1 Introduction

Machine learning is a rapidly growing field of study whose primary concern is the design and analysis of algorithms which enable computers to learn. While still a young discipline, with much more awaiting to be discovered than is currently known, today machine learning can be used to teach computers to perform a wide array of useful tasks. This includes tasks like the 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). In this chapter we give a high level introduction to the field of machine learning and the contents of this textbook. To get a big picture sense of how machine learning works we begin by discussing a simple toy machine learning problem: teaching a computer how to distinguish between pictures of cats from those with dogs. This will allow us to informally describe the procedures used to solve machine learning problems in general.

1.1 Teaching a computer to distinguish cats from dogs

To teach a child the difference between "cat" versus "dog", parents (almost!) never give their children some kind of formal scientific definition to distinguish the two; i.e., that a dog is a member of Canis Familiaris species from the broader class of Mammalia, and that a cat while being from the same class belongs to another species known as Felis Catus. No, instead the child is naturally presented with many images of what they are told are either "dogs" or "cats" 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. 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 of things, is referred to as a *classification* problem in machine learning.

1. Collecting data Like human beings, a computer must be trained to recognize the difference between these two types of animal 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

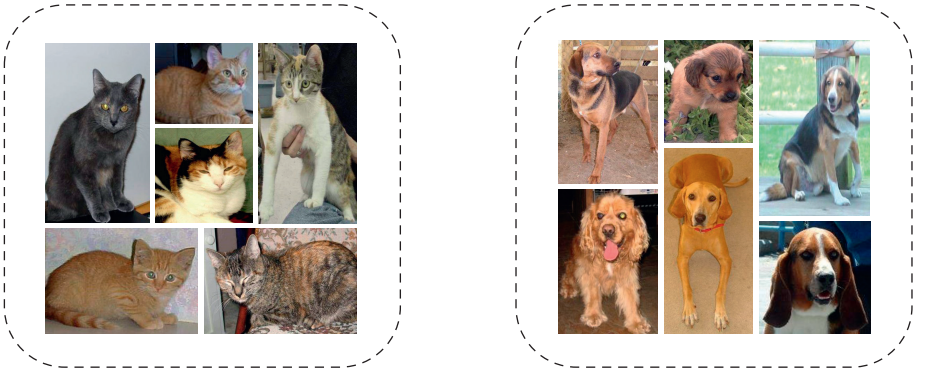

Fig. 1.1 A training set of six cats (left panel) and six dogs (right panel). This set is used to train a machine learning model that can distinguish between future images of cats and dogs. The images in this figure were taken from [31].

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.

2. Designing features Think for a moment about how you yourself tell the difference between images containing cats from those containing dogs. What do you look for in order to tell the two apart? You likely 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, you do not just look at an image as simply a collection of many small square pixels. You pick out details, or features, from images like these in order to identify what it is you 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 such features itself.

This is typically not a trivial task, as designing quality features can be very application dependent. For instance, a feature like "number of legs" would be unhelpful in discriminating between cats and dogs (since they both have four!), but quite helpful in telling cats and snakes 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's head, 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:

1. *size of nose*, relative to the size of the head (ranging from small to big);
2. *shape of ears* (ranging from round to pointy).

Examining the training images shown in Fig. 1.1, we can see that cats all have *small* noses and *pointy* ears, while dogs all have *big* noses and *round* ears. Notice that with the current choice of features each image can now be represented by just two numbers:

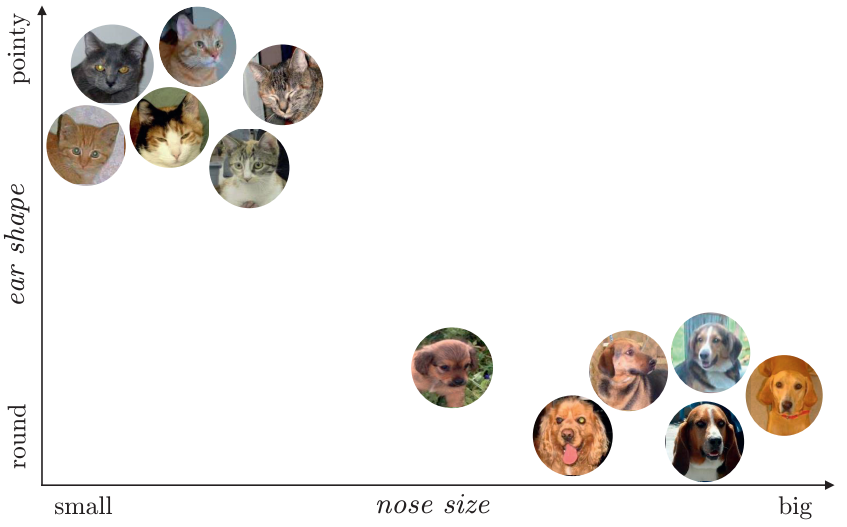

Fig. 1.2 Feature space representation of the training set where the horizontal and vertical axes represent the features "nose size" and "ear shape" respectively. The fact that the cats and dogs from our training set lie in distinct regions of the feature space reflects a good choice of features.

a number expressing the relative nose size, and another number capturing the pointyness or round-ness of ears. Therefore we now represent each image in our training set in a 2-dimensional *feature space* where the features "nose size" and "ear shape" are the horizontal and vertical coordinate axes respectively, as illustrated in Fig. 1.2. Because our designed features distinguish cats from dogs in our training set so well the feature representations of the cat images are all clumped together in one part of the space, while those of the dog images are clumped together in a different part of the space.

3. Training a model Now that we have a good feature representation of our training data the final act of teaching a computer how to distinguish between cats and dogs is a simple geometric problem: have the computer find a line or *linear model* that clearly separates the cats from the dogs in our carefully designed feature space.[1] Since a line (in a 2-dimensional space) has two parameters, a slope and an intercept, this means finding the right values for both. Because the parameters of this line must be determined based on the (feature representation) of the training data the process of determining proper parameters, which relies on a set of tools known as *numerical optimization,* 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. Once this line has been determined, any future image whose feature representation lies above it (in the blue region) will be considered a cat by the computer, and likewise any representation that falls below the line (in the red region) will be considered a dog.

[1] While generally speaking we could instead find a curve or *nonlinear model* that separates the data, we will see that linear models are by far the most common choice in practice when features are designed properly.

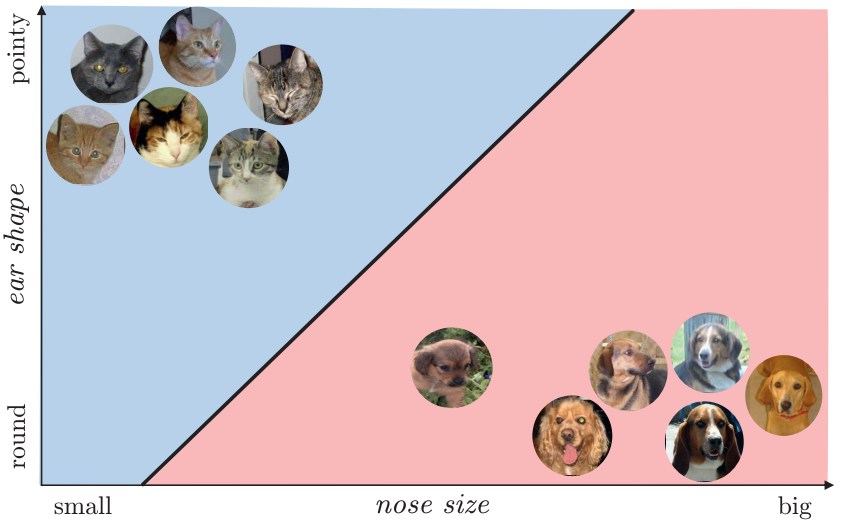

Fig. 1.3 A trained linear model (shown in black) perfectly separates the two classes of animal present in the training set. Any new image received in the future will be classified as a cat if its feature representation lies above this line (in the blue region), and a dog if the feature representation lies below this line (in the red region).

Fig. 1.4 A testing set of cat and dog images also taken from [31]. Note that one of the dogs, the Boston terrier on the top right, has both a short nose and pointy ears. Due to our chosen feature representation the computer will think this is a cat!

4. Testing the model To test the efficacy of our learner we now show the computer a batch of previously unseen images of cats and dogs (referred to generally as a *testing set* of data) and see how well it can identify the animal in each image. In Fig. 1.4 we show a sample testing 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 (nose size and ear shape), and simply check which side of our line the feature representation falls on. In this instance, as can be seen in Fig. 1.5 all of the new cats and all but one dog from the testing set have been identified correctly.

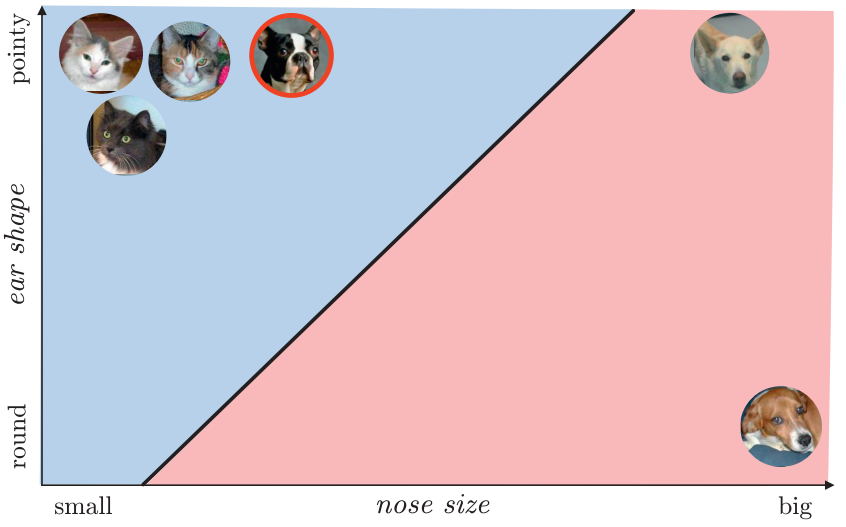

Fig. 1.5 Identification of (the feature representation of) our test images using our trained linear model. Notice that the Boston terrier is misclassified as a cat since it has pointy ears and a short nose, just like the cats in our training set.

The misidentification of the single dog (a Boston terrier) is due completely to our choice of features, which we designed based on the training set in Fig. 1.1. This dog has been misidentified simply because its features, a small nose and pointy ears, match those of the cats from our training set. So while it first appeared that a combination of nose size and ear shape could indeed distinguish cats from dogs, we now see that our training set was too small and not diverse enough for this choice of features to be completely effective.

To improve our learner we must begin again. First we should collect more data, forming a larger and more diverse training set. Then we will need to consider designing more discriminating features (perhaps eye color, tail shape, etc.) that further help distinguish cats from dogs. Finally we must train a new model using the designed features, and test it in the same manner to see if our new trained model is an improvement over the old one.

1.1.1 The pipeline of a typical machine learning problem

Let us now briefly review the previously described process, by which a trained model was created for the toy task of differentiating cats from dogs. The same process is used to perform essentially all machine learning tasks, and therefore it is worthwhile to pause for a moment and review the steps taken in solving typical machine learning problems. We enumerate these steps below to highlight their importance, which we refer to all together as the general pipeline for solving machine learning problems, and provide a picture that compactly summarizes the entire pipeline in Fig. 1.6.

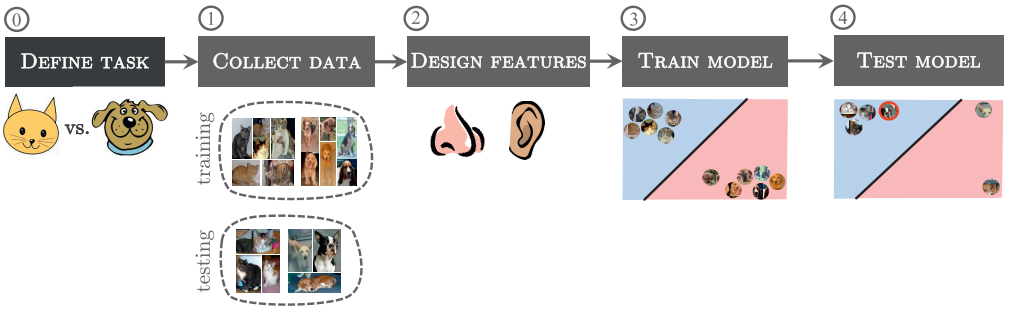

Fig. 1.6 The learning pipeline of the cat versus dog classification problem. The same general pipeline is used for essentially all machine learning problems.

(0) **Define the problem.** What is the task we want to teach a computer to do?

(1) **Collect data.** Gather data for training and testing sets. The larger and more diverse the data the better.

(2) **Design features.** What kind of features best describe the data?

(3) **Train the model.** Tune the parameters of an appropriate model on the training data using numerical optimization.

(4) **Test the model.** Evaluate the performance of the trained model on the testing data. If the results of this evaluation are poor, re-think the particular features used and gather more data if possible.

1.2 Predictive learning problems

Predictive learning problems constitute the majority of tasks machine learning can be used to solve today. Applicable to a wide array of situations and data types, in this section we introduce the two major predictive learning problems: *regression* and *classification*.

1.2.1 Regression

Suppose we wanted to predict the share price of a company that is about to go public (that is, when a company first starts offering its shares of stock to the public). Following the pipeline discussed in Section 1.1.1, 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

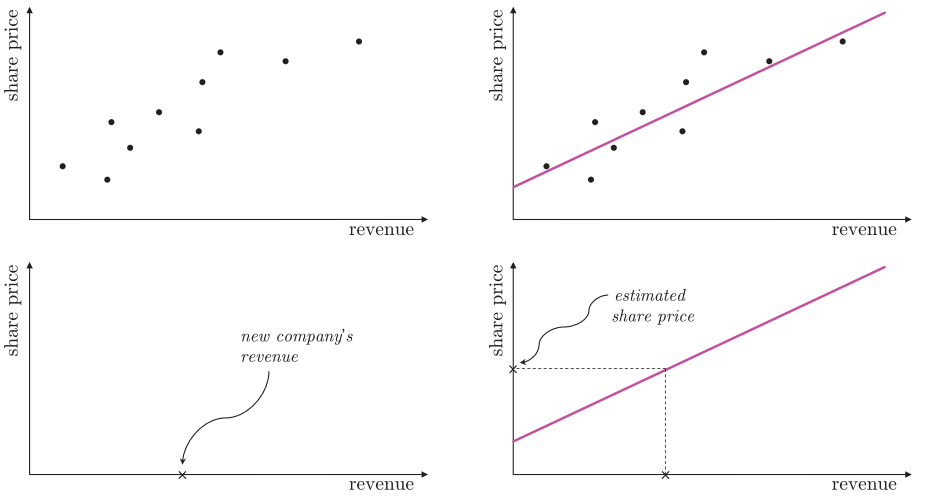

Fig. 1.7 (top left panel) A toy training dataset of ten corporations with their associated share price and revenue values. (top right panel) A linear model is fit to the data. This trend line models the overall trajectory of the points and can be used for prediction in the future as shown in the bottom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

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.[2] Now in order to connect the share price to the revenue we train a linear model or *regression line* using our training data.

The top panels of Fig. 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 testing set of data we can test the performance of our regression model and apply changes as needed (e.g., choosing a different feature). This sort of task, fitting a model to a set of training data so that predictions about a continuous-valued variable (e.g., share price) can be made, is referred to as *regression*. We now discuss some further examples of regression.

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

Figure 1.8 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 total student debt has tripled, totaling over one trillion dollars by the end of 2014. The regression line (in magenta) fit to this dataset represents the data quite well and, with its sharp positive slope, emphasizes the point that student debt is rising dangerously fast. Moreover, if

[2] Other potential features could include total assets, total equity, number of employees, years active, etc.

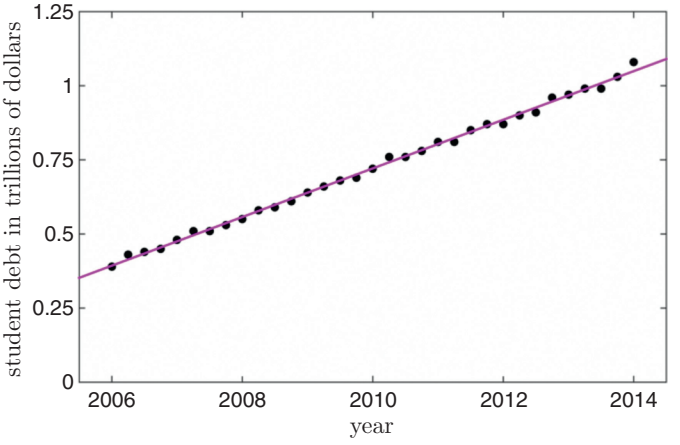

Fig. 1.8 Total student loan debt in the United States measured quarterly from 2006 to 2014. The rapid increase of the debt, measured by the slope of the trend line fit to the data, confirms the concerning claim that student debt is growing (dangerously) fast. The debt data shown in this figure was taken from [46].

this trend continues, we can use the regression line to predict that total student debt will reach a total of two trillion dollars by the year 2026.

Example 1.2 Revenue forecasting

In 1983, 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 to predict the success or failure of Hollywood movies. However, in the post-internet era of today, accurate estimation of box office revenue to be earned by upcoming movies is becoming possible. In particular, the quantity of internet searches for trailers, as well as the amount of discussion about a movie on social networks like Facebook and Twitter, have been shown to reliably predict a movie's opening weekend box office takings up to a month in advance (see e.g., [14, 62]). Sales forecasting for a range of products/services, including box office sales, is often performed using regression. Here the input feature can be for instance the volume of web searches for a movie trailer on a certain date, with the output being revenue made during the corresponding time period. Predicted revenue of a new movie can then be estimated using a regression model learned on such a dataset.

Example 1.3 Associating genes with quantitative traits

Genome-wide association (GWA) studies (Fig. 1.9) aim at understanding the connections between tens of thousands of genetic markers, taken from across the human genome of numerous subjects, with diseases like high blood pressure/cholesterol, heart

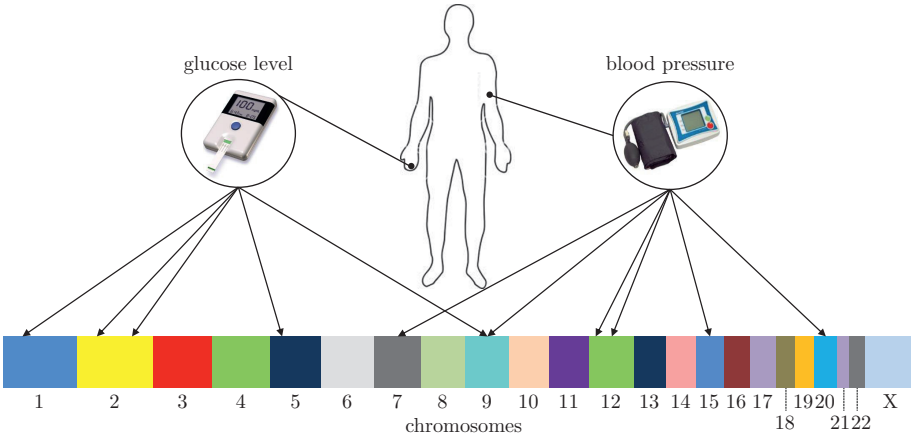

Fig. 1.9 Conceptual illustration of a GWA study employing regression, wherein a quantitative trait is to be associated with specific genomic locations.

disease, diabetes, various forms of cancer, and many others [26, 76, 80]. These studies are undertaken with the hope of one day producing gene-targeted therapies, like those used to treat diseases caused by a single gene (e.g., cystic fibrosis), that can help individuals with these multifactorial diseases. Regression as a commonly employed tool in GWA studies, is used to understand complex relationships between genetic markers (features) and quantitative traits like level of cholesterol or glucose (a continuous output variable).

1.2.2 Classification

The machine learning task of *classification* is similar in principle to that of regression. The key difference between the two is that instead of predicting a continuous-valued output (e.g., share price, blood pressure, etc.), with classification what 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 semi-autonomous and self-driving cars), is a very popular classification problem. The toy problem of distinguishing cats from dogs discussed in Section 1.1 was such a problem. 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 in e.g., controlling a computer without a mouse).

Geometrically speaking, a common way of viewing the task of classification is one of finding a separating line (or hyperplane in higher dimensions) that separates the two[3]

[3] Some classification problems (e.g., handwritten digit recognition) have naturally more than two classes for which we need a better model than a single line to separate the classes. We discuss in detail how multiclass classification is done later in Sections 4.4 and 6.3.

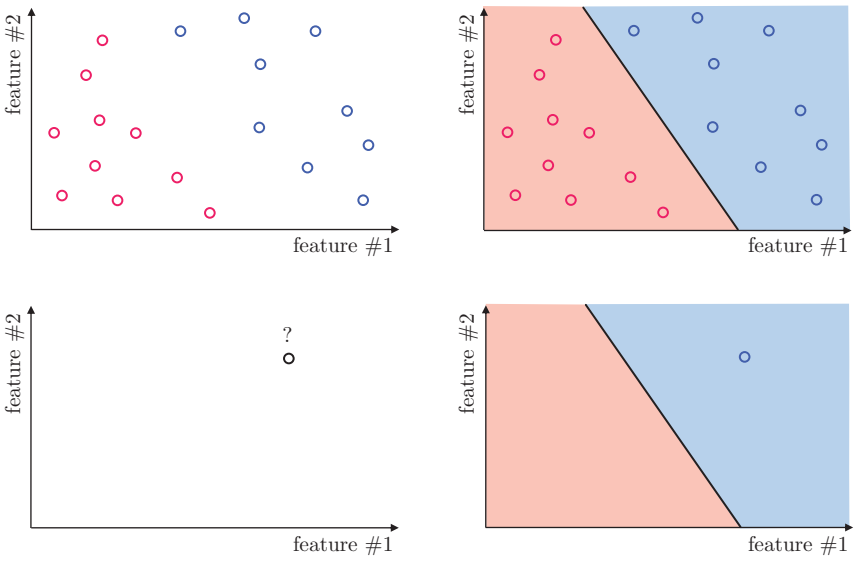

Fig. 1.10 (top left panel) A toy 2-dimensional training set consisting of two distinct classes, red and blue. (top right panel) A linear model is trained to separate the two classes. (bottom left panel) A test point whose class is unknown. (bottom right panel) The test point is classified as blue since it lies on the blue side of the trained linear classifier.

classes of data from a training set as best as possible. This is precisely the perspective on classification we took in describing the toy example in Section 1.1, where we used a line to separate (features extracted from) images of cats and dogs. New data from a testing set is then automatically classified by simply determining which side of the line/hyperplane the data lies on. Figure 1.10 illustrates the concept of a linear model or *classifier* used for performing classification on a 2-dimensional toy dataset.

Example 1.4 Object detection

Object detection, a common classification problem, 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,[4] 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 non-facial images, faces are sought after in a new test image by sliding a (typically) square

[4] While the problem of detecting pedestrians is a particularly well-studied classification problem [29, 32, 53], a standard semi-autonomous or self-driving car will employ a number of detectors that scan the vehicle's surroundings for other important objects as well, like road markings, signs, and other cars on the road.

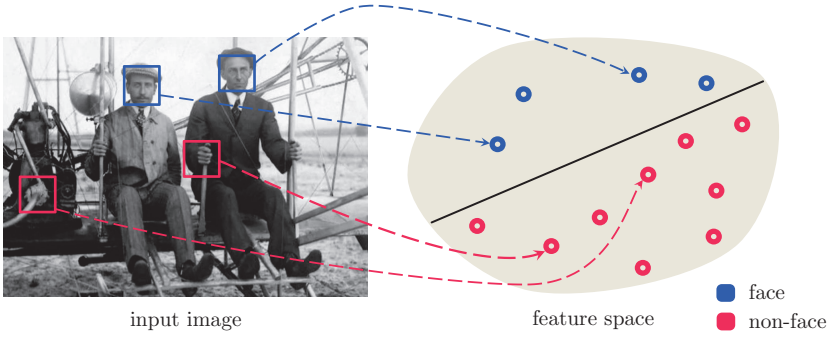

Fig. 1.11 To determine if any faces are present in a test image (in this instance an image of the Wright brothers, inventors of the airplane, sitting together in one of their first motorized flying machines in 1908) a small window is scanned across its entirety. The content inside the box at each instance 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 black on the right) are the "face" and "non-face" sides of the classifier, respectively.

window over the entire image. At each location of the sliding window the image content inside is tested to see which side of the classifier it lies on (as illustrated in Fig. 1.11). If the (feature representation of the) content lies on the "face side" of the classifier the content is classified as a face.[5]

Example 1.5 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. This has led many firms to seek out 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.

Example 1.6 Classification as a diagnostic tool in medicine

Cancer, in its many variations, remains among the most challenging diseases to diagnose and treat. Today it is believed that the culprit behind many types of cancers lies in accumulation of mutated genes, or in other words erroneous copies of an individual's

[5] In practice, to ensure that all faces at different distances from the camera are detected in a test image, typically windows of various sizes are used to scan as described here. If multiple detections are made around a single face they are then combined into a single highlighted window encasing the detected face.

DNA sequence. With the use of DNA microarray technology, geneticists are now able to simultaneously query expression levels of tens of thousands of genes from both healthy and tumorous tissues. This data can be used in a classification framework as a means of automatically identifying patients who have a genetic predisposition for contracting cancer. This problem is related to that of associating genes with quantitative biological traits, as discussed in Example 1.3.

Classification is also being increasingly used in the medical community to diagnose neurological disorders such as autism and attention deficit hyperactivity disorder (ADHD), using functional Magnetic Resonance Imaging (fMRI) of the human brain. These fMRI brain scans capture neural activity patterns localized in different regions of the brain over time as patients perform simple cognitive activities such as tracking a small visual object. The ultimate goal here is to train a diagnostic classification tool capable of distinguishing between patients who have a particular neurological disorder from those who do not, based solely on fMRI scans.

1.3 Feature design

As we have described in previous sections, *features* are those defining characteristics of a given dataset that allow for optimal learning. Indeed, well-designed features are absolutely crucial to the performance of both regression and classification schemes. However, broadly speaking the quality of features we can design is fundamentally dependent on our level of knowledge regarding the phenomenon we are studying. The more we understand (both intellectually and intuitively) the process generating the data we have at our fingertips, the better we can design features ourselves or, ideally, teach the computer to do this design work itself. At one extreme where we have near perfect understanding of the process generating our data, this knowledge having come from considerable intuitive, experimental, and mathematical reflection, the features we design allow near perfect performance. However, more often than not we know only a few facts, or perhaps none at all, about the data we are analyzing. The universe is an enormous and complicated place, and we have a solid understanding only of how a sliver of it works.

Below we give examples that highlight how our understanding of a phenomenon guides the design of features, from knowing quite a lot about a phenomenon to knowing just a few basic facts. A main thrust of the text will be to detail the current state of machine learning technology in dealing with this issue. One of the end goals of machine learning, which is still far from being solved adequately, is to develop effective tools for dealing with (finding patterns in) arbitrary kinds of data, a problem fundamentally having to do with finding good features.

Example 1.7 Galileo and uniform acceleration

In 1638 Galileo Galilei, infamous for his expulsion from the Catholic church for daring to claim that the earth orbited the sun and not the converse (as was the prevailing belief at

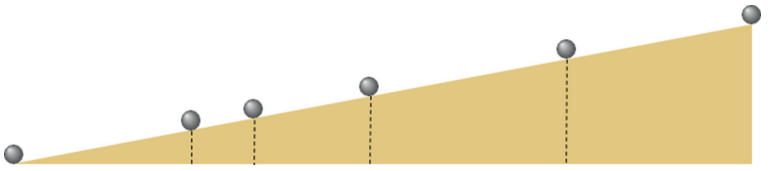

Fig. 1.12 Galileo's ramp experiment setup used for exploring the relationship between time and the distance an object falls due to gravity. To perform this experiment he repeatedly rolled a ball down a ramp and timed how long it took to get 1/4, 1/2, 2/3, 3/4, and all the way down the ramp.

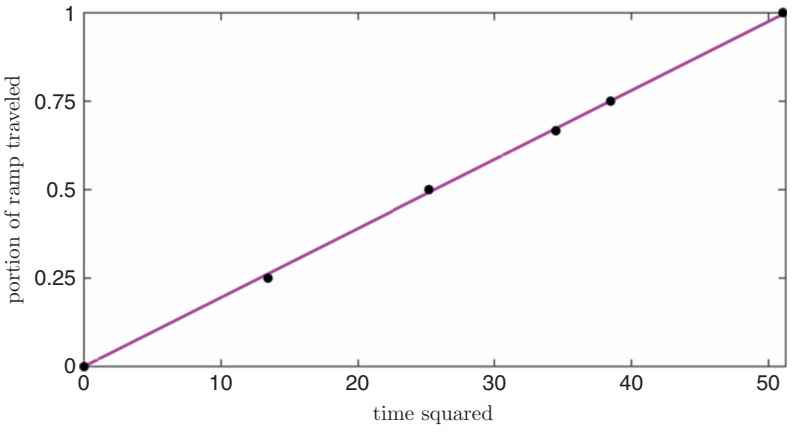

Fig. 1.13 Galileo's experimental data consisting of six points whose input is time and output is the fraction of the ramp traveled. Shown in the plot is the output with the feature time squared, along with a linear fit in magenta. In machine learning we call the variable "time squared" a feature of the original input variable "time."

the time) published his final book: *Dialogues Concerning Two New Sciences* [35]. In this book, written as a discourse among three men in the tradition of Aristotle, he described his experimental and philosophical evidence for the notion of uniformly accelerated physical motion. Specifically, Galileo (and others) had intuition that the acceleration of an object due to (the force we now know as) gravity is uniform in time, or in other words that the distance an object falls is directly proportional (i.e., linearly related) to the amount of time it has been traveling, squared. This relationship was empirically solidified using the following ingeniously simple experiment performed by Galileo.

Repeatedly rolling a metal ball down a grooved $5\,1/2$ meter long piece of wood set at an incline as shown in Fig. 1.12, Galileo timed how long the ball took to get 1/4, 1/2, 2/3, 3/4, and all the way down the wood ramp.[6]

Data from a modern reenactment [75] of these experiments (averaged over 30 trials), results in the six data points shown in Fig. 1.13. However, here we show not the original input (time) and output (corresponding fraction of the ramp traveled) data, but the output

[6] A ramp was used, as opposed to simply dropping the ball vertically, because time keeping devices in the time of Galileo were not precise enough to make accurate measurements if the ball was simply dropped.

paired with the feature *time squared*, measured in milliliters of water[7] as in Galileo's original experiment. By using square of the time as a feature the dataset becomes very much linearly related, allowing for a near perfect linear regression fit.

Example 1.8 Feature design for visual object detection

A more modern example of feature design, where we have only partial understanding of the underlying process generating data, lies in the task of visual object detection (first introduced in Example 1.4). Unlike the case with Galileo and uniform acceleration described previously, here we do not know nearly as much about the underlying process of visual cognition in both an experimental and philosophical sense. However, even with only pieces of a complete understanding we can still design useful features for object detection.

One of the most crucial and commonly leveraged facts in designing features for visual classification tasks (as we will see later in Section 4.6.2) is that the distinguishing information in a natural image, that is an image a human being would normally be exposed to like a forest or outdoor scene, cityscapes, other people, animals, the insides of buildings, etc., is largely contained in the relatively small number of edges in an image [15, 16]. For example Fig. 1.14 shows a natural image along with an image consisting of its most prominent edges. The majority of the pixels in this image do not belong to any edges, yet with just the edges we can still tell what the image contains.

Fig. 1.14 (left panel) A natural image, in this instance of the two creators/writers of the television show South Park (this image is reproduced with permission of Jason Marck). (right panel) The edge detected version of this image, where the bright yellow pixels indicate large edge content, still describes the scene very well (in the sense that we can still tell there are two people in the image) using only a fraction of the information contained in the original image. Note that edges have been colored yellow for visualization purposes only.

[7] Chronological watches (personal timepieces that keep track of hours/minutes/seconds like we have today) did not exist in the time of Galileo. Instead time was measured by calculating the amount of water dripped from a spout into a small cup while each ball rolled down the ramp. This clever time keeping device was known as a "water clock."

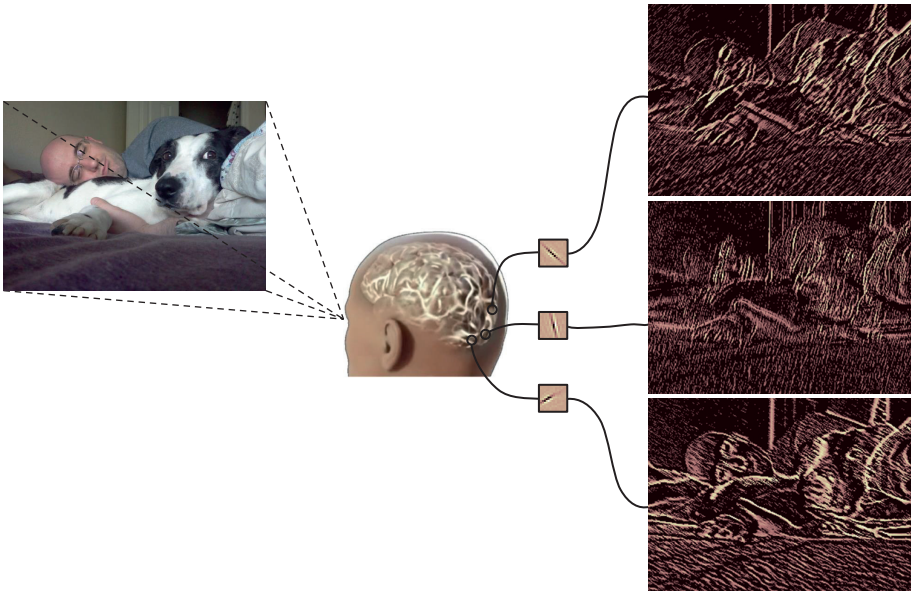

Fig. 1.15 Visual information is processed in an area of the brain where each neuron detects in the observed scene edges of a specific orientation and width. It is thought that what we (and other mammals) "see" is a processed interpolation of these edge detected images.

From visual studies performed largely on frogs, cats, and primates, where a subject is shown visual stimuli while electrical impulses are recorded in a small area in the subject's brain where visual information is processed, neuroscientists have determined that individual neurons involved roughly operate by identifying edges [41, 55]. Each neuron therefore acts as a small "edge detector," locating edges in an image of a specific orientation and thickness, as shown in Fig. 1.15. It is thought that by combining and processing these edge detected images, humans and other mammals "see."

1.4 Numerical 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 function*s, 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 from Section 1.2.1, in which we aimed at learning a regression line to predict a company's share price based on its revenue. This line is fit properly by optimally tuning its two parameters: slope and intercept. Geometrically, this corresponds to finding the set of parameters providing the smallest value (called a *minimum*) of a 2-dimensional cost function, as shown in Fig. 1.16.

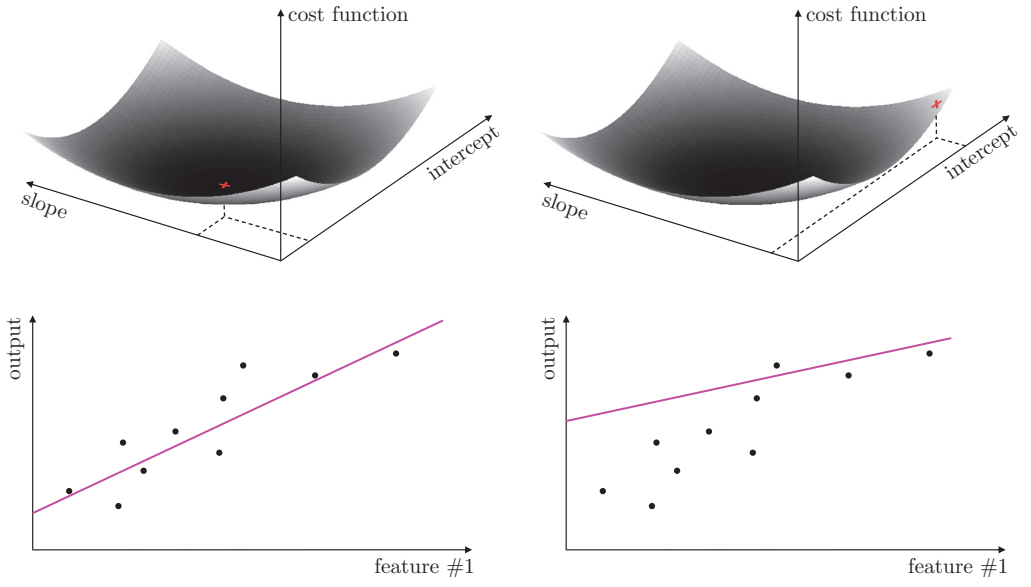

Fig. 1.16 (top panels) The 2-dimensional cost function associated with learning the slope and intercept parameters of a linear model for share price prediction based on revenue for a toy dataset first shown in Fig. 1.7. Also shown here are two different sets of parameter values, one (left) at the minimum 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 performance are found at the minimum of the cost surface.

This concept plays a similarly fundamental role with classification as well. Recall the toy problem of distinguishing between images of cats and dogs, described in Section 1.1. There we discussed how a linear classifier is trained on the feature representations of a training set by finding ideal values for its two parameters, which are again slope and intercept. The ideal setting for these parameters again corresponds with the minimum of a cost function, as illustrated in Fig. 1.17.

Because a low value corresponds to a high performing model in the case of both regression and classification, 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 *numerical optimization* therefore play a fundamental role throughout the text.

1.5 Summary

In this chapter we have given a broad overview of machine learning, with an emphasis on critical concepts we will see repeatedly throughout the book. We began in Section 1.1 where we described a prototypical machine learning problem, as well as the steps typically taken to solve such a problem (summarized in Fig. 1.6). In Section 1.2 we then introduced the two fundamental problems of machine learning: *regression*

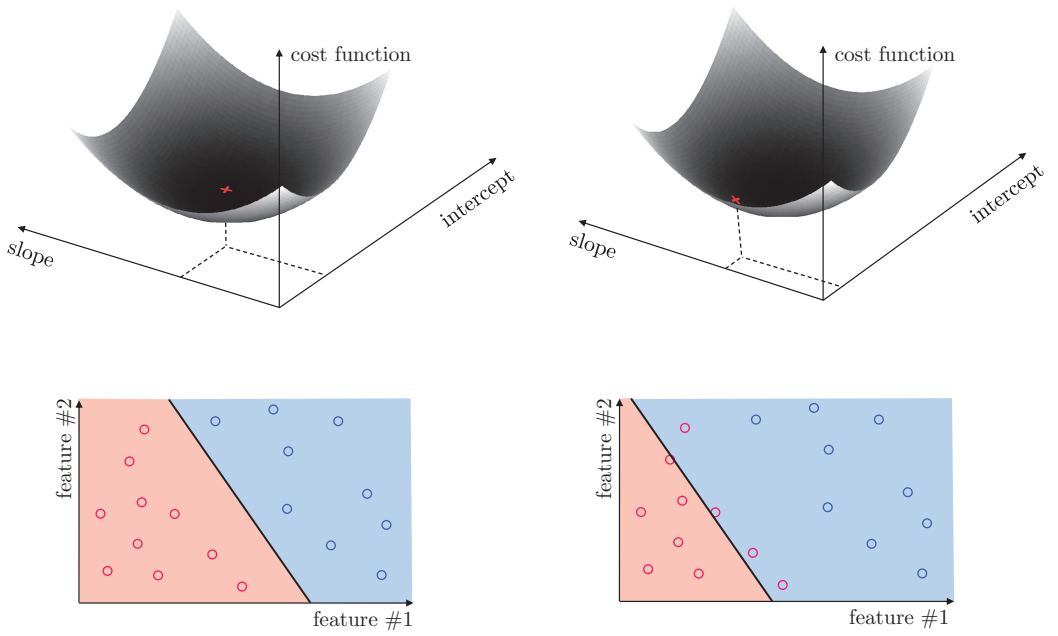

Fig. 1.17 (top panels) The 2-dimensional cost function associated with learning the slope and intercept parameters of a linear model separating two classes of data. Also shown here are two different sets of parameter values, one (left) corresponding to the minimum of the cost function and the other (right) 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 performance.

and *classification*, detailing a number of applications of both. Next, Section 1.3 introduced the notion of *features*, or those uniquely descriptive elements of data that are crucial to effective learning. Finally in Section 1.4 we motivated the need for *numerical optimization* due to the pursuit of ideal parameters of a learning model having direct correspondence to the geometric problem of finding the smallest value of an associated cost function (summarized pictorially in Fig. 1.16 and 1.17).