Introduction 1

This first chapter briefly reviews how the field of machine learning has evolved into a major discipline in computer science and engineering in the past decades. Afterward, it takes a descriptive approach and provides some simple examples to introduce basic concepts and general principles in machine learning to give readers a big picture of machine learning, as well as some general expectations on the topics that will be covered in this book. Finally, this introductory chapter concludes with a list of advanced topics in machine learning, which are currently pursued as active research topics in the machine learning community.

1.1 What Is Machine Learning?

Since its inception several decades ago, the digital computer has constantly amazed us with its unprecedented capability for computation and data storage. On the other hand, people are also extremely interested in investigating the limits on what a computer is able to do beyond the basic skills of computing and storing. The most interesting question along this line is whether the human-made machinery of digital computers can perform complex tasks that normally require human intelligence. For example, can computers be taught to play complex board games like chess and Go, transcribe and understand human speech, translate text documents from one language to another, and autonomously operate cars? These research pursuits have been normally categorized as a broad discipline in computer science and engineering under the umbrella of *artificial intelligence* (AI). However, artificial intelligence is a loosely defined term and is used colloquially to describe computers that mimic cognitive functions associated with the human mind, such as learning, perception, reasoning, and problem solving [207]. Traditionally, we tended to follow the same idea of computer programming to tackle an AI task because it was believed that we could write a large program to teach a computer to accomplish any complex task. Roughly speaking, such a program is essentially composed of a large number of "if-then" statements that are used to instruct the computer to take certain actions under certain conditions. These if-then statements are often called *rules*. All rules in an AI system are collectively called a *knowledge base* because they are often handcrafted based on the knowledge of human experts. Furthermore, some mathematical tools, such as logic and graphs, can also be adopted into some AI systems as

The term *artificial intelligence* (AI) was coined at a workshop at Dartmouth College in 1956 by John McCarthy, who was an MIT computer scientist and a founder of the AI field.

more advanced methods for knowledge representation. Once the knowledge base is established, some well-known search strategies can be used to explore all available rules in the knowledge base to make decisions for each observation. These methods are often called *symbolic* approaches [207]. Symbolic approaches were dominant in the early stage of AI because mathematically sound inference algorithms can be used to derive some highly explainable results through a transparent decision process, such as the *expert systems* popular in the 1970s and 1980s [110].

The key to the success of these knowledge-based (or rule-based) symbolic approaches lies in how to construct all necessary rules in the knowledge base. Unfortunately, this has turned out to be an insurmountable obstacle for any realistic task. First of all, the process of explicitly articulating human knowledge using some well-formulated rules is not straightforward. For example, when you see a picture of a cat, you can immediately recognize a cat, but it is difficult to express what rules you might have used to make your judgment. Second, the real world is often so complicated that it requires using an endless number of rules to cover all the different conditions in any realistic scenario. Constructing these rules manually is a tedious and daunting task. Third, even worse, as the number of rules increases in the knowledge base, it becomes impossible to maintain them. For example, some rules may contradict each other under some conditions, and we often have no good ways to detect these contradictions in a large knowledge base. Moreover, whenever we need to make an adjustment to a particular rule, this change may affect many other rules, which are not easy to identify as well. Fourth, rule-based symbolic systems do not know how to make decisions based on partial information and often fail to handle uncertainty in the decision-making process. As we know, neither partial information nor uncertainty is a major hurdle in human intelligence.

On the other hand, an alternative approach toward AI is to design learning algorithms by which computers can automatically improve their capability on any particular AI task through experience [165]. The past experience is fed to a learning algorithm as the so-called "training data" for the algorithm to learn from. The design of these learning algorithms has been motivated by different strategies, from biologically inspired learning machines [200, 206, 205] to probability-based statistical learning methods [56, 9, 112, 38]. Since the 1980s, the study of these automatic learning algorithms has quickly emerged as a prominent subfield in AI, under the name *machine learning*. The nature of automatic learning prevents machine learning from suffering the aforementioned drawbacks of the symbolic approaches. As opposed to the knowledge-based symbolic approaches, data-driven machine learning algorithms focus more on how to automatically exploit the training data to build some mathematical models in order to make decisions without having explicit programming to do so [212]. With the help of machine learning algorithms, the major burden in

The term *machine learning* was first coined in a 1959 paper [212] by Arthur Samuel, who was an IBM researcher and pioneer in the field of AI.

building an AI system has moved from the extremely challenging task of manual knowledge representation to a relatively feasible procedure of data collection. After initial success in some real-world AI applications during the 1970s and 1980s (e.g., speech recognition [9, 112] and machine translation [38]), a major paradigm shift occurred in the field of artificial intelligence—namely, the data-driven machine learning methods have replaced the traditional rule-based symbolic approaches to become the mainstream methodology for AI. As the computation power of modern computers constantly improves, machine learning has found a plethora of relevant applications in almost all engineering domains and has made a huge impact on our society.

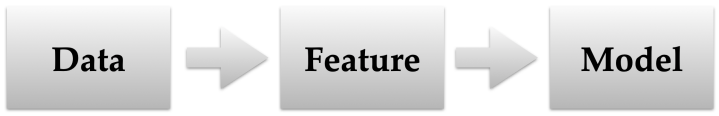

Figure 1.1: An illustration of the pipeline of building a machine learning system, consisting of three major steps of data collection, feature generation, and model training.

As shown in Figure 1.1, the pipeline of building a successful machine learning system normally consists of three key steps. In the first stage, we need to collect a sufficient amount of training data to represent the previous experience from which computers can learn. Ideally, the training data should be collected under the same conditions in which the system will be eventually deployed. The data collected in this way are often called *in-domain* data. Many learning algorithms also require human annotators to manually label the data in such a way to facilitate the learning algorithms. As a result, it is a fairly costly process to collect in-domain training data in practice. However, the final performance of a machine learning system in any practical task is largely determined by the amount of available in-domain training data. In most cases, accessing more in-domain data is the most effective way to boost performance for any real-world application. In the second stage, we usually need to apply some domain-specific procedures to extract the so-called features out of the raw data. The features should be compact but also retain the most important information in the raw data. The feature-extraction procedures need to be manually designed based on the nature of the data and the domain knowledge, and they often vary from one domain to another. For example, a good feature to represent speech signals should be derived based on our understanding of speech itself, and it should drastically differ from a good feature to represent an image. In the final stage, we choose a learning algorithm to build some mathematical models from the extracted feature representations of the training data. The machine learning research in the past few decades has provided us with a wide range of choices in terms of which learning algorithms to use and which models to build. The main purpose of this book is to introduce different choices of machine learning methods in a systematic way. Most of these learning methods are generic enough for a variety of

A recent trend in machine learning is to replace the handcrafted features with some automatic feature extraction algorithms. The recent *end-to-end learning* tends to combine the last two steps of feature extraction and modeling into a single uniform module that can be jointly learned from the training data. We will discuss the end-to-end learning in Section 8.5.

https://doi.org/10.1017/9781108938051.003 Published online by Cambridge University Press

problems and applications, and they are usually independent of domain knowledge. Therefore, most learning methods and their corresponding models can be introduced in a general manner without restricting their use to any particular application.

1.2 Basic Concepts in Machine Learning

In this section, we will use some simple examples to explain some common terminology, as well as several basic concepts widely used in machine learning.

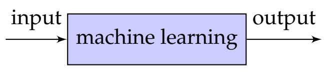

Figure 1.2: A system view of any machine learning problem.

Generally speaking, it is useful to take the system view of input and output to examine any machine learning problem, as shown in Figure 1.2. For any machine learning problem at hand, it is important to understand what its input and output are, respectively. For example, in a speech-recognition problem, the system's input is speech signals captured by a microphone, and the output is the words/sentences embedded in the signals. In an English-to-French machine translation problem, the input is a text document in English, and the output is the corresponding French translation. In a self-driving problem, the input is the videos and signals of the surrounding scenes of the car, captured by cameras and various sensors, and the output is the control signals generated to guide the steering wheel and brakes.

The system view in Figure 1.2 can also help us explain several popular machine learning terminologies.

1.2.1 Classification versus Regression

Depending on the type of the system outputs, machine learning problems can be broken down into two major categories. If the output is continuous—namely, it can take any real value within an interval—it is a *regression* problem. On the other hand, if the output is discrete—namely, it can only take a value out of a finite number of predefined choices—it is said to be a *classification* problem. For instance, speech recognition is a classification problem because the output must be constructed using a finite number of words allowed in the language. On the other hand, image generation is a regression problem because the pixels of an output image can take any arbitrary values. It is fundamentally similar in principle to solve classification and regression problems, but they often need slightly different treatments in problem formulation.

In some machine learning problems, the outputs are structured objects. These problems are referred to as *structured learning* (a.k.a. *structured prediction*) [10]. Some examples are when the output is a binary tree or a sentence following certain grammar rules.

1.2.2 Supervised versus Unsupervised Learning

As we know, all machine learning methods require collecting training data in the first place. *Supervised learning* deals with those problems where both the input and output shown in Figure 1.2 can be accessed in data collection. In other words, the training data in supervised learning consist of input–output pairs. For each input in the training data, we know its corresponding output, which can be used to guide learning algorithms as a supervision signal. Supervised learning methods are well studied in machine learning and usually guarantee good performance, as long as sufficient numbers of input–output pairs are available. However, collecting the input–output pairs for supervised learning often requires human annotation, which may be expensive in practice.

In contrast, *unsupervised learning* methods deal with the problems where we can only access the input shown in Figure 1.2 when collecting the training data. A good unsupervised learning algorithm should be able to figure out some criteria to group similar inputs together using only the information of all possible inputs, where two inputs are said to be similar only when they are expected to yield the same output label. The fundamental difficulty in unsupervised learning lies in how to know which inputs are similar when their output labels are unavailable. Unsupervised learning is a much harder problem because of the lack of supervision information. In unsupervised learning, it is usually cheaper to collect training data because it does not require extra human efforts to label each input with the corresponding output. However, unsupervised learning largely remains an open problem in machine learning. We desperately need good unsupervised learning strategies that can effectively learn from unlabeled data.

In many circumstances, unsupervised learning is also called *clustering* [66].

In between these two extremes, we can combine a small amount of labeled data with a large amount of unlabeled data during training. These learning methods are often called *semisupervised learning*. In other cases, if the true outputs shown in Figure 1.2 are too difficult or expensive to obtain, we can use other readily available information, which is only partially relevant to the true outputs, as some weak supervision signals in learning. These methods are called *weakly supervised learning*.

We know that it is difficult and costly to annotate the precise meaning of each word in text documents. However, due to the distribution hypothesis [91] in linguistics (i.e., "words that are close in meaning will occur in similar pieces of text"), the surrounding words can be used as weak supervision signals to learn the meanings of words. See Example 7.3.2.

1.2.3 Simple versus Complex Models

In machine learning, we run learning algorithms over training data to build some mathematical models for decision making. In terms of choosing the specific model to be used in learning, we usually have to make a sensible choice between simple models and complex models. The complexity of a model depends on the functional form of the model as well as the number of free parameters. In general, linear models are treated as simple models, whereas nonlinear models are viewed as complex models because nonlinear models can capture much more complicated patterns in data distributions than linear ones. A simple model requires much less computing resources and can be reliably learned from a much smaller training set. In many cases, we can derive a full theoretical analysis for simple models, which gives us a better understanding of the underlying learning process. However, the performance of simple models often saturates quickly as more training data become available. In many practical cases, simple models can only yield mediocre performance because they fail to handle complicated patterns, which are the norm in almost all real-world applications. On the other hand, complex models require much more computing resources in learning, and we need to prepare much more training data to reliably learn them. Due to their complex functional forms, there does not exist any theoretical analysis for many complex models. Hence, learning complex models is often a very awkward black-box process and usually requires many inexplicable tricks to yield optimal results.

We will introduce linear models in Chapter 6 and more complex models in Chapter 8.

Example 1.2.1 *Curve Fitting*

There exists an unknown function $y = f(x)$. Assume we can only observe its function values at several isolated points, indicated by blue circles in Figure 1.3. Show how to determine its values for all other points in the interval.

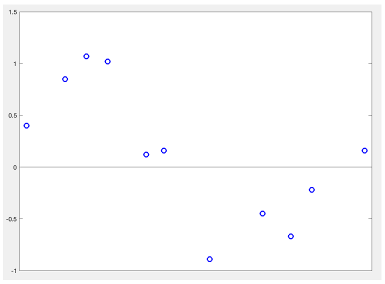

Figure 1.3: An illustration of a curve-fitting problem, which can be viewed as a regression problem in machine learning.

This is a standard curve-fitting problem in mathematics, which requires constructing a curve, or mathematical function, to best fit these observed points. From the perspective of machine learning, this curve-fitting problem is a regression problem because it requires us to estimate the function value y, which is continuous, for any x in the interval. The observed points serve as the training data for this regression problem. Because we can access both input x and output y in the training data, it is a supervised learning problem.

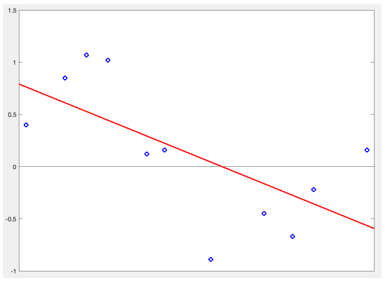

Figure 1.4: An illustration of using a linear model for the curve-fitting problem shown in Figure 1.3.

First of all, assume we construct a linear function for this problem:

$$f(x) = a_0 + a_1\, x.$$

Through a learning process that determines the two unknown coefficients (to be introduced in the later chapters), we can construct the best-fit linear function in Figure 1.4. We can see that this best-fit linear function yields values quite different from most of the observed points and has failed to capture the "up-and-down wiggly pattern" shown in the training data. This indicates that linear models may be too simple for this task. In fact, this problem can be easily solved by choosing a more complex model. A natural choice here is to use a higher-order polynomial function. We can choose a fourth-order polynomial function, as follows:

$$f(x) = a_0 + a_1\,x + a_2\,x^2 + a_3\,x^3 + a_4\,x^4.$$

After we determine all five unknown coefficients, we can find the best-fit fourth-order polynomial function, as shown in Figure 1.5. From that, we can see that this model captures the pattern in the data much better despite still yielding slightly different values at the observed points. ◆

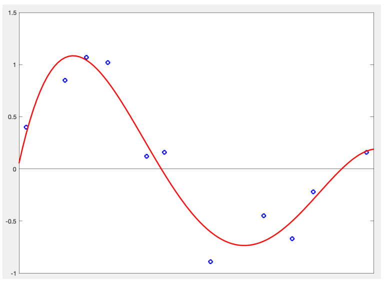

Figure 1.5: An illustration of using a fourth-order polynomial function for the curve-fitting problem.

Example 1.2.2 *Fruits Recognition*

Assume we want to teach a computer to recognize different fruits based on some observed characteristics, such as *size*, *color*, *shape*, and *taste*. Consider a suitable model that can be used for this purpose.

This is a typical classification problem because the output is discrete: it must be a known fruit (e.g., *apple*, *grape*). Among many choices, we can implement the tree-structured model shown in Figure 1.6 for this classification problem. In this model, each internal node is associated with a binary question regarding one aspect of the characteristics, and each leaf node corresponds to one class of fruits. For each unknown object, the decision process is simple: We start from the root node and ask the associated question for the unknown object. We then move down to a different child node based on the answer to this question. This process is repeated until a leaf node is reached. The class label of the reached leaf node is the classification result for the unknown object. This model is normally called a *decision tree* in the literature [34]. If this tree is manually constructed according to human knowledge, it is just a convenient way to represent various rules in a knowledge base. However, if we can automatically learn such a tree model from training data, it is considered to be an interesting method in machine learning, known as *decision trees*. ◆

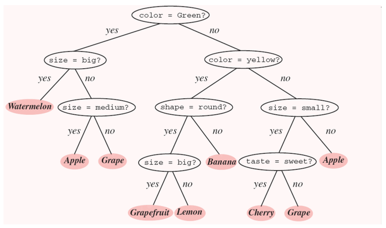

Figure 1.6: An illustration of using a decision tree to recognize various fruits based on some measured features. (Source: [57].)

We will introduce various learning methods for decision trees in Chapter 9.

1.2.4 Parametric versus Nonparametric Models

When we choose a model for a machine learning problem, there are two different types. The so-called *parametric models* (a.k.a. *finite-dimensional models*) are models that take a presumed functional form and are completely determined by a fixed set of model parameters. In the previous curve-fitting example, once we choose to use a linear model (or a fourth-order polynomial model), it can be fully specified by two (or five) coefficients. By definition, both linear and polynomial models are parametric models. In contrast, the so-called *nonparametric models* (a.k.a. *distribution-free models*) do not assume the functional form of the underlying model, and more importantly, the complexity of such a model is not fixed and may depend on the available data. In other words, a nonparametric model cannot be fully specified by a fixed number of parameters. For example, the decision tree is a typical nonparametric model. When we use a decision tree, we do not presume the functional form of the model, and the tree size is usually not fixed as well. If we have more training data, it may allow us to build a larger decision tree. Another well-known nonparametric model is the histogram. When we use a histogram to estimate a data distribution, we do not constrain the shape of the distribution, and the histogram can dramatically change as more and more samples become available.

Generally speaking, it is easier to handle parametric models than nonparametric models because we can always focus on estimating a fixed set of parameters for any parametric model. Parameter estimation is always a much simpler problem than estimating an arbitrary model without knowing of its form.

1.2.5 Overfitting versus Underfitting

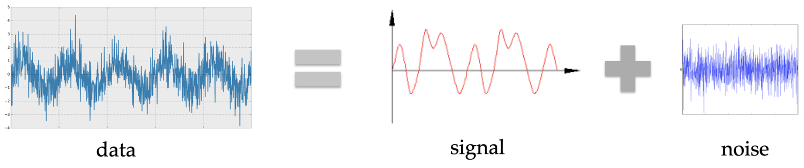

Figure 1.7: An illustration of how data can be conceptually viewed as being composed of signal and noise components.

All machine learning methods rely on training data. Intuitively speaking, training data contain the important information on certain regularities we want to learn with a model, which we informally call the *signal* component. On the other hand, training data also inevitably include some irrelevant or even distracting information, called the *noise* component. A major source of noise is the sampling variations exhibited in any finite set of random samples. If we randomly draw some samples, even from the same distribution, twice, we will not obtain identical samples. This variation can be conceptually viewed as a noise component in the collected data. Of course, noise may also come from measurement or recording errors. In general, we can conceptually represent any collected training data as a combination of two components:

$$\text{data} = \text{signal} + \text{noise}.$$

This decomposition concept is also illustrated in Figure 1.7, where we can see that the signal component represents some regularities in the data, whereas the noise component represents some unpredictable, highly fluctuating residuals. Once we have this conceptual view in mind, we can easily understand two important concepts in machine learning, namely, *underfitting* and *overfitting*.

We will formally introduce the theory behind overfitting in Chapter 5.

Assume we learn a simple model from a set of training data. If the used model is too simple to capture all regularities in the signal component, the learned model will yield very poor results even in the training data, not to mention any unseen data, which is normally called *underfitting*. Figure 1.4 clearly shows an underfitting case, where a linear function is too simple to capture the "up-and-down wiggly pattern" evident in the given data points. On the other hand, if the used model is too complex, the learning process may force a powerful model to perfectly fit the random noise component while trying to catch the regularities in the signal component. Moreover, perfectly fitting the noise component may obstruct the model from capturing all regularities in the signal component because the highly fluctuating noise can distract the learning outcome more when a complex model is used. Even worse, it is useless to perfectly fit the noise component because we will face a completely different noise component in another set of data samples. This will lead to the notorious phenomenon of *overfitting* in machine learning. Continuing with the curve fitting as an example, assume that we use a 10th-order polynomial to fit the given data points in Figure 1.3. After we learn all 11 coefficients, we can create the best-fit 10th-order polynomial model shown in Figure 1.8. As we can see, this model perfectly fits all given training samples but behaves wildly. Our intuition tells us that it yields a much poorer explanation of the data than the model in Figure 1.5.

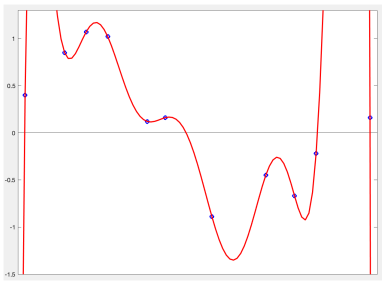

Figure 1.8: An illustration of using a 10th-order polynomial function for the previous curve-fitting problem. The best-fit model behaves wildly because the overfitting happened in the learning process.

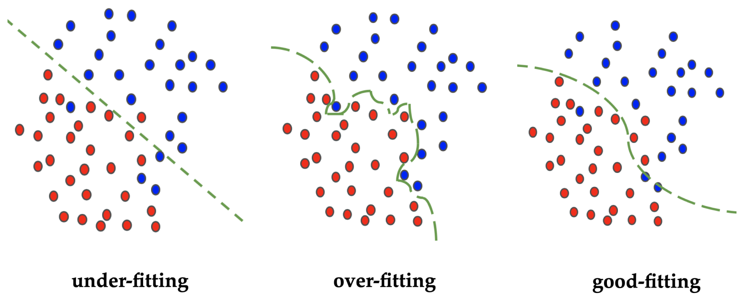

Figure 1.9: An illustration of underfitting and overfitting in a binary classification problem of two classes; the colors indicate class labels.

Not limited to regression, underfitting and overfitting can also occur in classification problems. In the simple classification problem of two classes shown in Figure 1.9, if a simple model is used for learning, it leads to a straight separation boundary between the two classes in the left figure, indicating an underfitting case because many training samples are located on the wrong side of the boundary. On the other hand, if we use a complex model in learning, it may end up with the complicated separation boundary shown in the middle figure. This implies an overfitting case because this boundary perfectly separates all training samples but is not a natural explanation of the data. Finally, among these three cases, the model on the right seems to provide the best explanation of the data set.

We should avoid underfitting and overfitting as much as possible in any machine learning problem because they both hurt the learning performance in one way or another. Underfitting occurs when the learning performance is not satisfactory even in the training set. We can easily get rid of the underfitting problem by increasing the model complexity (i.e., either increasing the number of free parameters or changing to a more complex model). On the other hand, we can identify the overfitting problem if we notice a nearly perfect performance in the training set but a fairly poor performance in another unseen evaluation set. Similarly, we can mitigate overfitting in machine learning either by augmenting more training data, or by reducing the model complexity, or by using so-called *regularization* techniques during the learning process.

We will formally discuss *regularization* in Chapter 7.

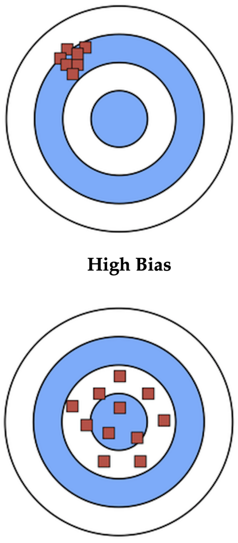

Figure 1.10: An illustration of high bias errors versus high variances in machine learning, where each square represents a learned model from a random training set, and the center of the circles indicates the true regularities to be learned. (Image credit: Sebastian Raschka/CC-BY-SA-4.0.)

1.2.6 Bias–Variance Trade-Off

Generally speaking, the total expected error of a machine learning algorithm on an unseen data set can be decomposed into the following two sources:

- *Bias* due to underfitting:
  The bias error quantifies the inability of a learned model to capture all regularities in the signal component due to erroneous assumptions in the used model. High biases indicate that the learned model consistently misses some important regularities in the data because of inherent weaknesses of the underlying method. As shown in Figure 1.10, each red square conceptually indicates a learned model obtained by running the same learning method on a random training set of equal size. A high bias error implies that the learned model yields a poor match with the regularities in the signal component that are truly relevant to the learning goal.
- *Variance* due to overfitting:
  Variance is the error arising from the learning sensitivity to small fluctuations in the training data. In other words, variance quantifies the overfitting error of a learning method when the learned model is forced to mistakenly capture the randomness in the noise component. As shown in Figure 1.10, when variance is high, all learning results randomly deviate from the true target in a different way because each training set contains a different noise component. High variance indicates that the learned model gives a weak match with the regularities in the signal component as it randomly deviates from the true learning target from one case to another.

We will formally prove the bias and variance decomposition

$$\text{error} = \text{bias}^2 + \text{variance}.$$

in Example 2.2.2.

In precise terms, we can show that the average error of a learning algorithm can be mathematically decomposed as follows:

$$\text{learning error} = \text{bias}^2 + \text{variance}$$

As shown in Figure 1.11, when we have chosen a particular method to learn for a given problem from *a fixed amount* of training data, we cannot reduce the two sources of error at the same time. When we choose a simple model, it usually yields a low variance but a high bias error as a result of underfitting. On the other hand, when we choose a complex model, it can reduce the bias error but leads to higher variance as a result of overfitting. This phenomenon is often called the *bias–variance trade-off* in machine learning. For any particular learning problem, we can usually adjust the model complexity to find the optimal model choice that results in the lowest total learning error.

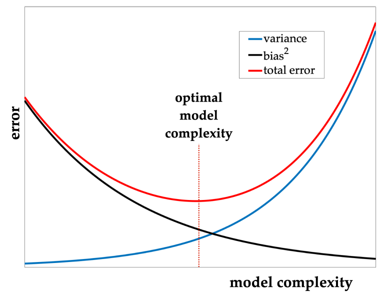

Figure 1.11: An illustration of how to manage the bias–variance trade-off by choosing the optimal model complexity in machine learning.

1.3 General Principles in Machine Learning

In this section, we will cover several general principles in machine learning, providing important insights necessary for understanding some fundamental ideas in machine learning.

1.3.1 Occam's Razor

Occam's razor is a general problem-solving principle in philosophy and science. It is sometimes paraphrased by a statement akin to "the simplest solution is most likely the right one." In the context of machine learning, Occam's razor means a preference for simplicity in model selection. If two different models are observed to yield similar performance on training data, we should prefer the simpler model to the more complicated one. Moreover, the principle of *minimum description length* (MDL) [198] is a formalization of Occam's razor in machine learning, which states that all machine learning methods aim to find regularities in data, and the best model (or hypothesis) to describe the regularities in data is also the one that can compress the data the most.

1.3.2 No-Free-Lunch Theorem

In the context of machine learning, the *no-free-lunch* theorem [253, 57, 220] states that no learning method is universally superior to other methods for all possible learning problems. Given any two machine learning algorithms, if we use them to learn all possible learning problems we can imagine, the average performance of these two algorithms must be the same. Or even worse, their average performance is no better than random guessing.

We can use the earlier curve-fitting problem as an example to explain why the no-free-lunch theorem makes sense. Given the training samples in Figure 1.3, our goal is to create a model to predict function values for other x points. No matter what learning method we use, we eventually end up with an estimated model, such as the red curve in Figure 1.12. Because we have no knowledge of the ground-truth function $y = f(x)$ other than the training samples, theoretically speaking, the ground-truth function $y = f(x)$ could take any arbitrary value for a new point, which is not in the training set. When we use the estimated model to predict function values at some new points, say, x_1 and x_2, it is easy to see that the estimated model yields a good prediction if the ground-truth function $y = f(x)$ happens to yield "good" values (as indicated by green dots in Figure 1.12). However, we can always imagine another scenario where the ground-truth function yields "bad" values (as indicated by red squares in Figure 1.12), for which the estimated model will give a very poor prediction. This is true no matter what learning algorithm we use to estimate the model. If we average the prediction performance of any estimated model over all possible scenarios for the ground-truth function, the average performance is close to a random guess because for each good-prediction case, we can also come up with any number of bad-prediction cases.

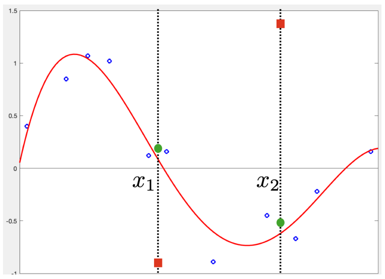

Figure 1.12: An illustration of the no-free-lunch theorem in a simple curve-fitting problem: when an estimated model (red curve) is used to predict function values at x_1 and x_2, it works well for some target functions (green dots), but meanwhile, it will work poorly for other functions (red squares).

The no-free-lunch theorem simply says that no machine learning algorithm can learn anything useful *merely* from the training data. If a machine learning method works well for some problems, the method must have explicitly or implicitly used other knowledge of the underlying problems beyond the training data.

1.3.3 Law of the Smooth World

Despite the aforementioned no-free-lunch theorem, a fundamental reason why many machine learning methods thrive in practice is that our physical world is always smooth. Because of the hard constraints that exist in reality, such as energy and power, any physical process in the macro world is smooth in nature (e.g., audios, images, videos). Furthermore, our intuition and perception are all built on top of the *law of the smooth world*. Therefore, if we use machine learning to tackle any problems arising from the real world, the law of the smooth world is always applicable, dramatically simplifying many of our learning problems at hand.

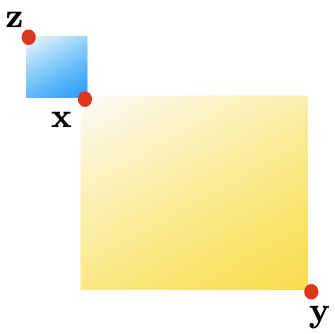

Figure 1.13: An illustration of why the law of the smooth world can simplify a machine learning problem.

For example, as shown in Figure 1.13, assume that a training set contains some measurements of a physical process at three points in the space, that is, $\mathbf{x}$, $\mathbf{y}$, and $\mathbf{z}$, where $\mathbf{x}$ and $\mathbf{y}$ are located far apart, whereas $\mathbf{x}$ and $\mathbf{z}$ are close by. If we need to learn a model to predict the process in the yellow region between $\mathbf{x}$ and $\mathbf{y}$, it is a hard problem because the training data do not provide any information for this, and many unpredictable things could happen within such a wide range. On the other hand, if we need to predict this process in the blue region between two nearby points, it should be relatively simple because the law of the smooth world significantly restricts the behavior of the process within such a narrow region given the two observations at **x** and **z**. In fact, some machine learning models can be built to give fairly accurate predictions in the blue region by simply interpolating these two observations at **x** and **z**. The exact prediction accuracy actually depends on the smoothness of the underlying process. In machine learning, such smoothness is often mathematically quantified using the concept of *Lipschitz continuity* (see margin note) or a more recent notion of *bandlimitedness* [115].

A function $f(x)$ is said to be *Lipschitz continuous* if there exists a real constant $L > 0$, for any two x_1 and x_2, where

$$\left|f(x_1) - f(x_2)\right| \leq L\left|x_1 - x_2\right|$$

always holds.

Moreover, let us go back to the no-free-lunch example in Figure 1.12. If we have enough training samples to ensure that the gaps between all samples are small enough, then many "bad" values as assumed by the no-free-lunch theorem will not actually occur in practice because they violate the law of the smooth world. As a result, when we only average all plausible scenarios in practice, suitable machine learning methods achieve much better prediction accuracy than random guessing.

Furthermore, the law of the smooth world immediately suggests a simple strategy for machine learning. Given any unknown observation, if we search over all known samples in the training set, the prediction for the unknown can be made based on the nearest sample in the training set. This leads to the famous *nearest neighbors* (NN) algorithm. In order to deal with some possible outliers in the training set, this algorithm can be extended to a more robust version, namely, the k-*nearest neighbors* (k-NN) algorithm.

Example 1.3.1 k-NN for Classification

For each unknown object, we search the whole training set to find the top k nearest neighbors, where k is a small positive integer to be manually specified beforehand. The class label of the unknown object is determined by a majority vote of these k-NN. If we choose $k = 1$, the object is simply assigned the class of the single nearest neighbor.

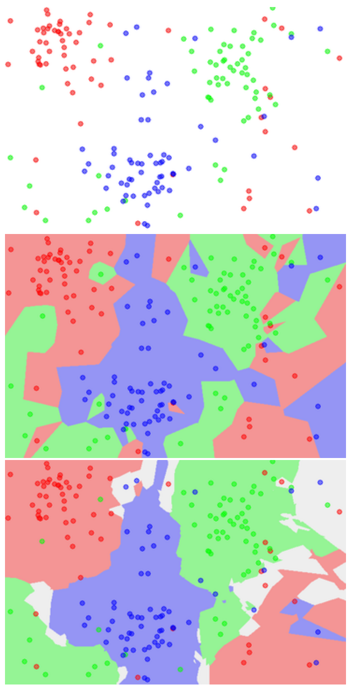

Figure 1.14: An illustration of the decision boundary of the k-nearest neighbors (k-NN) algorithm for classification:
Top panel: Three-class data (labeled by color). Middle panel: Boundary of 1-NN ($k = 1$). Bottom panel: Boundary of 5-NN ($k = 5$). (Image credit: Agor153/CC-BY-SA-3.0.)

The k-NN method is conceptually simple and intuitive, and it can yield the decision boundary in the entire space based on any given training set, as shown in Figure 1.14. In many cases, the simple k-NN method can yield satisfactory classification performance. In general, the success of the k-NN method depends on two factors:

- Whether we have a good similarity measure to properly compute the distance between any two objects in the space. This topic is usually studied in a subfield of machine learning called *metric learning* [255, 136].

- Whether we have enough samples in the training set to sufficiently cover all regions in the space.

In terms of how many samples are needed to ensure good performance for the k-NN method, some theoretical analysis [220] has shown that if we want to achieve an error rate below ϵ $(0 < \epsilon < 1)$, the minimum number of training samples N required by the k-NN algorithm increases exponentially with the dimensionality of the space, denoted as d, as follows:

See Exercise Q1.1.

$$N \propto \left(\frac{\sqrt{d}}{\epsilon}\right)^{d+1}.$$

Assume we need 100 samples to achieve an error rate $\epsilon = 0.01$ for a problem in a low-dimension space (e.g., $d = 3$). But for some similar problems in a higher-dimensional space, we need a huge number of training samples in order to achieve the same performance. For example, we may need roughly 2×10^8 training samples for a similar problem in a 10-dimension space and about 7×10^{123} training samples for a similar problem in a 100-dimension space. Obviously, these numbers are prohibitively large for any practical system. This result shows that the k-NN method can effectively solve problems in a low-dimensional space but will encounter challenges when the dimensionality of problems increases. In fact, this problem is not just limited to the k-NN method but implies another general principle in machine learning, known as the *curse of dimensionality*. ♦

1.3.4 Curse of Dimensionality

In machine learning, the *curse of dimensionality* refers to the dilemma of learning in high-dimensional spaces. As shown in the previous k-NN example, as the dimensionality of learning problems grows, the volume of the underlying space increases exponentially. This typically requires an exponentially increasing amount of training data and computing resources to ensure the effectiveness of any learning methods. Moreover, our intuition of the three-dimensional physical world often fails in high dimensions [54]. The similarity-based reasoning breaks down in high dimensions as the distance measures become unreliable and counterintuitive. For example, if many samples are uniformly placed inside a unit hypercube in a high-dimensional space, it is proven that most of these samples are closer to a face of the hypercube than to their nearest neighbors.

However, the worst-case scenarios predicted by the curse of dimensionality normally occur when the data are uniformly distributed in high-dimensional spaces. Most real-world learning problems involve high-dimensional data, but the good news is that real-world data never spread evenly throughout the high-dimensional spaces. This observation is often referred to as the *blessing of nonuniformity* [54]. The blessing of nonuniformity essentially allows us to be able to effectively learn these high-dimensional problems using a reasonable amount of training data and computing resources. A nonuniform data distribution suggests that all dimensions of the data are not independent but highly correlated in such a way that many dimensions are redundant. In other words, many dimensions can be discarded without losing much information about the data distribution. This idea motivates a group of machine learning methods called *dimensionality reduction*. Alternatively, a nonuniform distribution in a high-dimensional space also suggests that the real data are only concentrated in a linear subspace or a lower-dimensional nonlinear subspace, which is often called a *manifold*. In machine learning, the so-called *manifold learning* aims to identify such lower-dimensional topological spaces where high-dimensional data are congregated.

We will introduce various dimensionality-reduction methods and manifold learning in Chapter 4.

1.4 Advanced Topics in Machine Learning

This book aims to introduce only the basic principles and methods of machine learning, mainly focusing on the well-established supervised learning methods. Chapter 3 further sketches out these topics. This section briefly lists other advanced topics in machine learning that will not be fully covered in this book. These short summaries serve as an entry point for interested readers to further explore these topics in future study.

1.4.1 Reinforcement Learning

Reinforcement learning [234] is an area in machine learning that is concerned with how to teach a computer agent to take the best possible actions in a long interaction course with an unknown environment. Different from the standard supervised learning, the learning agent in a reinforcement learning setting does not receive any strong supervision from the environment regarding what the best action is at each step. Instead, the agent only occasionally receives some numerical rewards (positive or negative). The goal in reinforcement learning is to learn what action should be taken under each condition, often called *policy*, in order to maximize the notion of a cumulative reward over the long term. Traditionally, some numerical tables are used to represent the expected cumulative rewards of various actions under each policy, leading to the so-called *Q-learning* [248]. More recently, neural networks have been used as a function approximator to compute the expected cumulative rewards. These methods are sometimes called *deep reinforcement learning* (a.k.a. *deep Q-learning*) [166].

Reinforcement learning represents a general learning framework, but it is regarded as an extremely challenging task because a learning agent must learn how to explore potentially huge search spaces only based on weak reward signals. With the help of neural networks, the deep reinforcement learning methods have recently achieved some notable successes in several closed-ended gaming settings, such as Atari video games [167] and the ancient board game Go [224], but it still remains unclear how to extend these methods to cope with open-ended tasks in a real-world environment.

1.4.2 Meta-Learning

Meta-learning (a.k.a. *learning to learn*) is a subfield of machine learning that studies how to design automatic learning algorithms to improve the performance of existing learning algorithms or to learn the algorithm itself based on some meta-data about previous learning experiments. The meta-data may include hyperparameter settings, model structures (e.g., pipeline compositions or network architectures), the learned model parameters, accuracy, and training time, as well as other measurable properties of the learning tasks [241]. Next, another optimizer, also called the *meta-learner*, is used to learn from the meta-data in order to extract knowledge and guide the search for optimal models for new tasks.

The hyperparameters of a learning algorithm are the parameters that must be manually specified prior to automatic learning (e.g., the value of k in the k-NN algorithm).

1.4.3 Causal Inference

As we know, humans often rationalize the world in terms of cause and effect, that is, the so-called causal relations between variables or events. On the other hand, typical machine learning methods can only examine the statistical correlations in data. It is well known that correlation is not equal to causation. *Causal inference* is an area of machine learning that focuses on the process of drawing causal connections between variables in order to gain a better understanding of the physical world [183, 184, 186].

1.4.4 Other Advanced Topics

Transfer learning [190] is another subfield in machine learning that focuses on how to efficiently adapt an existing machine learning model, which has learned to perform well in one domain, to a different but related domain. Hence, it is also called *domain adaption* [143, 19], which was initially studied extensively for speaker adaption in speech recognition in the 1980s [37, 77, 144].

https://doi.org/10.1017/9781108938051.003 Published online by Cambridge University Press

Online learning methods [105] focus on scenarios where training data become available in a sequential order. In this case, each data sample is used to update the model as soon as it becomes available. Ideally, an online learning method does not need to store all previous data after the model has been updated so that it can also be used in some learning problems where it is computationally infeasible to train over the entire data set.

Active learning methods [219, 58] study a special case of machine learning in which a learning algorithm can interactively query a teacher to obtain necessary supervision information for desired inputs. The goal in active learning is to make the best use of proactive queries in order to learn models in the most efficient way.

Imitation learning techniques [106] aim to mimic human behaviors for a given task. A learning agent is trained to perform a task from some demonstrations by learning a mapping between observations and actions. Like reinforcement learning, imitation learning also aims to learn how to make a sequence of decisions in an unknown environment. The difference is that it is learned by observing some demonstrations rather than maximizing a cumulative reward. Therefore, imitation learning is often used in cases where the proper reward signals are difficult to specify.

Exercises

Q1.1 Is the k-NN method parametric or nonparametric? Explain why.

Q1.2 A real-valued function $f(x)$ ($x \in \mathbb{R}$) is said to be Lipschitz continuous if there exists a real constant $L > 0$, for any two points $x_1 \in \mathbb{R}$ and $x_2 \in \mathbb{R}$, where

$$\left|f(x_1) - f(x_2)\right| \leq L\,|x_1 - x_2|$$

always holds. If $f(x)$ is differentiable, prove that $f(x)$ is Lipschitz continuous if and only if

$$\left|f'(x)\right| \leq L$$

holds for all $x \in \mathbb{R}$.