Contents

CHAPTER ONE

Introduction to Machine Learning Methods

1

I have a great supervised machine learning joke. . .
But you need to have heard a similar one before.

1.1 WHAT IS MACHINE LEARNING?

The first question I am always asked when I see distant relatives at holiday gatherings is: What are you working on? And if for many years, "Astrophysics" was a popular yet often misleading answer (as I had to explain that I was not training to be an astronaut, and in fact, sadly, I would probably never even discover a new world), now that I've thrown machine learning into the mix, my answers have become even more murky and vague. However, I think it's important that we can explain without any jargon what we do, and as a consequence, I've spent many hours thinking about how to describe machine learning.

To the best of my knowledge/ability to explain, I would say that it's *the process of teaching a machine to make informed, data-driven decisions*. Examples of such decisions include recognizing and characterizing objects based on similarities or differences, detecting patterns, and distinguishing signal from noise.

In many ways, the boundaries and definition of machine learning as a discipline are fluid, and so far, it's often been approached without a scientific spirit of inquiry. But it is my opinion that this process should be subject to the same level of rigor and testing that any scientific investigation needs to endure. In this book, we will explore many ways to build, test, understand, and break machine learning models.

1.2 WHAT CAN WE DO WITH IT?

Once we've established a working definition of machine learning, a far more interesting question to ask is: What can we do with machine learning tools? And this is where the conversation can become really long. I won't strive for completeness here,

Figure 1.1: A cartoon example showing some types of problems that machine learning can help solve.

and there are many real-life applications that involve machine learning methods but are so complicated that they can't quite fit in a mold, for example, self-driving cars or AI (artificial intelligence) systems playing video games. But when reduced to building blocks of larger operations, I would say that often machine learning tools are used to do one of the following four things (see Figure 1.1):

- **Recognize** an instance of a certain type. For example, this could mean correctly labeling different type of animals, like cats or dogs, from images, or recognizing a specific person in a picture, as is done in social media "tagging" of people.
- **Predict** some property or information on the basis of some other information. For example, we may attempt to predict future behavior based on past behavior; use previous utility bills to predict the next one, or reconstruct a missing part of an image based on other examples of similar images.
- **Group together** objects that are similar, which can also be used to single out objects that are different ("outliers" in scientific parlance). For example, we might try figuring out how many different types of galaxies there are in a picture of the sky, or which strange-looking astrophysical sources are more likely to be artifacts of the camera. This type of application is referred to as *clustering*.
- **Simplify** the information contained in a complicated data set to condense it to its quintessential nature. For example, we could ask what's the simplest way to draw a cat so that it's still recognizable as a cat. This process has the double advantage of reducing the volume of data we need to deal with (which is desirable for manipulation, storage, and visualization purposes) and helping us understand what the essential properties are of a given category (in our case, cats). These are often called *dimensionality reduction* techniques by the initiated.

1.3 THE LANGUAGE OF MACHINE LEARNING

Unlike in the Fight Club,[1] the first rule of machine learning is to learn how to talk about it: Laying down very clearly what it is that we know and what it is that we want to know is incredibly important.

To begin with, the elements of a data set are often called *instances*; other commonly used names are "samples" or "examples." In the physical sciences, it is not uncommon to hear "observations" in reference to instances, but we will try to stay away from this confusing habit. For each one of these instances, there will be some properties that are known; these are usually things that can be measured, observed through experiment, or simulated. They are called *features* (or less technically, "input"). For example, if you are studying Newton's law of gravity, you might measure the distance and time of falling objects, and you might ask yourself about the final speed. In this case, the features of your problem are two: distance and time. If you are studying galaxies and you are observing them in different regions of the electromagnetic spectrum (say, for example, ultraviolet, visible, infrared, and radio), and you have one data point for each of those measurements, those will be your four features. Note that the features don't need to be numerical! They can also be of *categorical* type, like yes/no, or 0/1, or low/medium/high, or even have a descriptive quality like red/blue/green. Usually, it is necessary to map such types to numerical values before plugging them into ML machinery.

It is customary to organize a data set in rows and columns, where each row describes one instance, and each column contains the measured value of one feature associated with that instance. Therefore, usually the total size of your data set will be given by the number of instances times the number of features.

At times, there will be one or more additional pieces of information (properties; in the example above, the final speed) that you would like to estimate or predict, given the value of the features. This property is called *target*, or sometimes *label*, or more generally, output. In other cases, the output might not be a specific quantity, but rather a pattern, or a rule.

A summary plot of this basic terminology is shown in Figure 1.2.

Usually, the goal of an ML task is to build a relationship between input and output, which will then be described by our *machine learning model*. A model is a mathematical object that allows us to go from the input space of features to the output space of targets. Related terms include the words "method" and (more commonly) "algorithm," which usually refer to particular classes of mathematical relationships that can be used to build models. Examples of ML algorithms that you might have heard of include Random Forests and neural networks. They exist independently of our input and output, and once they are used for a specific problem, they define some rules on how we can build models.

1 https://en.wikipedia.org/wiki/Fight_Club

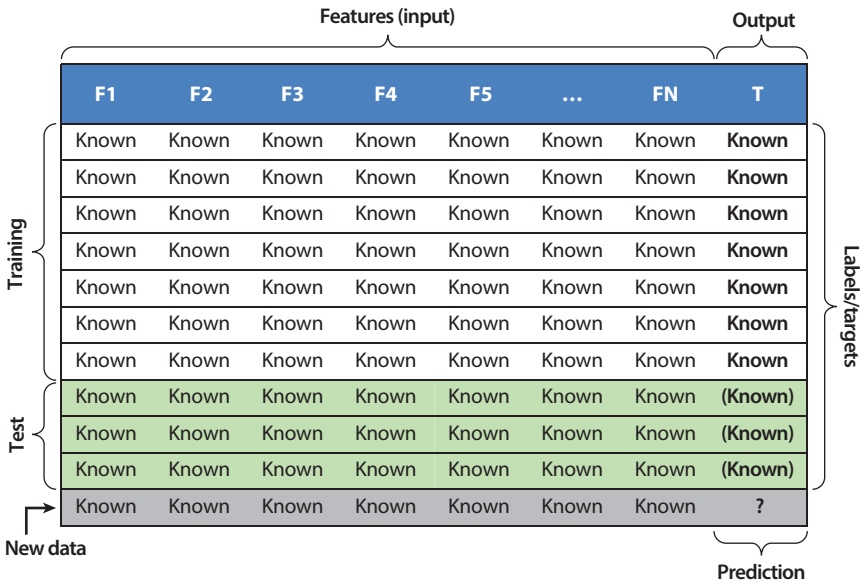

Figure 1.2: A visual summary of some terminology frequently used in machine learning.

One important distinction among machine learning methods is between *supervised* and *unsupervised* methods. The boundary between them is sometimes blurry, and their union doesn't cover the whole range of possibilities; some practitioners prefer to use different terminology or to do away with these denominations. Nonetheless, understanding the distinction between the two methods is, in my opinion, important, and we will discuss it below.

1.4 SUPERVISED LEARNING

In a supervised learning task, we assume that there is a collection of instances for which the target property is known (besides the features, which are assumed to be known for all data). This collection is called the *learning* set. For example, let's assume that each of the data points in the left panel of Figure 1.3 has a color associated with it: blue or green. We would like to learn to predict the color, based on each point's coordinates; gray indicates that the color is not yet known. This data set has two features (the x and y coordinates). Supervised learning consists of *learning by example:* we need to be shown some instances for which the color is known, in order to develop an intuition—in ML parlance, to build a model—of the relationship between coordinate and color. The learning set for this problem is shown in the right panel of the figure, and it contains 30 instances (the number of colored points).

To solve this problem, an ML algorithm will attempt to use the instances in the learning set to infer the rule that connects the coordinates to the color. If it is successful, when presented with another point for which only the features

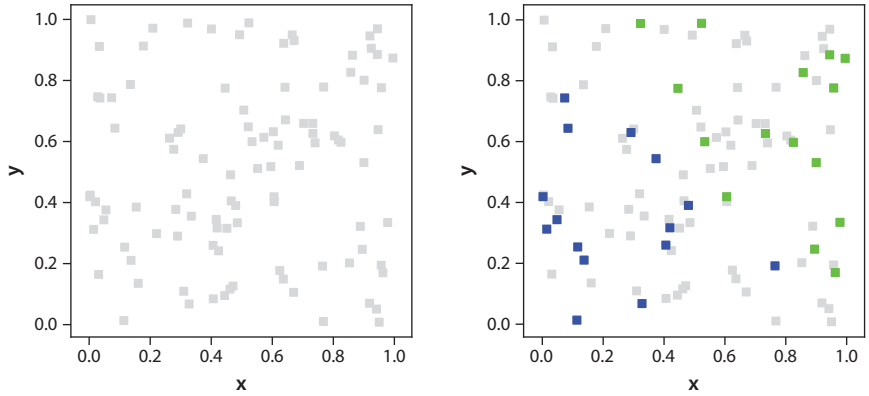

Figure 1.3: A simple example of a supervised problem. We would like to learn to predict the color of a point, given its coordinates (points for which the color is unknown are gray). To do so, we are provided with a *learning set* (right): a subset of instances for which both the features (coordinates) and the target property (color) are known.

(coordinates) are known (e.g., any of the gray-colored points in the figure), the algorithm will be able to make a correct prediction of the target property (color).

As a general rule, a *supervised learning method is only as good as its learning set* or at least it starts out like that. A training set that contains too few examples would not allow an ML model to learn the relationship correctly. Let's look at this simple example:

input	output
1	3
2	3
3	?

Here I am providing two examples in the learning set, and I am asking you to predict the output of a third example. Most likely, you would predict "3" as the output, but alas, this is not the correct answer. We can try to improve by adding a few more examples to the learning set:

input	output
1	3
2	3
3	5
4	4
5	?

Now, it is possible that some of you have already figured out what the rule is. But for the vast majority of us who still wouldn't know, let me rewrite the information above in a slightly more helpful form:

input	output
one	3
two	3
three	5
four	4
five	?

I won't spoil the fun right here, but you can check your intuition (or give up and move on) by reading the note at the bottom of the page.[2] Hopefully, this toy example serves to show some important properties of a good learning set. The first is that it needs to be large enough so that our algorithms can figure out what the rule is. The second is that the choice of features matters; some representations of the data work better than others, even if they are based on the same information. In our simple case, deciding to spell out the numbers instead of using their mathematical representation served to emphasize which aspect of the data was important. This is relevant, because it is usually the job of the scientist to decide how to organize the information when building a data set.

1.4.1 Train and test sets

One important consideration in supervised learning is that it would not be wise to use all the objects in the learning set to infer the relationship between features and target (again in ML parlance, to train the model). This is best understood, in my opinion, by thinking about the scientific method itself. The process of training the model is akin to formulating a hypothesis. The next step then is to make predictions that result from that hypothesis, and to test them to verify whether they are correct. It should be clear that we cannot verify the predictions of the model on the same instances that were used in the training process, because our verification process needs to be independent of the training process. Another way to think about this issue is that we are interested in assessing how well our model can predict the target property of *new* data; objects that participated in model building are not new to the model.

Therefore, it is customary to set aside a subset of the learning set that does not participate in the model building. Once the training process is complete, we can use the model to make predictions for the target property of those objects and verify that they are correct. Or more generally, we can check how the model performs on that subset and decide whether we are satisfied or require further improvement. The subdivisions of the learning set used for training and testing a model are called—you guessed it—training (or train) set and test set, respectively. The idea is illustrated in

2 The model should return the number of letters in the input.

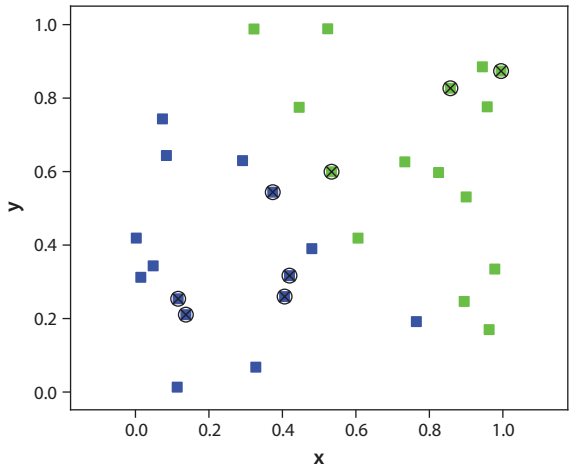

Figure 1.4: The learning set (the set of objects for which the labels are known) from the right panel of Figure 1.3 is split in a training set (square points) and test set (crossed-out points). The test set does not participate in the training process. The performance of the algorithm on new data can be estimated by applying the trained model to the test set features to generate predicted labels (colors) and comparing them to the true ones.

Figure 1.4. In the next few chapters, we will discuss at length what the best way is to split a data set into training and test sets.

The performance of a model can be expressed as rate of failure (error) or rate of success (score); these are just two equivalent ways of reporting how well a model works. We will often distinguish between the performance of a model when applied to the training set (the *training error* or *training score*) and the performance of a model when applied to the test set (the *test error* or *test score*). Finally, the performance of a model on *new* data, which were not part of our learning set, is called the *generalization error* (or *generalization score*) of the model. We don't have a way of calculating the generalization error directly, because the ground truth labels are not known. Therefore, we use the test error as a proxy for the generalization error.

Let's continue our preliminary investigation by looking at one more simple example. Imagine that we want to apply an ML method to the population of points from Figure 1.3. What would happen if our learning set was only made up of the blue objects, as in Figure 1.5? No matter how we decide to split in train and test sets, our model would learn that any combination of coordinates leads to predicting the color as blue, because it has never been shown that green points exist. And this is not even the worst problem. The problem is that if we then proceeded, like good ML practitioners, to verify our predictions on the test set, our constant prediction of "blue" would be correct 100% of the time. So we would get a false sense of confidence that our model is reliable, when instead it would fail miserably on the vast majority of the other data points in Figure 1.5, which (as we know from the previous sections) are mostly green.

This simple example illustrates one of the cruxes of machine learning techniques: Because they are driven by the data as opposed to relying on physical intuition, we are bound to make a fool of ourselves if we don't understand the data

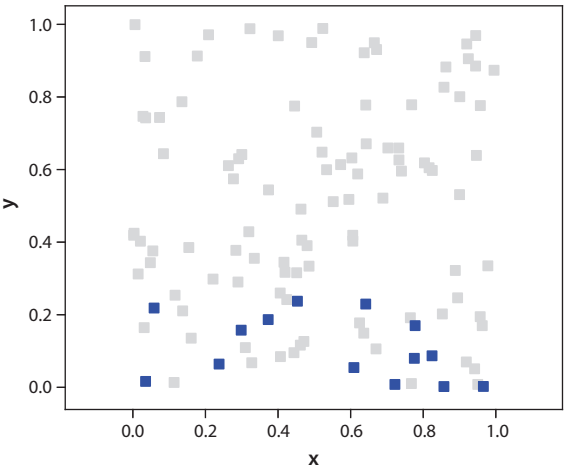

Figure 1.5: The subsample of blue points is a dangerous choice of learning set, if our goal is to learn how to predict color, based on coordinates, for all the points in this diagram (same population as Figure 1.3).

well. How can we save ourselves from this sorry destiny? In the case considered here, we should have noticed right away that the feature space (i.e., the range of x and y coordinates) spanned by our learning set is very different from the feature space spanned by the population we want to apply our model to (i.e., the gray points in Figure 1.5). This should have made us suspicious, because in general a learning set should be representative of the application domain, meaning that the learning set and application set should be statistically similar. If they are not, then we can hold no hope that the test error will be a good proxy for the generalization error. This is an important minimum requirement to add to the desired characteristics of learning sets: We'd like them to be large enough, representative of our application domain, and ideally organized in a way that optimizes the information content for the task at hand.

It's not always easy to make sure that these conditions hold, as we will see as we continue our ML journey.

1.4.2 Classification vs Regression

Another important distinction (although possibly more in terms of nomenclature than methods) in the realm of supervised problems is between classification and regression tasks. They only differ in the "output" part: in *classification*, the target property belongs to a discrete set of possibilities (in other words, a class). A simple toy example could be to correctly identify fruit in a bag on the basis of properties that can be measured through touch, such as height, width, weight, and shape. For something more relevant to daily life, recognizing people in pictures (the "tagging" of social media) is an example of a classification problem: The output can only be a specific person, and there is no notion of adjacency or continuity between outputs. As a result, classification algorithms will output a response that is either 100%

correct or 100% wrong. Either they recognize you (yay!), or they will mistake you for someone else (nay). When we evaluate the performance of a classifier, we will "count" (in more or less smart ways) the number of correct answers.

The other option is *regression*. In regression tasks, the output is a continuous variable (typically, a real number). For example, if we were trying to emulate Galileo and learn to predict the timings of different falling objects (without knowing the formula!) on the basis of distance traveled, our output would be time, probably in seconds. The output can take any value; in fact, the number of significant digits is only dictated by the precision of our measurements. As a consequence, in regression tasks, we cannot evaluate our model by asking for an *exact* answer; instead, we will assess how close we are to the correct value. Unlike in classification problems, where we are either correct or wrong, if the correct answer is 3.1415, predicting 3.0 and 10.0 are very different results.

It is important to note that the distinction between classification and regression problems can be quite fuzzy. For example, many problems in which the output is a decision ("Will it rain later today?" or "Will I be approved for a loan?") rely, both from a conceptual and a mathematical perspective, on the idea of a threshold. Even if the final output is discrete, the features will be implicitly mapped to a probability, which is a continuous variable, and then a threshold will be chosen to separate the classes (e.g., if the probability of rain is assessed to be more than 50%, the answer will be "yes"). Therefore, while these types of problems technically meet the definition of classification, they can be easily recast or thought of as regression problems. In fact, casting them as such might be advantageous, because outputting a probability retains more information about how confident we are in our prediction. Our model might indicate "no rain later" for two different sets of conditions, but if all weather models indicate a high pressure front, there is not a single cloud in the sky, and you are in the desert, the prediction for no rain will be much more solid than in a scenario of high humidity, low pressure, and a cloudy sky.

However, there are some classification problems that are truly "discretized" in nature. For example, one of the classic data sets for machine learning applications is the digits data set [Le Cun et al., 1989], where the machine is tasked with "reading" images of hand-written digits. The output is one of ten classes, the numbers from 0 to 9. In this case, the classes are truly separate from one another: 0 is not more similar to 1 than it is to 8 or 9. Another, more modern, example is the CIFAR-10 data set [Krizhevsky and Hinton, 2009], which is an image recognition problem with ten possible outputs, including birds, cars, and airplanes, as seen in Figure 1.6. Again, there is no obvious mapping of the distance between classes. This lack of contiguity among classes is what characterizes pure classification problems.

Finally, let me state again that the difference between classification and regression tasks is unrelated to the features of the problem and is only determined by the

Figure 1.6: The CIFAR-10 data set is composed of tiny, and therefore blurry, images belonging to 10 distinct classes (for example, cats, automobiles, and birds). Picture from the PyTorch [Paszke et al., 2019] website. Copyright ©2013 Valay Shah

target property, or output. Somehow, this always seems to be a tricky point for my students, so I just state it again here for emphasis.

1.5 UNSUPERVISED LEARNING

In unsupervised learning tasks, there are no labeled examples. Rather than predicting a specific quantity or property from the features, we are trying to discover patterns in the data. In a way, the target of an unsupervised learning algorithm is a pattern, as opposed to an unknown property.

1.5.1 Clustering

A vast subset of unsupervised learning tasks has to do with counting or grouping objects in a smart way; these applications are known as *clustering*. For example, imagine that you want to count how many friends appear in your childhood (or Facebook) photos. A successful clustering algorithm would be able to group

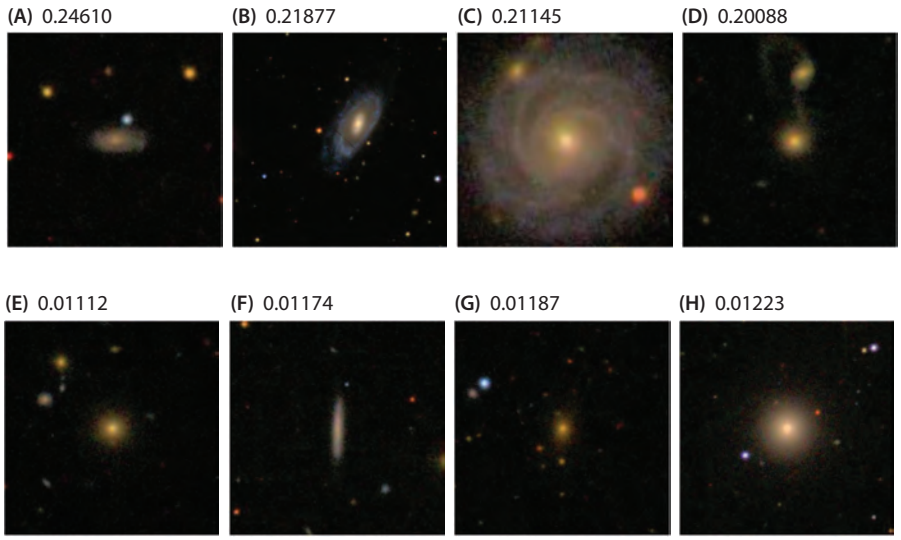

Figure 1.7: A sample set of galaxies observed by the Sloan Digital Sky Surveys (SDSS), showing different types of galaxy morphologies. Clustering could be used to determine from data how many fundamental morphological types exist. Reproduced with permission from [Dieleman et al., 2015].

together all instances of the same person, which would become a cluster; you could then count the number of occurrences. This happens without needing to know anything about the identity of the individuals beforehand; in other words, without the supervision process. Another application that we all, sadly, have become familiar with has been visualizing outbreaks of a disease; most recently, COVID-19. The outbreaks are usually represented with noncontiguous circles on a map. The clustering process can be used to determine the location (centers) and size (radii) of those circles that best represent the data.

Unsupervised learning methods often can be used to check a scientist's intuition. One historic example is the Hubble classification of galaxies. Edwin Hubble was one of the pioneers of extragalactic astronomy, and he proposed to divide galaxies into the main categories of elliptical, spiral, and irregular galaxies, according to their morphology (shape), as illustrated in Figure 1.7. There were further subdivisions, but let's ignore them for the time being. His reasoning was motivated by what he (largely incorrectly) believed to be the evolutionary track of galaxies. Nowadays, the task of automated morphology classification has become quite crucial, because we have observed many millions of galaxies, and upcoming instruments, such as the Vera Rubin Observatory or the James Webb Space Telescope are set to observe many more; with more than 200 billion of them in the Universe, we won't be done any time soon. This problem can be solved in a supervised manner, by deciding which classes to use beforehand (e.g., see [Dieleman et al., 2015]); in the case of Hubble's original proposal, they would be elliptical/spiral/irregular. Humans then

would need to build a learning set by providing visual classifications for an appropriate number and range of galaxies. However, it might be more advantageous to cast this problem as an unsupervised task, as in [Hocking et al., 2018]. In this case, our ML model will decide for itself which classes to pick by selecting its own criterion for forming clusters, which will determine which objects are assigned to each cluster, as well as (possibly) how many clusters will be found. This process creates a completely data-driven classification scheme, which might differ significantly from the one proposed in a supervised scheme and could reveal something new and important about galaxy formation and evolution. For example, if we found a "new" type of galaxy that is different enough from others to warrant its own cluster, this discovery might point to a different evolutionary pattern for those galaxies. Another advantage of the unsupervised approach is that no learning set is needed. However, the classification (or regression) schemes provided by unsupervised models are often harder to interpret, as it might not be easy to understand what classification criterion has been chosen (i.e., what the common properties of objects in the same cluster are), so human intervention might still be required. Sometimes, these two-step approaches are referred to as *semi-supervised learning*.

1.5.2 Dimensionality reduction

Another broad subcategory of unsupervised learning tasks refers to "simplification" processes, known as *dimensionality reduction* (DR). These techniques are typically used to make a data set smaller, and therefore more manageable and easier to visualize, with minimal loss of information. Their eventual success relies on the amount of redundancy present in the original data set and on the efficiency of the chosen compression technique.

DR methods tend to work in one of three ways. The simplest approach consists of retaining the original features and just selecting those that are expected to be more meaningful. The second approach is to remap the feature space to a different one, in which fewer components can express the highest amount of variance of the original data. Finally, a third approach consists of learning a *manifold*, a nonlinear space in which some interesting properties of the original data set (e.g., pairwise distances or dot products between elements) are preserved. See [Sorzano et al., 2014] for a review. Often, DR techniques can be used as a preprocessing step for clustering algorithms.

A simple graphical representation of clustering and dimensionality reduction is shown in Figure 1.8.

The most fun, useful, and creative applications of machine learning happen when we learn to mix and match all the techniques. For example, we can decide to learn (unsupervised) feature representation as a way of preprocessing a labeled data set, obtaining a more efficient and helpful representation of a learning set, and

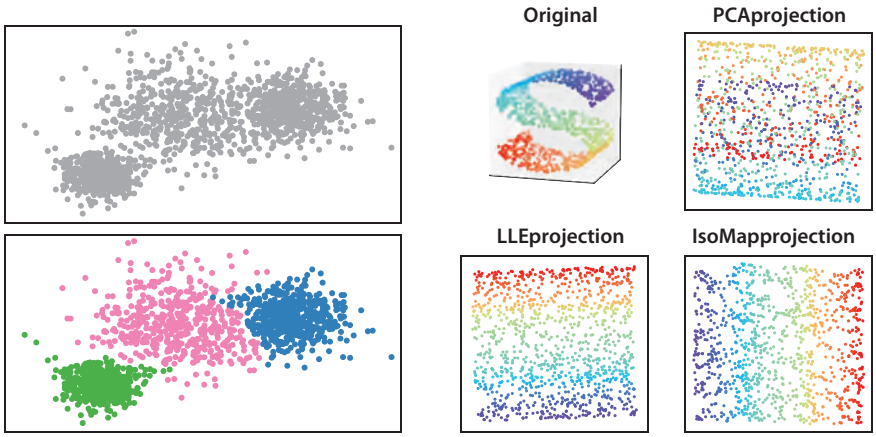

Figure 1.8: Left: An example of a clustering algorithm applied to a set of points. The algorithm assigns each point to one of three clusters. Right: Different examples of dimensionality reduction algorithms, with varying degrees of information loss. Figure from the `sklearn` manual [Pedregosa et al., 2011].

then apply a supervised learning method. Or we can recycle pieces of pre-trained algorithms to accelerate the training and learning process for a different but related data set or problem. Science progresses with rigor and creativity; machine learning methods provide a versatile range of techniques to tackle problems in innovative ways.

1.6 MACHINE LEARNING VERSUS INFERENCE

Machine learning methods can be used to solve many problems, as we will see as we continue our journey. But of course many research questions would be better approached from a different perspective, or put more simply, other approaches can be equally beneficial. If we think of machine learning as a means to build an implicit relationship between input and output (whether the output is a quantity, a rule, or a pattern), the alternative is often seen as classical *inference*, where we explicitly specify what the output would look like *as a function of the input features, with some parameters chosen by us*. This model could be a simple mathematical formula or the result of a complex numerical simulation; the important aspect is that we are able to predict y, the output, for a given value of x and the model parameters. The goal of the inference process is to determine the model's *parameters*, unlike in machine learning, where the goal is (usually) to make a prediction.

Let us consider a very simple example.

Suppose that a friend comes to you and says: I have this list of time measurements of a car moving at uniform speed. My wicked physics teacher asked me to

figure out where the car would be at time $t = 12$ seconds. I have no idea of how to proceed. Can you help? These are my data:

time (s)	distance (m)
0	5.1
1	5.5
2	8.4
3	11.1
4	11.8
5	14.4
6	16.1
7	19.5
8	20.2
9	23.1

So you—being a good scientist—would certainly know that you can use this well-tested formula for the relationship between the input (time) and the output (distance traveled):

$$d = d_0 + v \times t \tag{1.1}$$

This model has two parameters: the initial distance (or coordinate), d_0, measured at time $t = 0$, and the speed, v, which is constant, as we know from your friend's description. But how can we work out what the correct values are to plug in the formula for d_0 and v? This process is called *parameter fitting*.

There are many great books that describe this process in detail (some of which you can find in the References at the end of this book), so I'll just mention a very basic approach.

First of all, since we have only two coordinates, it's always a good idea to look at our data. The left panel of Figure 1.9 shows that the data lie approximately on

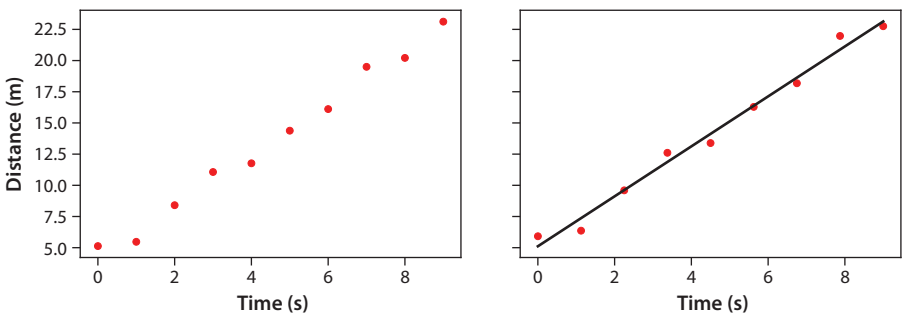

Figure 1.9: A simple example of fitting data by using a linear model.

a straight line, as expected; the deviation can be interpreted as the consequence of statistical error (measurement noise). The parameters d_0 and v that we want to measure are the y intercept and the slope of this hypothetical line.

The main idea of parameter fitting is to try out many different combinations (in this case, pairs) of parameter values until we find something that works well. So we need at least three steps: 1. Decide on a range of possible values to try out; 2. Decide how to pick parameters; and 3. Establish an evaluation criterion that tells us whether a model is good or bad.

For the first step, a visual inspection of our graph tells us that probably the y intercept of a line that goes approximately through the points will be between 4 and 6, and the slope of the line will safely be between 1 and 3 (we can get a quick estimate by calculating slopes as rise over run for a couple of time intervals). Because our data space is very small, we can use a brute-force approach, dividing our ranges in equally spaced intervals and trying out every possible combination. We can either decide the spacing between different values we want to consider, or the total number of points; for this example, let us consider 100 values, which corresponds to a 0.02 spacing. The third step requires building a measure of the distance between the predictions of each model (i.e., the 10 distances provided by the model, given the 10 times, slopes, and y intercepts) and the observed data (the nine measured distances). In the simplest case where there is no uncertainty associated with each measurement (which is deeply unphysical!), we can use the square of the Euclidean distance between the 10 distances predicted by the model (D_i^m) and the 10 observed distances (D_i^o):

$$\Sigma_{i=1}^{10}(D^m(d_0, v, x_i) - D_i^o)^2. \tag{1.2}$$

Better models should be closer to the observed data, so the combination of parameters that generates the minimum distance can be considered the winning model, or *best fit*. In the real world, where every measurement has an uncertainty associated to it, the distance should be weighed against the uncertainty, so that points with large measurement errors contribute less to the total distance. If some assumptions about the distribution of uncertainties can be made, this process corresponds to writing a χ^2 distribution, or a *likelihood*. As already mentioned, there is a vast literature on this subject; see, for example, [Hogg et al., 2010] and [Hastie et al., 2001].

To summarize, we pick every possible pair of (d_0, v) according to the chosen range and spacing, calculate the above metric, and choose the pair that minimizes it. We can do this easily as a Python exercise. We start by choosing an appropriate range and spacing for the parameter values:

```
slopes = np.linspace(1,3,101)
intercepts = np.linspace(4,6,101)
```

Then we define our model and fitness metric, the sum of squared errors:

```
def model(x,m,b):
return m*x+b

def se(m,b,x,y):
return np.sum(((model(x,m,b) - y)**2))
```

Finally, we calculate the squared error for all combinations of parameters and choose the ones that lead to the minimum error:

```
square_errs = np.array([[se(m,b,x,y) for b in intercepts] for m in slopes])
indices = np.unravel_index(square_errs.argmin(), square_errs.shape)
bestm, bestb = slopes[indices[0]],intercepts[indices[1]]
```

Through this process, we end up with a minimum distance value $d_{\min} \sim 3.65$, corresponding to a y intercept (d_0 in the language of our problem) of 4.34 and a slope (v) of 2.04. The corresponding line is plotted in Figure 1.9, so we can convince ourselves that our procedure works.

Finally, we can use the equation of the line we found—in other words, our model—to predict the coordinate of the car at other times, for example, the $t = 12$s required by the physics problem:

$$d = d_0 + v \times t = 4.34\,\text{m} + 2.04\,\text{m/s} \times 12\,\text{s} = 28.82\,\text{m}. \tag{1.3}$$

What would be the corresponding approach in machine learning? The main difference is that we won't explicitly write out a model, so we won't write parameters or a likelihood; in general, we are not required to be able to predict the observed outcome of a given combination of parameters. However, in practice the choice of ML algorithm or method that we make will affect what kind of input/output (I/O) relationship can be represented by our model, as well as its ability to learn from the training data.

In this case, we are dealing with a supervised learning problem, and we have 10 points in our learning set. As we know from the previous sections, we need to split them in a training set and a test set. For now, let us assume that we will use seven of them for training and three of them for testing; the test/train split can be selected randomly. Note that, in general, this is not allowed in time series problems; but for the purpose of our problem, the fact that our independent (input) variable is a time is not relevant, so making this choice is OK.

Our problem is a regression problem, because we are predicting a continuous variable. So we need to pick an algorithm that can handle regression, and the metric we use to estimate how good our model is will also be sensitive to the distance

between predicted and observed points; for example, it could be the mean squared error (MSE), which is in fact the square of the Euclidean distance along the y axis, averaged over the number of points.

Even if we are not properly defining machine learning algorithms yet, for the sake of the argument, let us try out two very simple ones: a linear regressor and a decision tree. You can see the implementation in the lecture notebook "Straight Line with ML.ipynb."

We can select a training set for each model, which contains seven points from the learning set. In `scikit-learn` or `sklearn` [Pedregosa et al., 2011], the Python package for machine learning that will be our main software library throughout the book, we can do this easily by using the auxiliary function `train_test_split` and fixing the random seed for reproducibility:

```
np.random.seed(10)
X_train, X_test, y_train, y_test = train_test_split(x, y, test_size=3)
```

In this case, the training set is made by instances with time coordinates 6, 3, 1, 0, 7, 4, and 9 s.

After training each model, we can ask them to predict the y coordinates (i.e., traveled distances) of the points in the test set, which have x coordinates 8, 2, and 5, respectively. This is an example for the Decision Tree algorithm (note that `scikit-learn` is imported in code as `sklearn`, and for this reason, we will use the latter notation throughout the book):

```
from sklearn.tree import DecisionTreeRegressor
treemodel = DecisionTreeRegressor()
y_pred_tree = treemodel.fit(X_train.reshape(-1, 1),y_train).predict(X_test.reshape
(-1, 1))
```

The responses of the two algorithms are as follows:

	x = 8	**x = 2**	**x = 5**	**MSE**
True values y(x)	20.2	8.4	14.4	
Prediction LR	20.9	8.42	14.7	0.18
Prediction DT	19.5	5.5	11.8	5.22

The built-in evaluation process of the machine learning approach tells us immediately that the linear regression model is superior to the decision tree model, because the mean squared error on the test set is lower; the decision tree model is quite severely underestimating each prediction.

Just as in the inference exercise above, we can now use either model to predict the traveled distance at time $t = 12$ s; the prediction of the linear regression model is 29.2 m, and the prediction of the decision tree model is 23.1 m.

1.6.1 Who fit it better?

The simple example above showed us two approaches to solving the same problem: one using classic inference, the other relying on machine learning methods. Both can be used to output a prediction. So which one is better? There is no hard-and-fast rule, but we can try to focus on some differences, and perhaps debunk some myths at the same time.

In the inference approach, we choose the functional form of the input/output relation explicitly, in parametric form, and we optimize the *model parameters*. This strategy is usually convenient when we have a good understanding (e.g., from physical principles, as in this case) of which variables matter and how different variables are related, or when we are looking at a very simple problem and data set. It is intuition driven; I like to joke that it is only as good as the scientist, but in general it's only as good as the model. If our choice of model is unphysical (e.g., if we had tried to model our data with a sinusoidal curve instead of a straight line), the parameter values we would have obtained would have not made any sense, and the model would have no predictive power.

Machine learning approaches are, in principle, more model agnostic, although as we saw above, the results obtained can be significantly algorithm dependent, because the choice of algorithm may shape the forms of input/output relations that we can explore. Even when we use very flexible ML methods, such as deep neural networks, which can alleviate this problem, they can only be good as the data that are used to build them. As a general and simplistic rule, I think that when we understand the physics but don't have data, we should use a probabilistic approach to model fitting, and when we don't understand the physics but we have data, we should use machine learning. Figure 1.10 summarizes some more of my thoughts on either approach; often, the synergy between these two approaches can be most powerful.

As a final note, I should add that distinguishing between probabilistic inference and the machine learning approach as I did here is too naive, and rather incorrect. There is more overlap between the two methods than Figure 1.10 implies, and ML methods can serve the purpose of probabilistic inference very well (e.g., see the recent review of simulation-based inference [Cranmer et al., 2020]). However, I think this framework is useful for building understanding of the scope and limits of ML methods for beginning practitioners, and I will stand by it for this introductory textbook.

1.6.2 The black box issue

Even after solving a problem using machine learning, the input/output relationship is never obtained explicitly, so the process of making predictions for new instances consists of feeding the trained algorithm new data and receiving some numbers

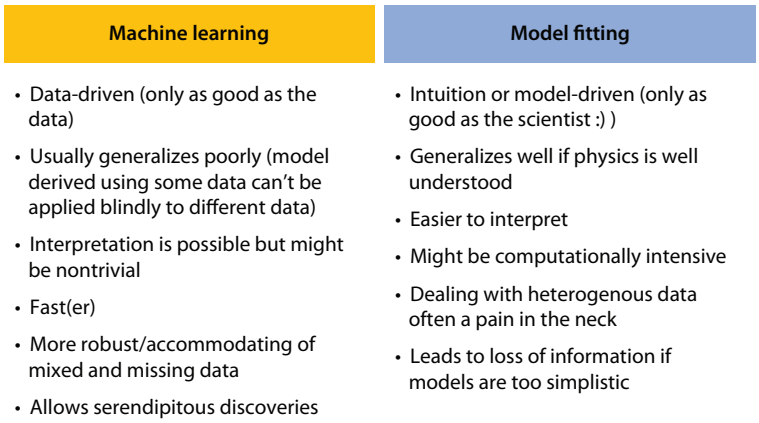

Figure 1.10: Some advantages and disadvantages of the two approaches. My claim is that synergy is often the best strategy. Figure from [Acquaviva, 2019].

in return. The fact that we cannot simply write down an equation has gained ML models the infamous moniker of "black boxes." However, there are many ways of gathering insights on the nature of the trained model, and in my opinion, a good scientist would want to open the black box. This approach has the double advantage of validating the model and possibly gaining new information by analyzing the reasons for the algorithm's response. We will look at some ways to gather intelligence from ML models throughout this book.

1.6.3 There is no magic!

Although I do, in many ways, "believe the hype" about machine learning, and I think it is important for scientists to become proficient with these methods, I also think we should keep in mind that ML techniques are just tools of the trade, not the trade itself. When I was in graduate school, the most useful class I took was one about numerical methods; we learned from a giant book titled Numerical Recipes; I've often joked that ML methods are the Numerical Recipes[3] of this decade. In fact, many ML algorithms are nicely packaged, linear-algebra-based sequences of operations. They enable us to solve some problems in new, or more efficient, ways; as always, though, it's part of a scientist's job to understand how to tackle a difficult problem, with or without machine learning. One of my hopes for this book is to generate some understanding of when machine learning can help and of the strengths and weaknesses of specific methods. In all cases, and especially when we lose some transparency because of complex mathematics, it is essential to stick to the rigorous process of hypothesis testing spelled out by the scientific method.

3 http://numerical.recipes/

1.7 REVIEW AND DISCUSSION QUESTIONS

Note: Questions and exercises marked by ** are more complex, open-ended, or time consuming.

Exercise 1. For each one of the scenarios described here, answer the following questions. 1. Is it a supervised or unsupervised learning problem? 2. Is it a classification or regression task? 3. What could be useful features (data) to collect? You should be able to motivate your answers; several of them can be interpreted in multiple ways, so the argument will be more important than the answer!

Scenario 1. Email providers placing emails in the "Spam" folder.
Scenario 2. House selling prices: imagine that you are a real estate agent and want to come up with a data-driven listing price.
Scenario 3. Predicting used car prices (same as before, but now you sell cars).
Scenario 4. Twitter showing "trending topics."
Scenario 5. Recommending sizes for clothes bought online.
Scenario 6. Guessing a college student's major from personal information.
Scenario 7. Counting the number of people appearing in a collection of photos.
Scenario 8. Predicting the number of hours a college student will sleep tonight.

Note: This also works well as class exercise, or think/pair/share.

Exercise 2. Popular services like Netflix or Spotify include a "recommendation engine" for movies or music. Discuss similarities and differences in how they collect information, what information they collect, and how they use it to provide suggestions.

Exercise 3. For each of the following problems, discuss whether they could be partially or completely solved with machine learning. If the answer is positive, specify whether you would use supervised or unsupervised learning, the potential features, target properties if relevant, and potential process of data collection:

Problem 1. Determining the constant of gravity.
Problem 2. Determining the specific heat of a liquid.
Problem 3. Counting the number of sources in an astronomical image.

Exercise 4. ** Come up with a problem (related to your area of research or personal interest) that is better solved through classic inference and with a problem that is better solved through machine learning.

1.8 PROGRAMMING EXERCISES

Exercise 1. The linear model we used should be mathematically equivalent to the linear regression ML model, but the predictions for the distance at time $t = 12$ s are slightly different. Can you figure out why? Hint: There are (at least) two different reasons!

Exercise 2. ** Modify the "ModelingStraightLine.ipynb" notebook to include measurement errors for all the measured distances generated from a normal distribution with mean = 0 and variance = 2.0 (set the random seed to ensure that your results are reproducible). Assume that the errors on the time measurements are negligible. Modify the evaluation metric for the model to inverse-weigh the uncertainties. Does the best fit model change? Why or why not?

Exercise 3. ** Use the Dark Energy exercise notebook ("DarkEnergyFromSupernovae.ipynb") to find evidence for the existence of Dark Energy from supernovae data with classic inference methods.

Index