1 Introduction to Machine Learning

1.1 Introduction

Machine learning is a unified algorithmic framework designed to identify computational models that accurately describe empirical data and the phenomena underlying it, with little or no human involvement. While still a young discipline with much more awaiting discovery than is currently known, today machine learning can be used to teach computers to perform a wide array of useful tasks including automatic detection of objects in images (a crucial component of driver-assisted and self-driving cars), speech recognition (which powers voice command technology), knowledge discovery in the medical sciences (used to improve our understanding of complex diseases), and predictive analytics (leveraged for sales and economic forecasting), to just name a few.

In this chapter we give a high-level introduction to the field of machine learning as well as the contents of this textbook.

1.2 Distinguishing Cats from Dogs: a Machine Learning Approach

To get a big-picture sense of how machine learning works, we begin by discussing a toy problem: teaching a computer how to distinguish between pictures of *cats* from those with *dogs*. This will allow us to informally describe the terminology and procedures involved in solving the typical machine learning problem.

Do you recall how you first learned about the difference between cats and dogs, and how they are different animals? The answer is probably no, as most humans learn to perform simple cognitive tasks like this very early on in the course of their lives. One thing is certain, however: young children do not need some kind of formal scientific training, or a zoological lecture on *felis catus* and *canis familiaris* species, in order to be able to tell cats and dogs apart. Instead, they learn by example. They are naturally presented with many images of what they are told by a *supervisor* (a parent, a caregiver, etc.) are either cats or dogs, until they fully grasp the two concepts. How do we know when a child can successfully distinguish between cats and dogs? Intuitively, when

they encounter new (images of) cats and dogs, and can correctly identify each new example or, in other words, when they can *generalize* what they have learned to new, previously unseen, examples.

Like human beings, computers can be taught how to perform this sort of task in a similar manner. This kind of task where we aim to teach a computer to distinguish between different types or *classes* of things (here *cats* and *dogs*) is referred to as a *classification* problem in the jargon of machine learning, and is done through a series of steps which we detail below.

1. Data collection. Like human beings, a computer must be trained to recognize the difference between these two types of animals by learning from a batch of examples, typically referred to as a *training set* of data. Figure 1.1 shows such a training set consisting of a few images of different cats and dogs. Intuitively, the larger and more diverse the training set the better a computer (or human) can perform a learning task, since exposure to a wider breadth of examples gives the learner more experience.

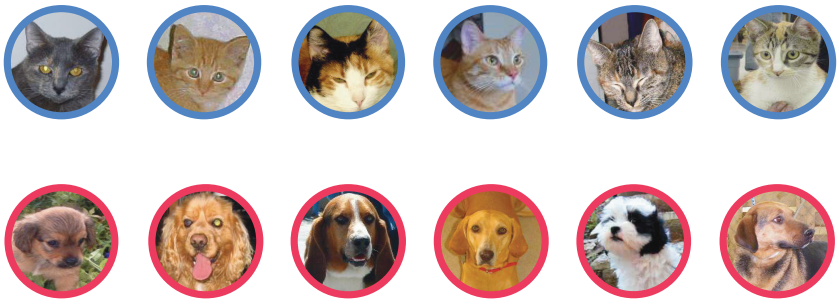

Figure 1.1 A training set consisting of six images of cats (highlighted in blue) and six images of dogs (highlighted in red). This set is used to train a machine learning model that can distinguish between future images of cats and dogs. The images in this figure were taken from [1].

2. Feature design. Think for a moment about how we (humans) tell the difference between images containing cats from those containing dogs. We use color, size, the shape of the ears or nose, and/or some combination of these *features* in order to distinguish between the two. In other words, we do not just look at an image as simply a collection of many small square pixels. We pick out grosser details, or features, from images like these in order to identify what it is that we are looking at. This is true for computers as well. In order to successfully train a computer to perform this task (and any machine learning task more generally)

we need to provide it with properly designed features or, ideally, have it find or *learn* such features itself.

Designing quality features is typically not a trivial task as it can be very application dependent. For instance, a feature like *color* would be less helpful in discriminating between cats and dogs (since many cats and dogs share similar hair colors) than it would be in telling grizzly bears and polar bears apart! Moreover, extracting the features from a training dataset can also be challenging. For example, if some of our training images were blurry or taken from a perspective where we could not see the animal properly, the features we designed might not be properly extracted.

However, for the sake of simplicity with our toy problem here, suppose we can easily extract the following two features from each image in the training set: *size of nose* relative to the size of the head, ranging from small to large, and *shape of ears*, ranging from round to pointy.

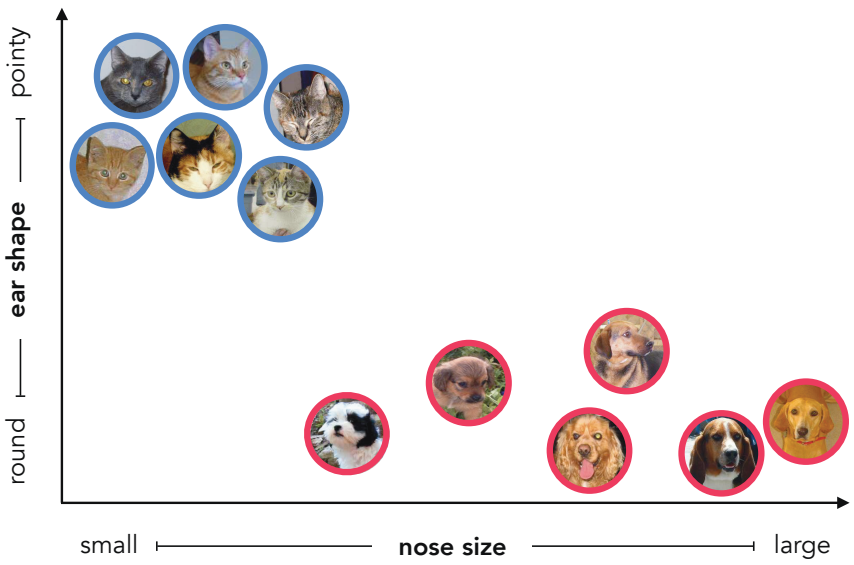

Figure 1.2 Feature space representation of the training set shown in Figure 1.1 where the horizontal and vertical axes represent the features *nose size* and *ear shape*, respectively. The fact that the cats and dogs from our training set lie in distinct regions of the feature space reflects a good choice of features.

Examining the training images shown in Figure 1.1, we can see that all cats have small noses and pointy ears, while dogs generally have large noses and round ears. Notice that with the current choice of features each image can now be represented by just two numbers: a number expressing the relative nose size, and another number capturing the pointiness or roundness of the ears. In other words, we can represent each image in our training set in a two-dimensional

feature space where the features *nose size* and *ear shape* are the horizontal and vertical coordinate axes, respectively, as illustrated in Figure 1.2.

3. Model training. With our feature representation of the training data the machine learning problem of distinguishing between cats and dogs is now a simple geometric one: have the machine find a line or a curve that separates the cats from the dogs in our carefully designed feature space. Supposing for simplicity that we use a line, we must find the right values for its two parameters – a slope and vertical intercept – that define the line's orientation in the feature space. The process of determining proper parameters relies on a set of tools known as *mathematical optimization* detailed in Chapters 2 through 4 of this text, and the tuning of such a set of parameters to a training set is referred to as the training of a model.

Figure 1.3 shows a trained linear model (in black) which divides the feature space into cat and dog regions. This linear model provides a simple computational rule for distinguishing between cats and dogs: when the feature representation of a future image lies above the line (in the blue region) it will be considered a cat by the machine, and likewise any representation that falls below the line (in the red region) will be considered a dog.

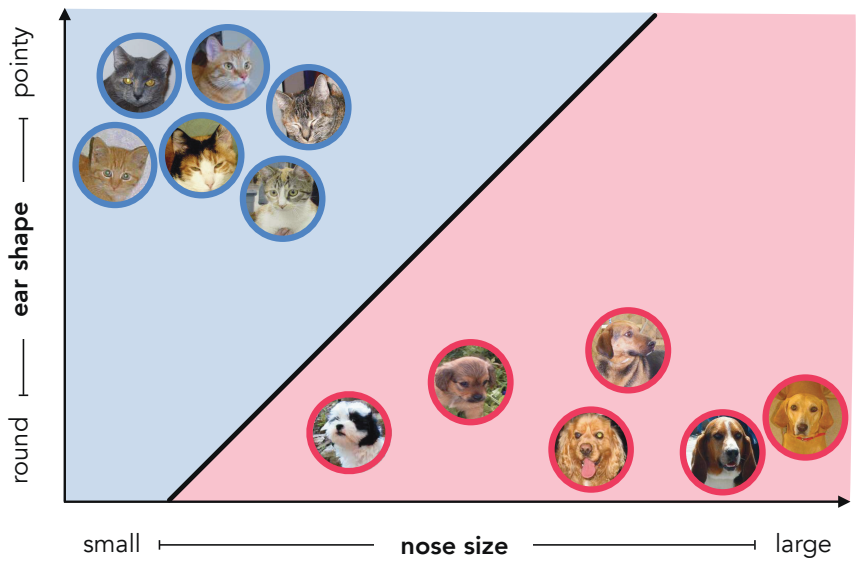

Figure 1.3 A trained linear model (shown in black) provides a computational rule for distinguishing between cats and dogs. Any new image received in the future will be classified as a cat if its feature representation lies above this line (in the blue region), and a dog if the feature representation lies below this line (in the red region).

Figure 1.4 A validation set of cat and dog images (also taken from [1]). Notice that the images in this set are not highlighted in red or blue (as was the case with the training set shown in Figure 1.1) indicating that the true identity of each image is not revealed to the learner. Notice that one of the dogs, the Boston terrier in the bottom right corner, has both a small nose and pointy ears. Because of our chosen feature representation the computer will think this is a cat!

4. Model validation. To validate the efficacy of our trained learner we now show the computer a batch of previously unseen images of cats and dogs, referred to generally as a *validation set* of data, and see how well it can identify the animal in each image. In Figure 1.4 we show a sample validation set for the problem at hand, consisting of three new cat and dog images. To do this, we take each new image, extract our designed features (i.e., nose size and ear shape), and simply check which side of our line (or *classifier*) the feature representation falls on. In this instance, as can be seen in Figure 1.5, all of the new cats and all but one dog from the validation set have been identified correctly by our trained model.

The misidentification of the single dog (a Boston terrier) is largely the result of our choice of features, which we designed based on the training set in Figure 1.1, and to some extent our decision to use a *linear* model (instead of a *nonlinear* one). This dog has been misidentified simply because its features, a small nose and pointy ears, match those of the cats from our training set. Therefore, while it first appeared that a combination of nose size and ear shape could indeed distinguish cats from dogs, we now see through validation that our training set was perhaps too small and not diverse enough for this choice of features to be completely effective in general.

We can take a number of steps to improve our learner. First and foremost we should collect more data, forming a larger and more diverse training set. Second, we can consider designing/including more discriminating features (perhaps eye color, tail shape, etc.) that further help distinguish cats from dogs using a linear model. Finally, we can also try out (i.e., train and validate) an array of *nonlinear* models with the hopes that a more complex rule might better distinguish between cats and dogs. Figure 1.6 compactly summarizes the four steps involved in solving our toy cat-versus-dog classification problem.

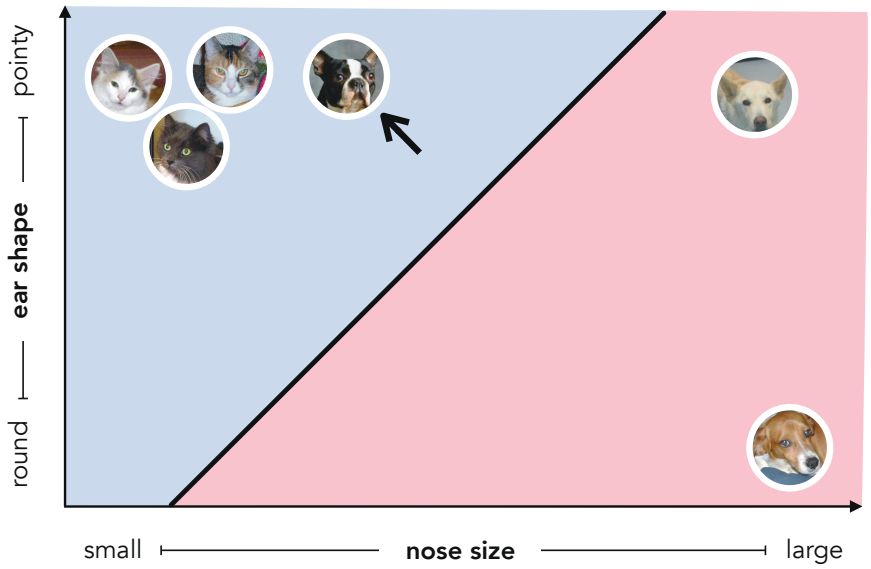

Figure 1.5 Identification of (the feature representation of) validation images using our trained linear model. The Boston terrier (pointed to by an arrow) is *misclassified* as a cat since it has pointy ears and a small nose, just like the cats in our training set.

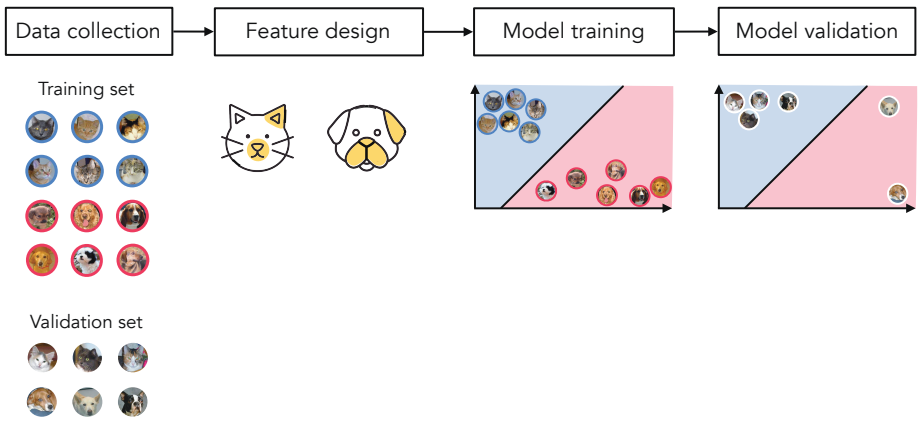

Figure 1.6 The schematic pipeline of our toy cat-versus-dog classification problem. The same general pipeline is used for essentially all machine learning problems.

1.3 The Basic Taxonomy of Machine Learning Problems

The sort of computational rules we can learn using machine learning generally fall into two main categories called *supervised* and *unsupervised* learning, which we discuss next.

1.3.1 Supervised learning

Supervised learning problems (like the prototypical problem outlined in Section 1.2) refer to the automatic learning of computational rules involving input/output relationships. Applicable to a wide array of situations and data types, this type of problem comes in two forms, called *regression* and *classification*, depending on the general numerical form of the output.

Regression

Suppose we wanted to predict the share price of a company that is about to go public. Following the pipeline discussed in Section 1.2, we first gather a training set of data consisting of a number of corporations (preferably active in the same domain) with known share prices. Next, we need to design feature(s) that are thought to be relevant to the task at hand. The company's revenue is one such potential feature, as we can expect that the higher the revenue the more expensive a share of stock should be. To connect the share price (output) to the revenue (input) we can train a simple linear model or *regression line* using our training data.

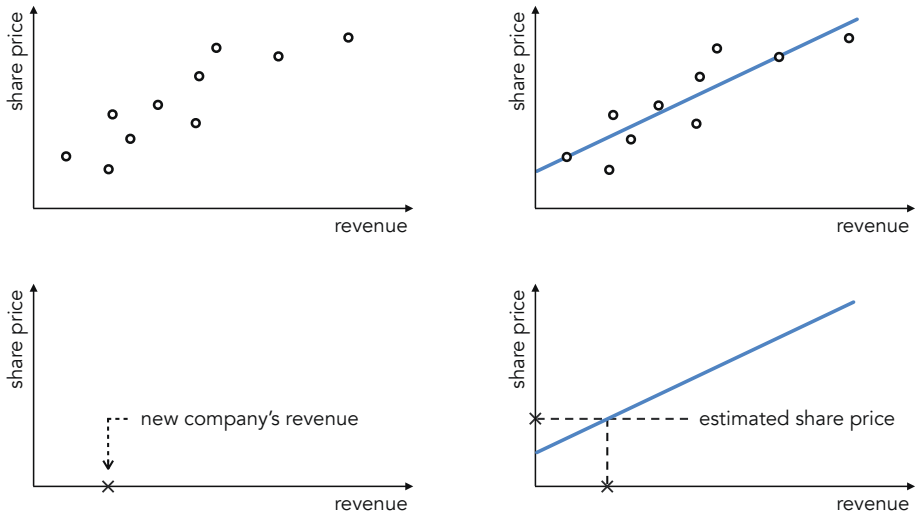

Figure 1.7 (top-left panel) A toy training dataset consisting of ten corporations' share price and revenue values. (top-right panel) A linear model is fit to the data. This trend line models the overall trajectory of the points and can be used for prediction in the future as shown in the bottom-left and bottom-right panels.

The top panels of Figure 1.7 show a toy dataset comprising share price versus revenue information for ten companies, as well as a linear model fit to this data. Once the model is trained, the share price of a new company can be predicted

based on its revenue, as depicted in the bottom panels of this figure. Finally, comparing the predicted price to the actual price for a validation set of data we can test the performance of our linear regression model and apply changes as needed, for example, designing new features (e.g., total assets, total equity, number of employees, years active, etc.) and/or trying more complex nonlinear models.

This sort of task, i.e., fitting a model to a set of training data so that predictions about a *continuous-valued* output (here, share price) can be made, is referred to as regression. We begin our detailed discussion of regression in Chapter 5 with the linear case, and move to nonlinear models starting in Chapter 10 and throughout Chapters 11–14. Below we describe several additional examples of regression to help solidify this concept.

Example 1.1 The rise of student loan debt in the United States

Figure 1.8 (data taken from [2]) shows the total student loan debt (that is money borrowed by students to pay for college tuition, room and board, etc.) held by citizens of the United States from 2006 to 2014, measured quarterly. Over the eight-year period reflected in this plot the student debt has nearly tripled, totaling over one trillion dollars by the end of 2014. The regression line (in black) fits this dataset quite well and, with its sharp positive slope, emphasizes the point that student debt is rising dangerously fast. Moreover, if this trend continues, we can use the regression line to predict that total student debt will surpass two trillion dollars by the year 2026 (we revisit this problem later in Exercise 5.1).

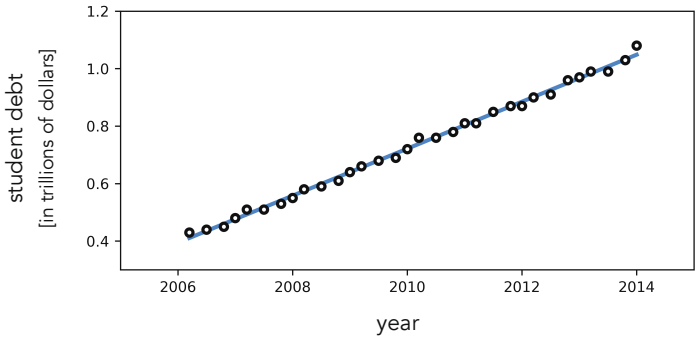

Figure 1.8 Figure associated with Example 1.1, illustrating total student loan debt in the United States measured quarterly from 2006 to 2014. The rapid increase rate of the debt, measured by the slope of the trend line fit to the data, confirms that student debt is growing very fast. See text for further details.

Example 1.2 Kleiber's law

Many natural laws in the sciences take the form of regression models. For example, after collecting a considerable amount of data comparing the body mass versus metabolic rate (a measure of at-rest energy expenditure) of a variety of animals, early twentieth-century biologist Max Kleiber found that the log of these two quantities are related *linearly*. This linear relationship can be seen visually by examining the dataset shown in Figure 1.9. Examining a similar dataset, Kleiber found the slope of the regression line to be around $\frac{3}{4}$ or, in other words, that metabolic rate $\propto$ mass$^{\frac{3}{4}}$.

This sublinear relationship means that compared to smaller-bodied species (like birds), larger-bodied species (like walruses) have lower metabolic rates, which is consistent with having lower heart rates and larger life spans (we revisit this problem later in Exercise 5.2).

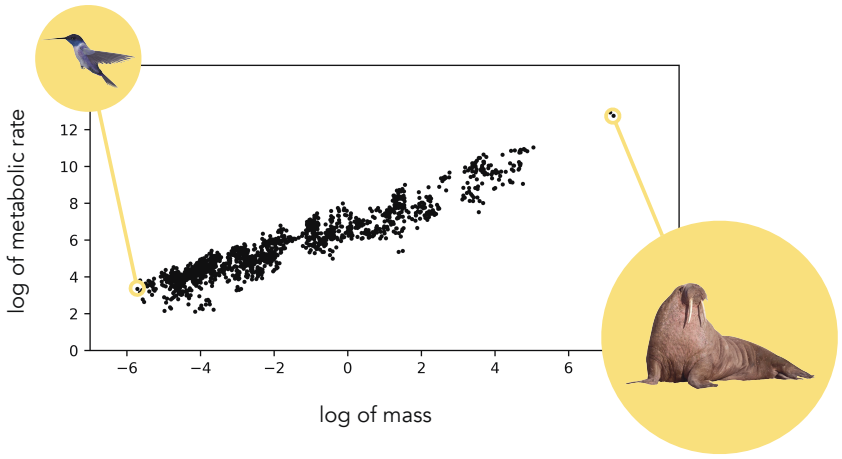

Figure 1.9 Figure associated with Example 1.2. A large set of body mass versus metabolic rate data points, transformed by taking the log of each value, for various animals over a wide range of different masses. See text for further details.

Example 1.3 Predicting box office success

In 1983 the Academy award winning screenwriter William Goldman coined the phrase "nobody knows anything" in his book *Adventures in the Screen Trade*, referring to his belief that at the time it was impossible for anyone to predict the success or failure of Hollywood movies. While this may be true – in the era of the internet – by leveraging data like the quantity of internet searches for a movie's trailer, as well as the amount of discussion about a movie on social networks (see, e.g., [3, 4]), machine learning can accurately predict opening box office revenue for certain films. Sales forecasting for a range of products/services including box office sales is often performed using regression since the output to be predicted is continuous (enough) in nature.

Example 1.4 Business and industrial applications
Examples of regression are plentiful in business and industry. For instance, using regression to accurately predict the *price* of consumer goods (from electronics, to automobiles, to houses) are hugely valuable enterprises unto themselves (explored further in Example 5.5). Regression is also commonly used in industrial applications to better understand a given system, e.g., how the configuration of an automobile affects its performance (see Example 5.6), so that such processes can be optimized.

Classification

The machine learning task of classification is similar in principle to that of regression, with the key difference between the two being that instead of predicting a continuous-valued output, with classification the output we aim at predicting takes on *discrete values* or *classes*. Classification problems arise in a host of forms. For example, object recognition where different objects from a set of images are distinguished from one another (e.g., handwritten digits for the automatic sorting of mail or street signs for driver-assisted and self-driving cars) is a very popular classification problem. The toy problem of distinguishing cats from dogs discussed in Section 1.2 falls into this bucket as well. Other common classification problems include speech recognition (recognizing different spoken words for voice recognition systems), determining the general sentiment of a social network like Twitter towards a particular product or service, as well as determining what kind of hand gesture someone is making from a finite set of possibilities (for use, for instance, in controlling a computer without a mouse).

Geometrically speaking, a common way of viewing the task of classification in two dimensions is one of finding a separating line (or, more generally, a separating curve) that accurately separates two kinds of data.[1] This is precisely the perspective on classification we took in describing the toy example in Section 1.2, where we used a line to separate (features extracted from) images of cats and dogs. New data from a validation set are then automatically classified by simply determining which side of the line the data lies on. Figure 1.10 illustrates the concept of a linear model or classifier used for performing classification on a two-dimensional toy dataset.

Many classification problems (e.g., handwritten digit recognition, discussed below) have naturally more than two classes. After describing linear two-class classification in Chapter 6 we detail linear multi-class classification in Chapter 7. The nonlinear extension of both problems is then described starting in Chapter 10 and throughout Chapters 11–14. Below we briefly describe several further examples of classification to help solidify this concept.

[1] In higher dimensions we likewise aim at determining a separating linear hyperplane (or, more generally, a nonlinear manifold).

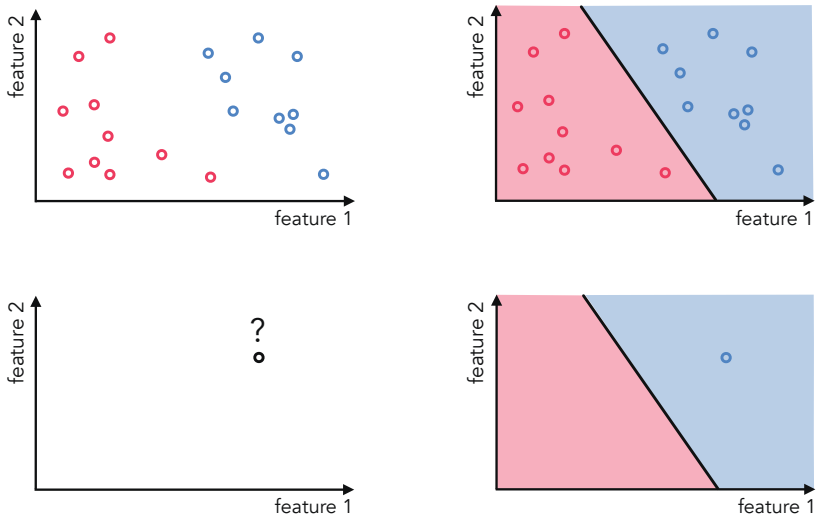

Figure 1.10 (top-left panel) A toy two-dimensional training set of data consisting of two distinct classes: *red* and *blue*. (top-right panel) A linear model is trained to separate the two classes. (bottom-left panel) A validation point whose class is unknown. (bottom-right panel) The validation point is classified as *blue* since it lies on the blue side of the trained linear classifier.

Example 1.5 Object detection

Object detection, a common classification problem (see, e.g., [5, 6, 7]), is the task of automatically identifying a specific object in a set of images or videos. Popular object detection applications include the detection of faces in images for organizational purposes and camera focusing, pedestrians for autonomous driving vehicles, and faulty components for automated quality control in electronics production. The same kind of machine learning framework, which we highlight here for the case of face detection, can be utilized for solving many such detection problems.

After training a (linear) classifier on a set of training data consisting of facial and nonfacial images, faces are sought after in a new validation image by sliding a (typically) square window over the entire image. At each location of the sliding window the image content inside is examined to see which side of the classifier it lies on. This is illustrated in Figure 1.11. If the (feature representation of the) content lies on the face side of the classifier the content is classified as a face, otherwise a nonface.

Example 1.6 Sentiment analysis

The rise of social media has significantly amplified the voice of consumers, providing them with an array of well-tended outlets on which to comment, discuss, and rate products and services (see, e.g., [8]). This has led many firms to seek out

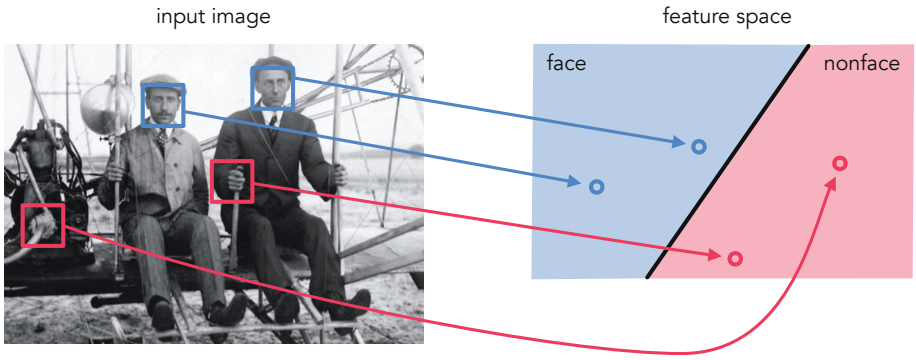

Figure 1.11 Figure associated with Example 1.5. To determine if any faces are present in an input image (in this instance an image of the Wright brothers, inventors of the airplane, sitting together in one of their first motorized flying machines in 1908) a small window is scanned across its entirety. The content inside the box at each instance is determined to be a face by checking which side of the learned classifier the feature representation of the content lies. In the figurative illustration shown here the area above and below the learned classifier (shown in blue and red) are the face and nonface sides of the classifier, respectively. See text for further details.

data intensive methods for gauging their customers' feelings towards recently released products, advertising campaigns, etc. Determining the aggregated feelings of a large base of customers, using text-based content like product reviews, tweets, and comments, is commonly referred to as *sentiment analysis*. Classification models are often used to perform sentiment analysis, learning to identify consumer data of either positive or negative feelings. We discuss this problem further in Example 9.1.

Example 1.7 Computer-aided diagnosis of medical conditions

A myriad of two-class classification problems naturally arise in medical settings when a healthcare professional looks to determine whether or not a patient suffers from a particular malady, and when medical researchers conduct research into what features distinguish those suffering from such a malady from those who do not (in the hopes that a remedy can then be devised to ameliorate the malady based on these features). The modality of these sorts of experiments varies widely, from statistical measurements of affected areas (e.g., the shape and area of biopsied tumorous tissue; see Exercise 6.13), to biochemical markers, to information derived from radiological image data, to genes themselves (as explored further in Example 11.18).

For instance, paired with a classification algorithm functional Magnetic Resonance Imaging (fMRI) of the brain is an increasingly useful method for diagnosing neurological disorders such as Autism, Alzheimer's, and Attention Deficit Hyperactivity Disorder (ADHD). To perform classification a dataset is acquired consisting of statistically based features extracted from fMRI brain scans of pa-

tients suffering from one such previously mentioned cognitive disorder, as well as individuals from a control group who are not afflicted. These fMRI brain scans capture neural activity patterns localized in different regions of the brain as patients perform simple activities such as tracking a small visual object. Figure 1.12, taken from [9], illustrates the result of applying a classification model to the problem of diagnosing patients with ADHD. This is discussed further in Example 11.19.

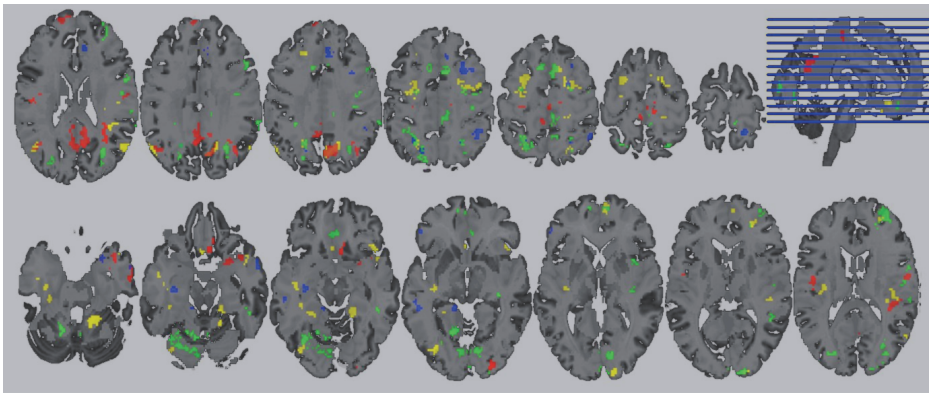

Figure 1.12 Figure associated with Example 1.7. See text for details.

Example 1.8 Spam detection

Spam detection is a standard text-based two-class classification problem. Implemented in most email systems, spam detection automatically identifies unwanted messages (e.g., advertisements), referred to as spam, from the emails users want to see (often referred to as ham). Once trained, a spam detector can remove unwanted messages without user input, greatly improving a user's email experience. This example is discussed in further detail in Examples 6.10 and 9.2.

Example 1.9 Financial applications

Two-class classification problems arise in all sorts of financial applications. They are often used in commercial lending to determine whether or not an individual should receive a commercial loan, credit card, etc., based on their historical financial information. This standard two-class classification problem, either "lend" or "do not lend," is explored in greater detail in Examples 6.11 and 9.7, with the latter example described in the context of *feature selection*.

Fraud detection is another hugely popular two-class classification problem in the financial space. The detection of fraudulent financial (e.g., credit card) transactions is naturally framed as a two-class classification problem where the two classes consist of legitimate transactions and fraudulent ones. The challenge with such problems is typically that the record of fraudulent transactions is

dwarfed many times over by valid ones, making such datasets highly *imbalanced*, an issue discussed further in Sections 6.8.4 and 6.9).

Example 1.10 Recognition problems

Recognition problems are a popular form of multi-class classification where the aim is to train a classifier to automatically distinguish between a collection of things, whether those things be human gestures (gesture recognition), various visual objects (object recognition), or spoken words (speech recognition).

For example, recognizing handwritten digits is a popular object recognition problem commonly built into the software of mobile banking applications, as well as more traditional Automated Teller Machines, to give users among others the ability to automatically deposit paper checks. In this application each class of data consists of (images of) several handwritten versions of a single digit in the range 0–9, resulting in a total of ten classes (see Figure 1.13).

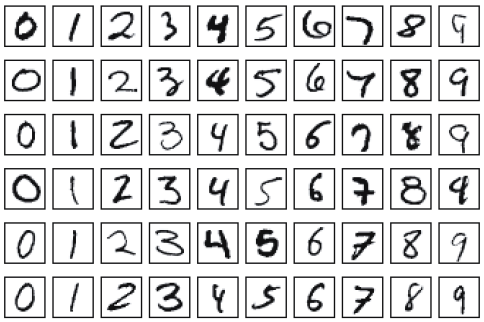

Figure 1.13 Figure associated with Example 1.10. An illustration of various handwritten digits. See text for further details.

We discuss handwritten digit recognition in further detail later at several different points (for instance, in Example 7.10), and more general applications in object and speech recognition later in Sections 9.2.4 and 9.2.3, respectively.

1.3.2 Unsupervised learning

Unsupervised learning (unlike supervised learning problems outlined previously) deals with the automatic learning of computational rules that describe *input* data only. Often such rules are learned in order to simplify a dataset to allow for easier supervised learning, or for human analysis and interpretation. Two fundamental unsupervised problems, *dimension reduction* and *clustering*, allow for simplification of a dataset via two natural paths: by either reducing the ambient dimension of input data (in the former case) or by determining a small number of representatives that adequately describe the diversity of a larger set of data (in the latter case). Both subcategories of unsupervised learning are first introduced in Chapter 8 (where the linear version of each is detailed),

and discussed further in Chapters 10–14 (where their nonlinear extensions are discussed).

Dimension reduction

The dimensionality of modern-day data such as images, videos, text documents, and genetic information, is often far too large for effective use of predictive modeling and analysis. For example, even a megapixel image, a medium-resolution image by today's standards, is a million-dimensional piece of data. This is true for a gray-scale image that has only one dimension for each of its one million pixels. A color megapixel image would have three million dimensions. Therefore, reducing the dimension of this sort of data is often crucial to effective application of many machine learning algorithms, making dimension reduction a common preprocessing step for prediction and analysis tasks.

Geometrically speaking, to reduce the dimension of a dataset means to squash or project it down onto a proper lower-dimensional line or curve (or more generally a linear hyperplane or nonlinear manifold), preferably one that retains as much of the original data's defining characteristics as possible.

This general idea is illustrated for two toy datasets in Figure 1.14, with the two-dimensional (left panel) and three-dimensional (right panel) data squashed (or projected) onto a proper one-dimensional line and two-dimensional hyperplane, respectively, reducing the ambient dimension of data by one in each case while retaining much of the shape of the original data. In practice, the reduction in dimension of modern-day large datasets can be much greater than achieved in this illustration.

Figure 1.14 Two toy datasets consisting of two-dimensional (left panel) and three-dimensional (right panel) input data, shown as hollow black circles. The data in each case is projected onto a lower-dimensional subspace, and is effectively lowered a dimension while retaining a good amount of the original data's structure.

Clustering

Clustering aims at identifying gross underlying structure in a set of input data by grouping together points that share some structural characteristic, e.g., proximity to one another in the feature space, which helps better organize or summarize

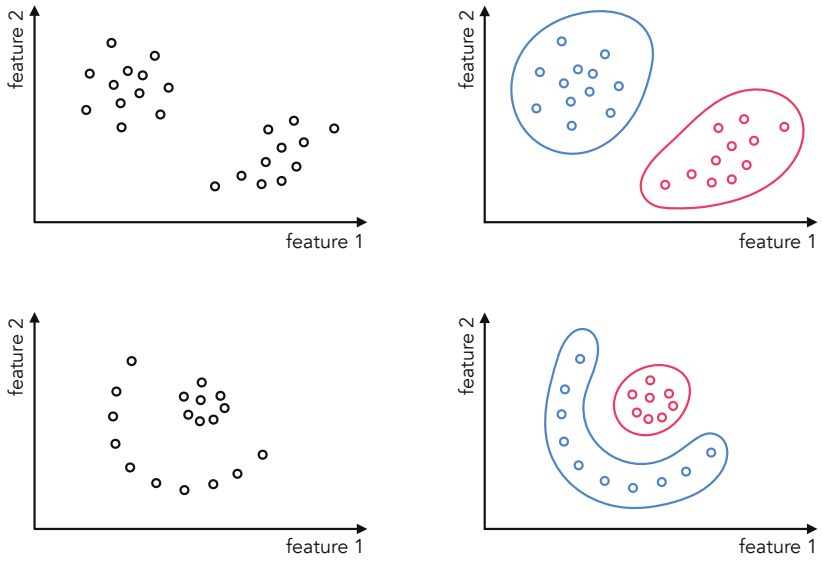

Figure 1.15 Two toy examples of the vast array in which input data can cluster. Clustering algorithms are designed to uncover these kinds of distinct structures. In each instance the distinct clusters in the original data (on the left) are colored for visualization purposes (on the right).

a set of training data for analysis by a human or machine interpreter. The structure of data can vary immensely, with data falling into globular clusters or along nonlinear manifolds as illustrated in Figure 1.15.

1.4 Mathematical Optimization

As we will see throughout the remainder of the book, we can formalize the search for parameters of a learning model via well-defined mathematical functions. These functions, commonly referred to as *cost* or *loss functions*, take in a specific set of model parameters and return a score indicating how well we would accomplish a given learning task using that choice of parameters. A high value indicates a choice of parameters that would give poor performance, while the opposite holds for a set of parameters providing a low value. For instance, recall the share price prediction example outlined in Figure 1.7 where we aimed at learning a regression line to predict a company's share price based on its revenue. This line is learned to the data by optimally tuning its two parameters: slope and vertical intercept. Geometrically, this corresponds to finding the set of parameters providing the smallest value (called a *minimum*) of a two-dimensional cost function, as shown pictorially in Figure 1.16. This concept plays a similarly fundamental role with classification (and indeed with all ma-

chine learning problems) as well. In Figure 1.10 we detailed how a general linear classifier is trained, with the ideal setting for its parameters again corresponding with the minimum of a cost function as illustrated pictorially in Figure 1.17.

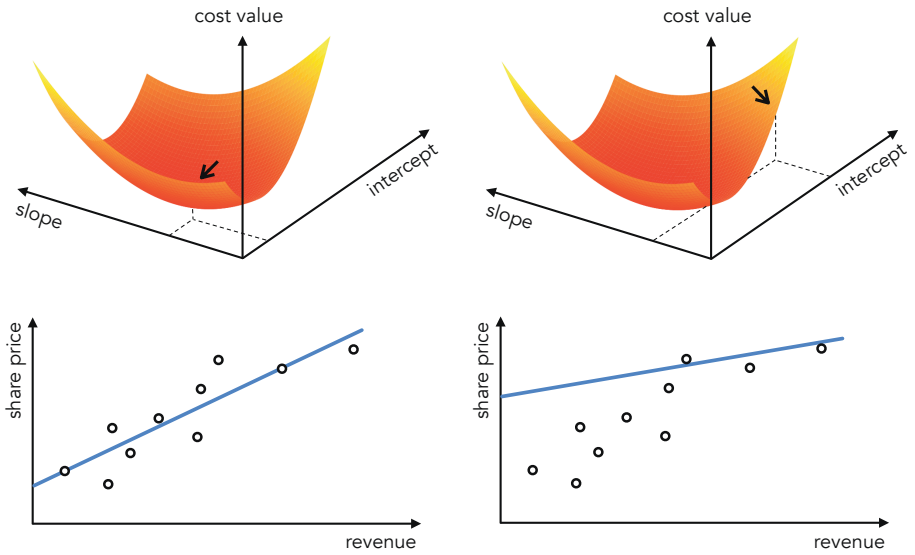

Figure 1.16 (top panels) A figurative drawing of the two-dimensional cost function associated with learning the slope and intercept parameters of a linear model for the share price regression problem discussed in the previous section and shown in Figure 1.7. Also shown here are two different sets of parameter values, one (left) at the minimum of the cost function and the other (right) at a point with larger cost function value. (bottom panels) The linear model corresponding to each set of parameters in the top panel. The set of parameters resulting in the best fit are found at the minimum of the cost surface.

Because a low value corresponds to a high-performing model in the case of both regression and classification (and, as we will see, for unsupervised learning problems as well) we will always look to *minimize* cost functions in order to find the ideal parameters of their associated learning models. As the study of computational methods for minimizing formal mathematical functions, the tools of *mathematical optimization* therefore play a fundamental role throughout the text. Additionally, as we will see later in the text starting in Chapter 11, optimization also plays a fundamental role in *cross-validation* or the learning of a proper *nonlinear* model automatically for any dataset. Because of these critical roles mathematical optimization plays in machine learning we begin this text with an exhaustive description of the fundamental tools of optimization in Chapters 2–4.

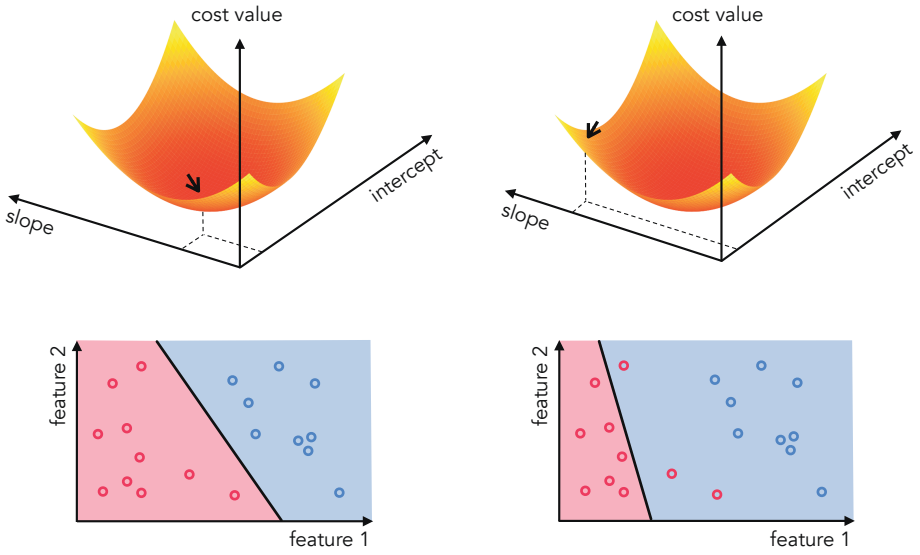

Figure 1.17 (top panels) A figurative drawing of the two-dimensional cost function associated with learning the slope and intercept parameters of a linear model separating two classes of the toy dataset first shown in Figure 1.10. Also shown here are two different sets of parameter values, one (left) corresponding to the minimum of the cost function and the other (right) corresponding to a point with larger cost function value. (bottom panels) The linear classifiers corresponding to each set of parameters in the top panels. The optimal set of parameters, i.e., those giving the minimum value of the associated cost function, allow for the best separation between the two classes.

1.5 Conclusion

In this chapter we have given a broad overview of machine learning, with an emphasis on critical concepts we will see repeatedly throughout the text. We began in Section 1.2 by describing a prototypical machine learning problem, as well as the steps typically taken to solve such a problem (summarized in Figure 1.6). In Section 1.3 we then introduced the fundamental families of machine learning problems – supervised and unsupervised learning – detailing a number of applications of both. Finally in Section 1.4 we motivated the need for mathematical optimization by the pursuit of ideal parameters for a machine learning model, which has direct correspondence to the geometric problem of finding the smallest value of an associated cost function (summarized pictorially in Figures 1.16 and 1.17).