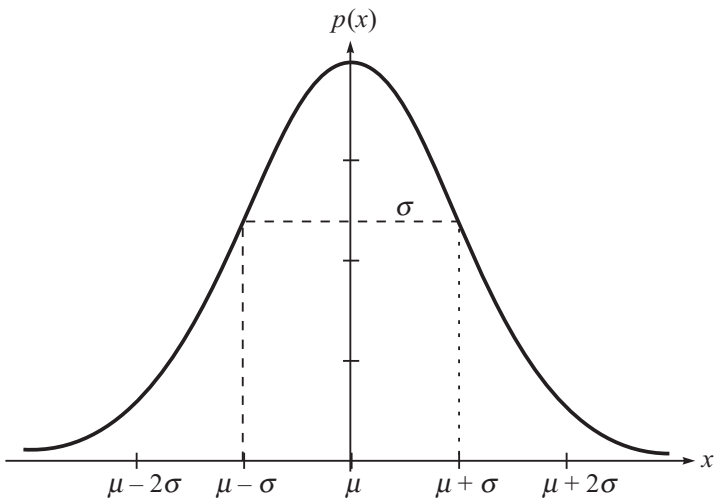


Figure 3.4 A univariate normal distribution

If the random variable is the standardized random variable, $u = (x - \mu)/\sigma$, then the normal distribution is said to be *standardized* having zero mean and unit standard deviation—that is,

$$p(u) = \frac{1}{\sqrt{2\pi}} e^{-u^2/2} \tag{3.22}$$

Consider now a vector random variable **x** with each of the n scalar random variable x_j normally distributed with mean μ_j and variance σ_j^2. If these variables are independent, their joint density has the form

$$\begin{aligned} p(\mathbf{x}) &= \prod_{j=1}^{n} p(x_j) \\ &= \frac{1}{(2\pi)^{n/2} \prod_{j=1}^{n} \sigma_j} \exp\left[-\tfrac{1}{2} \sum_{j=1}^{n} \left(\frac{x_j - \mu_j}{\sigma_j} \right)^2 \right] \end{aligned} \tag{3.23}$$

In this case, covariance matrix $\boldsymbol{\Sigma}$ is diagonal.

$$\boldsymbol{\Sigma} = \begin{bmatrix} \sigma_1^2 & 0 & \dots & 0 \\ 0 & \sigma_2^2 & \dots & 0 \\ \vdots & \vdots & & \vdots \\ 0 & 0 & \dots & \sigma_n^2 \end{bmatrix} \tag{3.24a}$$

$$\boldsymbol{\Sigma}^{-1} = \begin{bmatrix} 1/\sigma_1^2 & 0 & \dots & 0 \\ 0 & 1/\sigma_2^2 & \dots & 0 \\ \vdots & \vdots & & \vdots \\ 0 & 0 & \dots & 1/\sigma_n^2 \end{bmatrix} \tag{3.24b}$$

$$|\boldsymbol{\Sigma}|^{1/2} = \prod_{j=1}^{n} \sigma_j$$

We can write joint density compactly in terms of quadratic form:

$$p(\mathbf{x}) = \frac{1}{(2\pi)^{n/2} |\boldsymbol{\Sigma}|^{1/2}} \exp\left[-\tfrac{1}{2} (\mathbf{x} - \boldsymbol{\mu})^T \boldsymbol{\Sigma}^{-1} (\mathbf{x} - \boldsymbol{\mu}) \right] \tag{3.24c}$$

This, in fact, is the general form of a *multivariate normal density function*, where the covariance matrix $\boldsymbol{\Sigma}$ is no longer required to be diagonal (normally distributed random variables x_j may be dependent).

3.2.5 Data Similarity

Machine learning, like animal learning, relies heavily on the notion of *similarity*, in its search for valuable knowledge in database. Machine learning algorithms that interface with numerical form

of data representation: patterns are specified by a fixed number (n) of features, where each feature has a numerical value (real) for the pattern, visualize each data as a point in n-dimensional state space. Characterizing the similarity of the patterns in state space can be done through some form of *metric* (distance) measure: distance between two vectors is a measure of similarity between two corresponding patterns. Many measures of 'distance' have been proposed in the literature.

The ease with which humans classify and describe patterns, often leads to incorrect assumption that the capability is easy to automate. The choice of *similarity measure* is a deep question that lies at the core of machine learning.

Let us use d_{il} to depict a *distance metric* or *dissimilarity measure*, between patterns i and l. For pattern i, we have the vector of n measurements $(x_1^{(i)}, x_2^{(i)}, \ldots, x_n^{(i)})$, while for pattern l, we have the vector of measurements $(x_1^{(l)}, x_2^{(l)}, \ldots, x_n^{(l)})$.

Distances can be defined in multiple ways, but in general the following properties are required:

$$\begin{aligned} &\textit{Nonnegativity: } d_{il} \geq 0 \\ &\textit{Self-proximity: } d_{ii} = 0 \\ &\textit{Symmetry: } d_{il} = d_{li} \\ &\textit{Triangle Inequality: } d_{il} \leq d_{ik} + d_{kl} \end{aligned} \tag{3.25}$$

Euclidean distance: The most popular distance measure is the *Euclidean distance*, d_{il}, which between two patterns i and l, is defined by,

$$d_{il} = \sqrt{(x_1^{(i)} - x_1^{(l)})^2 + (x_2^{(i)} - x_2^{(l)})^2 + \cdots + (x_n^{(i)} - x_n^{(l)})^2} \tag{3.26}$$

Despite Euclidean distance being the most commonly used similarity measure, there are three primary characteristics that need to be taken into consideration.

(i) It depends highly on scale, and variables that possess bigger scales impact the total distance to a great extent. Thus, we first *normalize* continuous measurements and only then calculate the Euclidean distance. This transforms all measurements to the same scale.

(ii) Euclidean distance entirely ignores the relationship between measurements. If there is a strong correlation between measurements, a *statistical distance* measure seems to be a better candidate.

(iii) Euclidean distance is sensitive to outliers, and if the data comprises outliers, a preferred choice is the use of robust distances like *Manhattan distance*.

Statistical distance: This metric takes into consideration the correlation between measurements. With this metric, measurements extremely correlated with other measurements do not contribute as much as the uncorrelated or less correlated. The *statistical distance*, also referred to as the *Mahalanobis distance*, between patterns i and l is defined as,

$$d_{il} = \sqrt{(\mathbf{x}^{(i)} - \mathbf{x}^{(l)})^T \, \boldsymbol{\Sigma}^{-1} \, (\mathbf{x}^{(i)} - \mathbf{x}^{(l)})} \tag{3.27}$$

where $\mathbf{x}^{(i)}$ and $\mathbf{x}^{(l)}$ are n-dimensional vectors of measurement values of patterns i and l, respectively, and $\boldsymbol{\Sigma}$ is the covariance matrix of these vectors.

Manhattan distance: This distance looks at the absolute differences rather than squared differences, and is defined by,

$$d_{il} = \sum_{j=1}^{n} |x_j^{(i)} - x_j^{(l)}| \tag{3.28}$$

Minkowski metric: One general class of metrics for n-dimensional patterns is the Minkowski metric (also referred to as the L_p norm):

$$L_p(\mathbf{x}^{(i)}, \mathbf{x}^{(l)}) = \left(\sum_{j=1}^{n} |x_j^{(i)} - x_j^{(l)}|^p \right)^{1/p} \tag{3.29}$$

where $p \geq 1$ is a selectable parameter. Setting $p = 2$ gives the familiar Euclidean distance (L_2 norm) and setting $p = 1$ gives the Manhattan distance (L_1 norm). With respect to Eqns (3.26)–(3.29),

$$L_p(\mathbf{x}^{(i)}, \mathbf{x}^{(l)}) = \|\mathbf{x}^{(i)} - \mathbf{x}^{(l)}\|_p = \left(\sum_{j=1}^{n} |x_j^{(i)} - x_j^{(l)}|^p \right)^{1/p} \tag{3.30a}$$

(Minkowski norm)

$$\|\mathbf{x}^{(i)} - \mathbf{x}^{(l)}\|_2 = [(x_1^{(i)} - x_1^{(l)})^2 + (x_2^{(i)} - x_2^{(l)})^2 + \cdots + (x_n^{(i)} - x_n^{(l)})^2]^{1/2} \tag{3.30b}$$

(Euclidean norm)

$$\|\mathbf{x}^{(i)} - \mathbf{x}^{(l)}\|_1 = \sum_{j=1}^{n} |x_j^{(i)} - x_j^{(l)}| \tag{3.30c}$$

(Manhattan norm)

In this section, we have presented a handful of key ideas from descriptive statistics that we will be using in the book. Some other concepts/measures from statistics that have proven to be useful in machine learning are: Null Hypothesis; *P*-values; *Z*-scores, Confidence Interval; Chi-Square Distribution; *t*-Distribution; Chi-Square Test; *t*-Test; Analysis of Variance (ANOVA), and other measures [6, 19].

3.3 BAYESIAN REASONING: A PROBABILISTIC APPROACH TO INFERENCE

In the earlier chapters, while defining a prediction problem, the assumption made was that the objective is to maximize the success rate of predictions. In a classification case, for instance, the result of each instance is *correct* (in case the prediction agrees with the actual value for that instance), or *incorrect* (in case it does not agree). Everything is either black or white; there is no question of grey.

The Bayesian model of statistical decision making has a softer edge. It associates a probability each with prediction. The Bayesian classification aims to estimate the probabilities that a pattern to be classified belongs to various possible categories. The Bayesian method assumes that the