1 Introduction

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.

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.

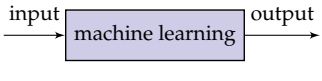

Figure 1.2: A system view of any machine learning problem.

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.

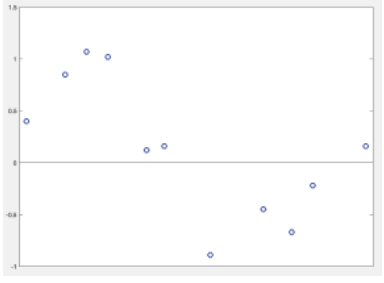

Figure 1.3: An illustration of a curve-fitting problem, which can be viewed as a regression problem in machine learning.

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.

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.

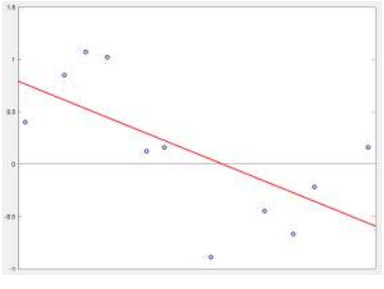

Figure 1.4: An illustration of using a linear model for the curve-fitting problem shown in Figure 1.3.

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. ♦

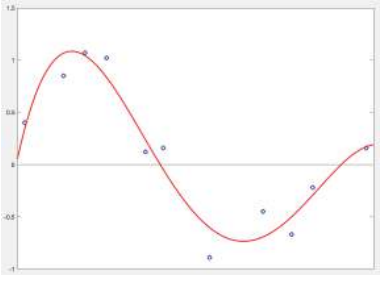

Figure 1.5: An illustration of using a fourth-order polynomial function for the curve-fitting problem.

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*. ♦

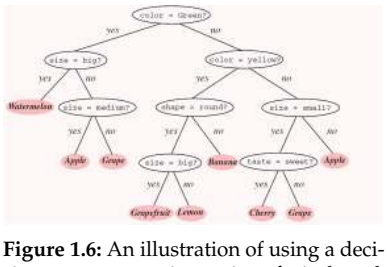

Figure 1.6: An illustration of using a decision tree to recognize various fruits based on some measured features. (Source: [57].)

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

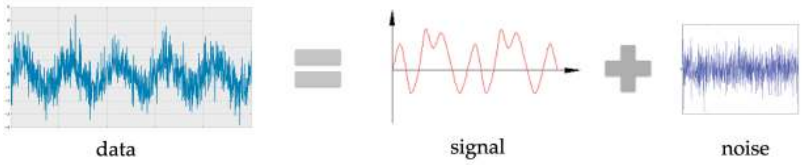

Figure 1.7: An illustration of how data can be conceptually viewed as being composed of signal and noise components.

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.

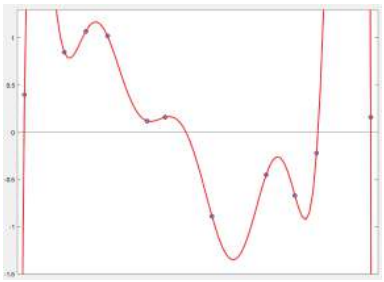

Figure 1.8: An illustration of using a 10th-order polynomial function for the previous curve-fitting problem. The best-fit model behaves wildly because the over-fitting happened in the learning process.

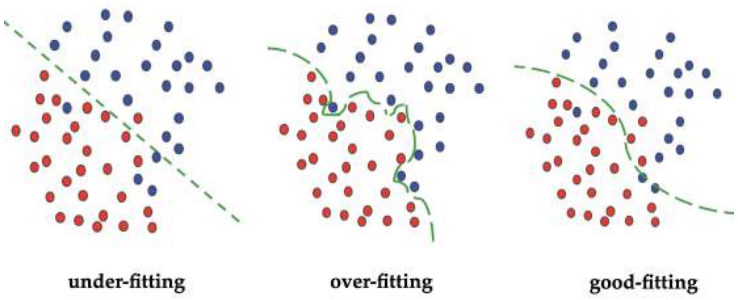

Figure 1.9: An illustration of underfitting and overfitting in a binary classification problem of two classes; the colors indicate class labels.

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.

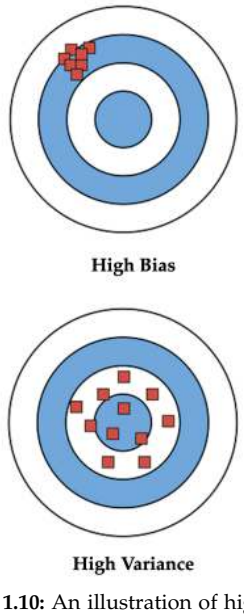

Figure 1.10: An illustration of high bias errors versus high variances in machine learning, where each square represents a learned model from a random training set, and the center of the circles indicates the true regularities to be learned. (Image credit: Sebastian Raschka/CC-BY-SA-4.0.)

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.

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

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

We will formally prove the bias and variance decomposition

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

in Example 2.2.2.