6

Pattern Recognition: Learning from Experience

The Analytical Engine has no pretensions whatever to *originate* anything. It can do whatever we *know how to order it* to perform.

—ADA LOVELACE, from her 1843 notes on the Analytical Engine

In each previous chapter, we've looked at an area in which the ability of computers far outstrips the ability of humans. For example, a computer can typically encrypt or decrypt a large file within a second or two, whereas it would take a human many years to perform the same computations by hand. For an even more extreme example, imagine how long it would take a human to manually compute the PageRank of billions of web pages according to the algorithm described in chapter 3. This task is so vast that, in practice, it is impossible for a human. Yet the computers at web search companies are constantly performing these computations.

In this chapter, on the other hand, we examine an area in which humans have a natural advantage: the field of *pattern recognition*. Pattern recognition is a subset of artificial intelligence and includes tasks such as face recognition, object recognition, speech recognition, and handwriting recognition. More specific examples would include the task of determining whether a given photograph is a picture of your sister, or determining the city and state written on a hand-addressed envelope. Thus, pattern recognition can be defined more generally as the task of getting computers to act "intelligently" based on input data that contains a lot of variability.

The word "intelligently" is in quotation marks here for good reason: the question of whether computers can ever exhibit true intelligence is highly controversial. The opening quotation of this chapter represents one of the earliest salvos in this debate: Ada Lovelace commenting, in 1843, on the design of an early mechanical computer called the Analytical Engine. Lovelace is sometimes described as the world's first computer programmer because of her profound insights about the Analytical Engine. But in this pronouncement, she emphasizes that computers lack originality: they must slavishly follow the

instructions of their human programmers. These days, computer scientists disagree on whether computers can, in principle, exhibit intelligence. And the debate becomes even more complex if philosophers, neuroscientists, and theologians are thrown into the mix.

Fortunately, we don't have to resolve the paradoxes of machine intelligence here. For our purposes, we might as well replace the word "intelligent" with "useful." So the basic task of pattern recognition is to take some data with extremely high variability—such as photographs of different faces in different lighting conditions, or samples of many different words handwritten by many different people—and do something useful with it. Humans can unquestionably process such data intelligently: we can recognize faces with uncanny accuracy, and read the handwriting of virtually anyone without having to see samples of their writing in advance. It turns out that computers are vastly inferior to humans at such tasks. But some ingenious algorithms have emerged that enable computers to achieve good performance on certain pattern recognition tasks. In this chapter, we will learn about three of these algorithms: nearest-neighbor classifiers, decision trees, and artificial neural networks. But first, we need a more scientific description of the problem we are trying to solve.

WHAT'S THE PROBLEM?

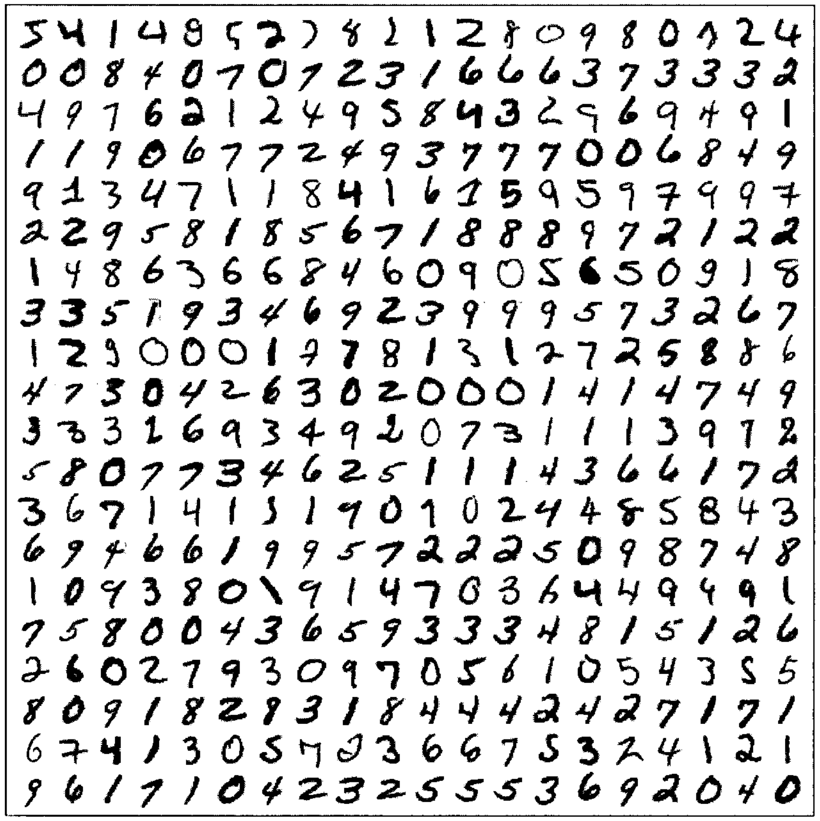
The tasks of pattern recognition might seem, at first, to be almost absurdly diverse. Can computers use a single toolbox of pattern recognition techniques to recognize handwriting, faces, speech, and more? One possible answer to this question is staring us (literally) in the face: our own human brains achieve superb speed and accuracy in a wide array of recognition tasks. Could we write a computer program to achieve the same thing?

Before we can discuss the techniques that such a program might use, we need to somehow unify the bewildering array of tasks and define a single problem that we are trying to solve. The standard approach here is to view pattern recognition as a *classification* problem. We assume that the data to be processed is divided up into sensible chunks called *samples*, and that each sample belongs to one of a fixed set of possible *classes*. For example, in a face recognition problem, each sample would be a picture of a face, and the classes would be the identities of the people the system can recognize. In some problems, there are only two classes. A common example of this is in medical diagnosis for a particular disease, where the two classes might be "healthy" and "sick," while each data sample could consist of all the test results for a single patient (e.g., blood pressure, weight, x-ray images, and possibly many other things). So the computer's task is to process new data samples that it has never seen before and *classify* each sample into one of the possible classes.

To make things concrete, let's focus on a single pattern recognition task for now. This is the task of recognizing handwritten digits. Some typical data samples are shown in the figure on the facing page. There are exactly ten classes in this problem: the digits 0, 1, 2, 3, 4, 5, 6, 7, 8, and 9. So the task is to classify samples of handwritten digits as belonging to one of these ten classes. This is, of course, a problem of great practical significance, since mail in the United States and many other countries is addressed using numeric postcodes. If a computer can rapidly and accurately recognize these postcodes, mail can be sorted by machines much more efficiently than by humans.

Obviously, computers have no built-in knowledge of what handwritten digits look like. And, in fact, humans don't have this built-in knowledge either: we *learn* how to recognize digits and other handwriting, through some combination of explicit teaching by other humans and by seeing examples that we use to teach ourselves. These two strategies (explicit teaching and learning from examples) are also used in computer pattern recognition. However, it turns out that for all but the simplest of tasks, explicit teaching of computers is ineffective. For example, we can think of the climate controls in my house as a simple classification system. A data sample consists of the current temperature and time of day, and the three possible classes are "heat on," "air-conditioning on," and "both off." Because I work in an office during the day, I program the system to be "both off" during daytime hours, and outside those hours it is "heat on" if the temperature is too low and "air-conditioning on" if the temperature is too high. Thus, in the process of programming my thermostat, I have in some sense "taught" the system to perform classification into these three classes.

Unfortunately, no one has ever been able to explicitly "teach" a computer to solve more interesting classification tasks, such as the handwritten digits on the next page. So computer scientists turn to the other strategy available: getting a computer to automatically "learn" how to classify samples. The basic strategy is to give the computer a large amount of *labeled data*: samples that have already been classified. The figure on page 84 shows an example of some labeled data for the handwritten digit task. Because each sample comes with a label (i.e., its class), the computer can use various analytical tricks to extract characteristics of each class. When it is later presented with an unlabeled sample, the computer can guess its class by choosing the one whose characteristics are most similar to the unlabeled sample.

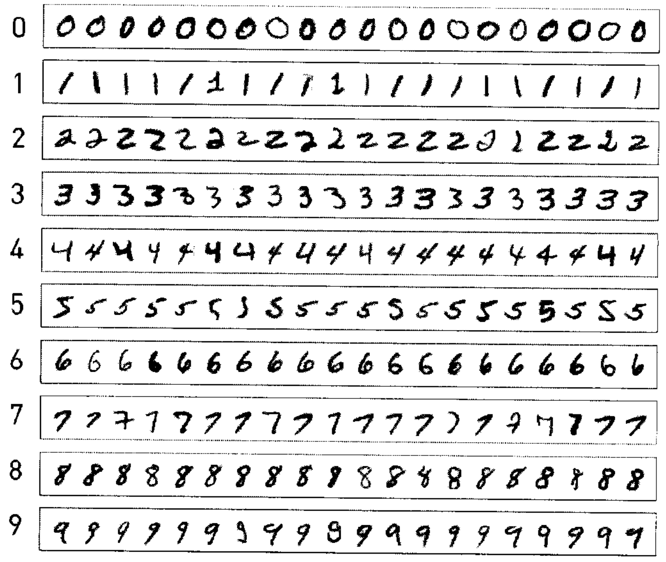


Most pattern recognition tasks can be phrased as classification problems. Here, the task is to classify each handwritten digit as one of the 10 digits 0, 1, . . . , 9. Data source: MNIST data of LeCun *et al.* 1998.

The process of learning the characteristics of each class is often called "training," and the labeled data itself is the "training data." So in a nutshell, pattern recognition tasks are divided into two phases: first, a training phase in which the computer learns about the classes based on some labeled training data; and second, a classification phase in which the computer classifies new, unlabeled data samples.

THE NEAREST-NEIGHBOR TRICK

Here's an interesting classification task: can you predict, based only on a person's home address, which political party that person will

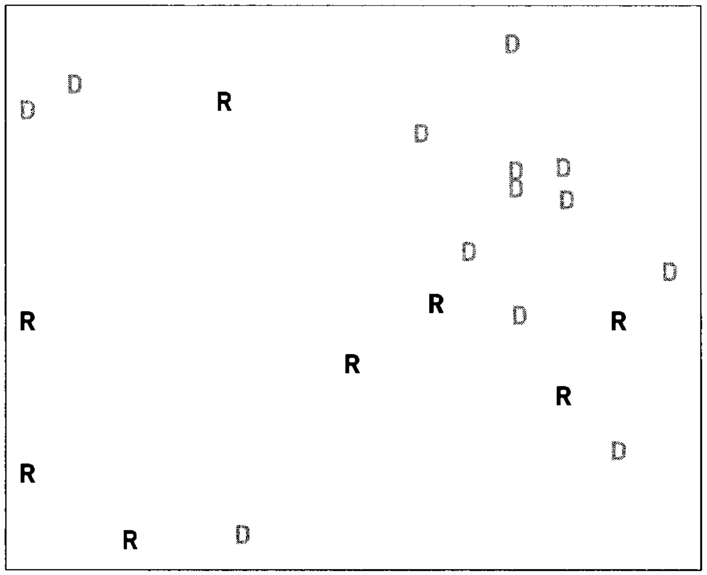


To train a classifier, a computer needs some labeled data. Here, each sample of data (a handwritten digit) comes with a label specifying one of the 10 possible digits. The labels are on the left, and the training samples are in boxes on the right. Data source: MNIST data of LeCun *et al.* 1998.

make a donation to? Obviously, this is an example of a classification task that cannot be performed with perfect accuracy, even by a human: a person's address doesn't tell us enough to predict political affiliations. But, nevertheless, we would like to train a classification system that predicts which party a person is *most likely* to donate to, based only on a home address.

The figure on the next page shows some training data that could be used for this task. It shows a map of the actual donations made by the residents of a particular neighborhood in Kansas, in the 2008 U.S. presidential election. (In case you are interested, this is the College Hill neighborhood of Wichita, Kansas.) For clarity, streets are not shown on the map, but the actual geographic location of each house that made a donation is shown accurately. Houses that donated to the Democrats are marked with a "D," and an "R" marks donations to the Republicans.

So much for the training data. What are we going to do when given a new sample that needs to be classified as either Democrat or Republican? The figure on page 86 shows this concretely. The training data is shown as before, but in addition there are two new locations shown as question marks. Let's focus first on the upper question mark. Just

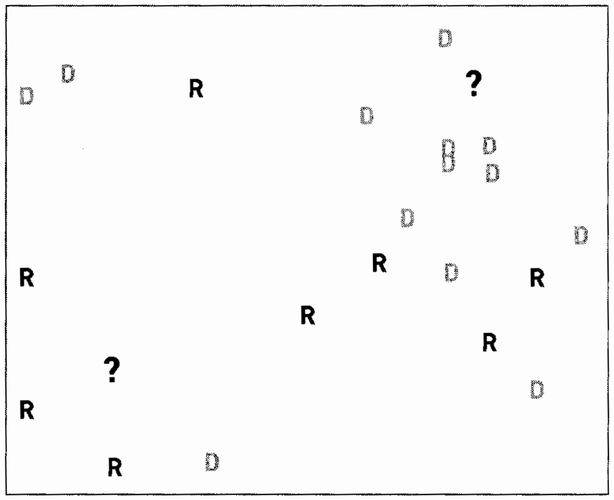


Training data for predicting political party donations. A "D" marks a house that donated to the Democrats, and "R" marks Republican donations. Data source: Fundrace project, Huffington Post.

by glancing at it, and without trying to do anything scientific, what would you guess is the most likely class for this question mark? It seems to be surrounded by Democratic donations, so a "D" seems quite probable. How about the other question mark, on the lower left? This one isn't exactly surrounded by Republican donations, but it does seem to be more in Republican territory than Democrat, so "R" would be a good guess.

Believe it or not, we have just mastered one of the most powerful and useful pattern recognition techniques ever invented: an approach that computer scientists call the *nearest-neighbor classifier*. In its simplest form, this "nearest-neighbor" trick does just what it sounds like. When you are given an unclassified data sample, first find the nearest neighbor to that sample in the training data and then use the class of this nearest neighbor as your prediction. In the figure on the next page, this just amounts to guessing the closest letter to each of the question marks.

A slightly more sophisticated version of this trick is known as "*K*-nearest-neighbors," where *K* is a small number like 3 or 5. In this formulation, you examine the *K* nearest neighbors of the question mark and choose the class that is most popular among these neighbors. We can see this in action in the figure on page 87. Here, the nearest



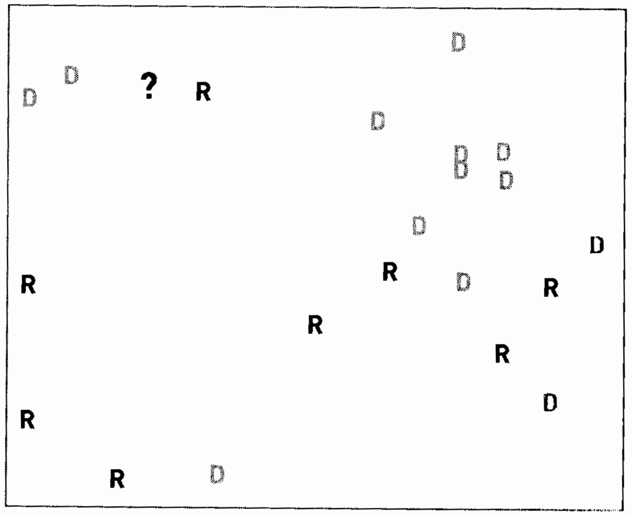
Classification using the nearest-neighbor trick. Each question mark is assigned the class of its nearest neighbor. The upper question mark becomes a "D," while the lower one becomes an "R." Data source: Fundrace project, Huffington Post.

single neighbor to the question mark is a Republican donation, so the simplest form of the nearest-neighbor trick would classify this question mark as an "R." But if we move to using 3 nearest neighbors, we find that this includes two Democrat donations and one Republican donation—so in this particular set of neighbors, Democrat donations are more popular and the question mark is classified as a "D."

So, how many neighbors should we use? The answer depends on the problem being tackled. Typically, practitioners try a few different values and see what works best. This might sound unscientific, but it reflects the reality of effective pattern recognition systems, which are generally crafted using a combination of mathematical insight, good judgment, and practical experience.

Different Kinds of "Nearest" Neighbors

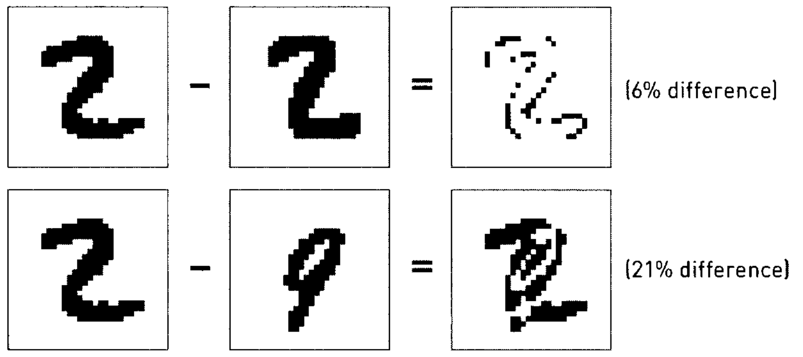
So far, we've worked on a problem that was deliberately chosen to have a simple, intuitive interpretation of what it means for one data sample to be the "nearest" neighbor of another data sample. Because each data point was located on a map, we could just use the geographic distance between points to work out which ones were closest. But what are we going to do when each data sample is a handwritten digit like the ones on page 83? We need some



An example of using K-nearest-neighbors. When using only the single nearest neighbor, the question mark is classified as an "R," but with three nearest neighbors, it becomes a "D." Data source: Fundrace project, Huffington Post.

way of computing the "distance" between two different examples of handwritten digits. The figure on the following page shows one way of doing this.

The basic idea is to measure the difference between images of digits, rather than a geographical distance between them. The difference will be measured as a percentage—so images that are only 1% different are very close neighbors, and images that are 99% different are very far from each other. The figure shows specific examples. (As is usual in pattern recognition tasks, the inputs have undergone certain preprocessing steps. In this case, each digit is rescaled to be the same size as the others and centered within its image.) In the top row of the figure, we see two different images of handwritten 2's. By doing a sort of "subtraction" of these images, we can produce the image on the right, which is white everywhere except at the few places where the two images were different. It turns out that only 6% of this difference image is black, so these two examples of handwritten 2's are relatively close neighbors. On the other hand, in the bottom row of the figure, we see the results when images of different digits (a 2 and a 9) are subtracted. The difference image on the right has many more black pixels, because the images disagree in more places. In fact, about 21% of this image is black, so the two images are not particularly close neighbors.



Computing the "distance" between two handwritten digits. In each row, the second image is subtracted from the first one, resulting in a new image on the right that highlights the differences between the two images. The percentage of this difference image that is highlighted can be regarded as a "distance" between the original images. Data source: MNIST data of LeCun *et al.*, 1998.

Now that we know how to find out the "distance" between handwritten digits, it's easy to build a pattern recognition system for them. We start off with a large amount of training data—just as in the figure on page 84, but with a much bigger number of examples. Typical systems of this sort might use 100,000 labeled examples. Now, when the system is presented with a new, unlabeled handwritten digit, it can search through all 100,000 examples to find the single example that is the closest neighbor to the one being classified. Remember, when we say "closest neighbor" here, we really mean the smallest percentage difference, as computed by the method in the figure above. The unlabeled digit is assigned the same label as this nearest neighbor.

It turns out that a system using this type of "closest neighbor" distance works rather well, with about 97% accuracy. Researchers have put enormous effort into coming up with more sophisticated definitions for the "closest neighbor" distance. With a state-of-the-art distance measure, nearest-neighbor classifiers can achieve over 99.5% accuracy on handwritten digits, which is comparable to the performance of much more complex pattern recognition systems, with fancy-sounding names such as "support vector machines" and "convolutional neural networks." The nearest-neighbor trick is truly a wonder of computer science, combining elegant simplicity with remarkable effectiveness.

It was emphasized earlier that pattern recognition systems work in two phases: a *learning* (or training) phase in which the training data is processed to extract some characteristics of the classes, and a *classification* phase in which new, unlabeled data is classified. So, what happened to the learning phase in the nearest-neighbor classifier we've examined so far? It seems as though we take the training data, don't bother to learn anything from it, and jump straight into classification using the nearest-neighbor trick. This happens to be a special property of nearest-neighbor classifiers: they don't require any explicit learning phase. In the next section, we'll look at a different type of classifier in which learning plays a much more important role.

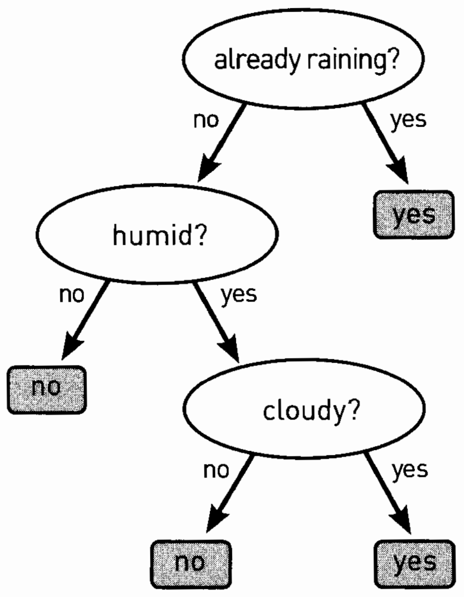
THE TWENTY-QUESTIONS TRICK: DECISION TREES

The game of "twenty questions" holds a special fascination for computer scientists. In this game, one player thinks of an object, and the other players have to guess the identity of the object based only on the answers to no more than twenty yes-no questions. You can even buy small handheld devices that will play twenty questions against you. Although this game is most often used to entertain children, it is surprisingly rewarding to play as an adult. After a few minutes, you start to realize that there are "good questions" and "bad questions." The good questions are guaranteed to give you a large amount of "information" (whatever that means), while the bad ones are not. For example, it's a bad idea to ask "Is it made of copper?" as your first question, because if the answer is "no," the range of possibilities has been narrowed very little. These intuitions about good questions and bad questions lie at the heart of a fascinating field called information theory. And they are also central to a simple and powerful pattern recognition technique called *decision trees.*

A decision tree is basically just a pre-planned game of twenty questions. The figure on the next page shows a trivial example. It's a decision tree for deciding whether or not to take an umbrella with you. You just start at the top of the tree and follow the answers to the questions. When you arrive at one of the boxes at the bottom of the tree, you have the final output.

You are probably wondering what this has to do with pattern recognition and classification. Well, it turns out that if you are given a sufficient amount of training data, it is possible to *learn* a decision tree that will produce accurate classifications.

Let's look at an example based on the little-known, but extremely important, problem known as *web spam.* We already encountered this in chapter 3, where we saw how some unscrupulous website operators try to manipulate the ranking algorithms of search engines



Decision tree for "Should I take an umbrella?"

by creating an artificially large number of hyperlinks to certain pages. A related strategy used by these devious webmasters is to create web pages that are of no use to humans, but with specially crafted content. You can see a small excerpt from a real web spam page in the figure on the facing page. Notice how the text makes no sense, but repeatedly lists popular search terms related to online learning. This particular piece of web spam is trying to increase the ranking of certain online learning sites that it provides links to.

Naturally, search engines expend a lot of effort on trying to identify and eliminate web spam. It's a perfect application for pattern recognition: we can acquire a large amount of training data (in this case, web pages), manually label them as "spam" or "not spam," and train some kind of classifier. That's exactly what some scientists at Microsoft Research did in 2006. They discovered that the best-performing classifier on this particular problem was an old favorite: the decision tree. You can see a small part of the decision tree they came up with on page 92.

Although the full tree relies on many different attributes, the part shown here focuses on the popularity of the words in the page. Web spammers like to include a large number of popular words in order to improve their rankings, so a small percentage of popular words indicates a low likelihood of spam. That explains the first decision in the tree, and the others follow a similar logic. This tree achieves an

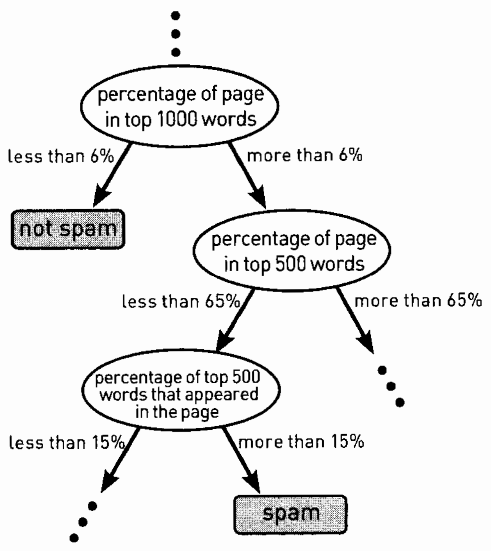
human resource management study,
web based distance education
Magic language learning online mba certificate and self-directed learning—various law degree online study, on online an education an graduate an degree. Living it consulting and computer training courses. So web development degree for continuing medical education conference, news indiana online education, none college degree online service information systems management program—in computer engineering technology program set online classes and mba new language learning online degrees online nursing continuing education credits, dark distance education graduate hot pc service and support course.

Excerpt from a page of "web spam." This page contains no information useful to humans—its sole purpose is to manipulate web search rankings. Source: Ntoulas *et al.* 2006.

accuracy of about 90%—far from perfect, but nevertheless an invaluable weapon against web spammers.

The important thing to understand is not the details of the tree itself, but the fact that the entire tree was generated automatically, by a computer program, based on training data from about 17,000 web pages. These "training" pages were classified as spam or not spam by a real person. Good pattern recognition systems can require significant manual effort, but this is a one-time investment that has a many-time payoff.

In contrast to the nearest-neighbor classifier we discussed earlier, the learning phase of a decision tree classifier is substantial. How does this learning phase work? The main intuition is the same as planning a good game of twenty questions. The computer tests out a huge number of possible first questions to find the one that yields the best possible information. It then divides the training examples into two groups, depending on their answer to the first question and comes up with a best possible second question for each of those groups. And it keeps on moving down the tree in this way, always determining the best question based on the set of training examples that reach a particular point in the tree. If the set of examples ever becomes "pure" at a particular point—that is, the set contains only spam pages or only non-spam pages—the computer can


percentage of page in top 1000 words
less than 6%
more than 6%
not spam
percentage of page in top 500 words
less than 65%
more than 65%
percentage of top 500 words that appeared in the page
less than 15%
more than 15%
spam

Part of a decision tree for identifying web spam. The dots indicate parts of the tree that have been omitted for simplicity. Source: Ntoulas *et al.* 2006.

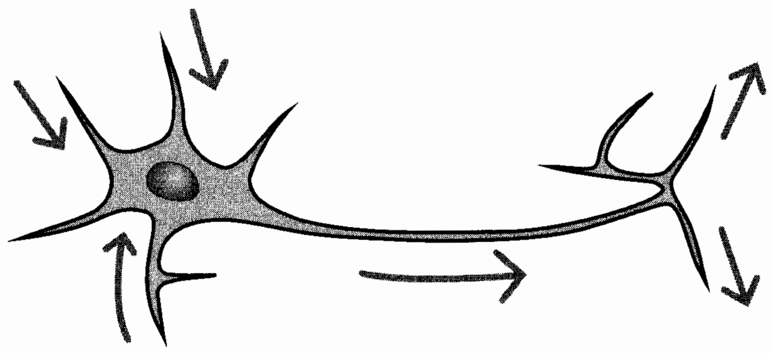
stop generating new questions and instead output the answer corresponding to the remaining pages.

To summarize, the learning phase of a decision tree classifier can be complex, but it is completely automatic and you only have to do it once. After that, you have the decision tree you need, and the classification phase is incredibly simple: just like a game of twenty questions, you move down the tree following the answers to the questions, until you reach an output box. Typically, only a handful of questions are needed and the classification phase is thus extremely efficient. Contrast this with the nearest-neighbor approach, in which no effort was required for the learning phase, but the classification phase required us to do a comparison with all training examples (100,000 of them for the hand-written digits task), for each item to be classified.

In the next section, we encounter neural networks: a pattern recognition technique in which the learning phase is not only significant, but directly inspired by the way humans and other animals learn from their surroundings.

NEURAL NETWORKS

The remarkable abilities of the human brain have fascinated and inspired computer scientists ever since the creation of the first digital computers. One of the earliest discussions of actually simulating a brain using a computer was by Alan Turing, a British scientist who was also a superb mathematician, engineer, and code-breaker. Turing's classic 1950 paper, entitled *Computing Machinery and Intelligence*, is most famous for a philosophical discussion of whether a computer could masquerade as a human. The paper introduced a scientific way of evaluating the similarity between computers and humans, known these days as a "Turing test." But in a less well-known passage of the same paper, Turing directly analyzed the possibility of modeling a human brain using a computer. He estimated that only a few gigabytes of memory might be sufficient.



A typical biological neuron. Electrical signals flow in the directions shown by the arrows. The output signals are only transmitted if the sum of the input signals is large enough.

Sixty years later, it's generally agreed that Turing significantly underestimated the amount of work required to simulate a human brain. But computer scientists have nevertheless pursued this goal in many different guises. One of the results is the field of *artificial neural networks*, or neural networks for short.

Biological Neural Networks

To help us understand artificial neural networks, we first need an overview of how real, biological neural networks function. Animal brains consist of cells called neurons, and each neuron is connected to many other neurons. Neurons can send electrical and chemical signals through these connections. Some of the connections are set up to *receive* signals from other neurons; the remaining connections *transmit* signals to other neurons (see the figure above).

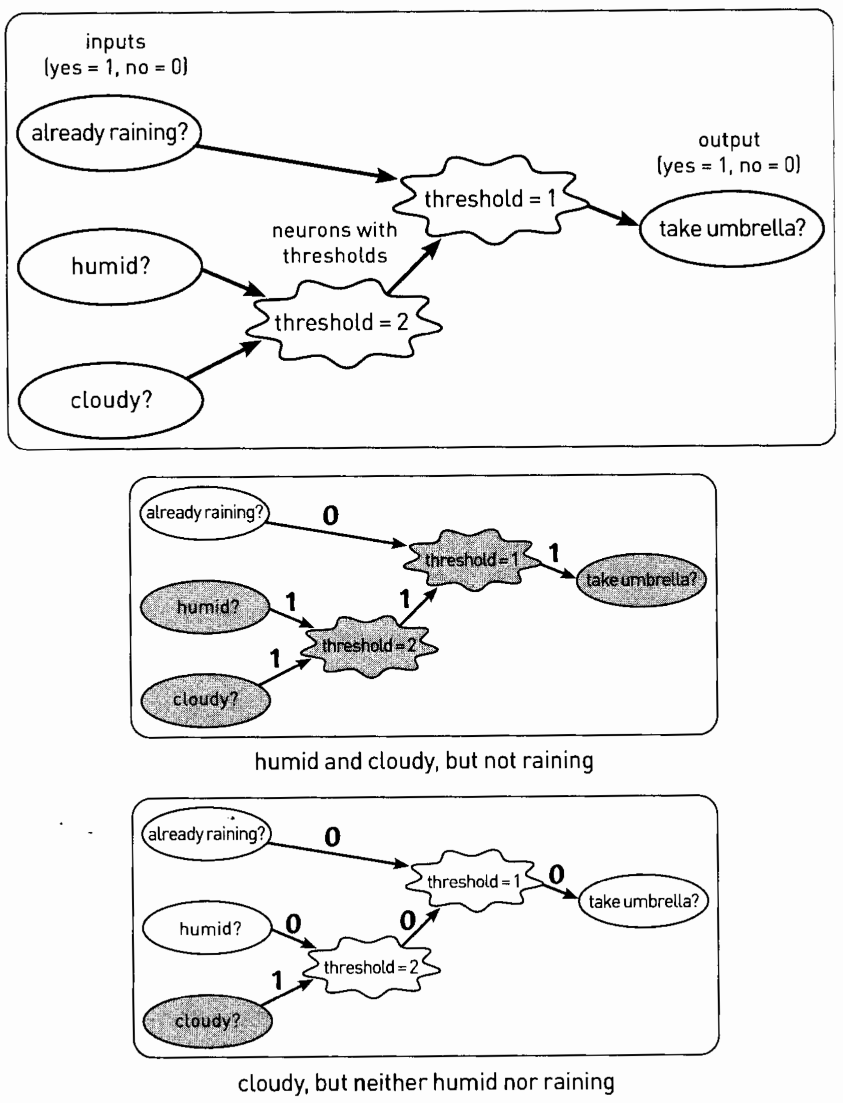
One simple way of describing these signals is to say that at any given moment a neuron is either "idle" or "firing." When it's idle, a neuron isn't transmitting any signals; when it's firing, a neuron sends frequent bursts of signals through all of its outgoing connections. How does a neuron decide when to fire? It all depends on the strength of the incoming signals it is receiving. Typically, if the total of all incoming signals is strong enough, the neuron will start firing; otherwise, it will remain idle. Roughly speaking, then, the neuron "adds up" all of the inputs it is receiving and starts firing if the sum is large enough. One important refinement of this description is that there are actually two types of inputs, called *excitatory* and *inhibitory*. The strengths of the excitatory inputs are added up just as you would expect, but the inhibitory inputs are instead *subtracted* from the total—so a strong inhibitory input tends to prevent the neuron from firing.

A Neural Network for the Umbrella Problem

An artificial neural network is a computer model that represents a tiny fraction of a brain, with highly simplified operations. We'll initially discuss a basic version of artificial neural networks, which works well for the umbrella problem considered earlier. After that, we'll use a neural network with more sophisticated features to tackle a problem called the "sunglasses problem."

Each neuron in our basic model is assigned a number called its *threshold*. When the model is running, each neuron adds up the inputs it is receiving. If the sum of the inputs is at least as large as the threshold, the neuron fires, and otherwise it remains idle. The figure on the next page shows a neural network for the extremely simple umbrella problem considered earlier. On the left, we have three inputs to the network. You can think of these as being analogous to the sensory inputs in an animal brain. Just as our eyes and ears trigger electrical and chemical signals that are sent to neurons in our brains, the three inputs in the figure send signals to the neurons in the artificial neural network. The three inputs in this network are all excitatory. Each one transmits a signal of strength +1 if its corresponding condition is true. For example, if it is currently cloudy, then the input labeled "cloudy?" sends out an excitatory signal of strength +1; otherwise, it sends nothing, which is equivalent to a signal of strength zero.

If we ignore the inputs and outputs, this network has only two neurons, each with a different threshold. The neuron with inputs for humidity and cloudiness fires only if both of its inputs are active (i.e., its threshold is 2), whereas the other neuron fires if any one of its inputs is active (i.e., its threshold is 1). The effect of this is shown



Top panel: A neural network for the umbrella problem. Bottom two panels: The umbrella neural network in operation. Neurons, inputs, and outputs that are "firing" are shaded. In the center panel, the inputs state that it is not raining, but it is both humid and cloudy, resulting in a decision to take an umbrella. In the bottom panel, the only active input is "cloudy?," which feeds through to a decision not to take an umbrella.

Faces to be "recognized" by a neural network. In fact, instead of recognizing faces, we will tackle the simpler problem of determining whether a face is wearing sunglasses. Source: Tom Mitchell, *Machine Learning*, McGraw-Hill (1998). Used with permission.

in the bottom of the figure on the previous page, where you can see how the final output can change depending on the inputs.

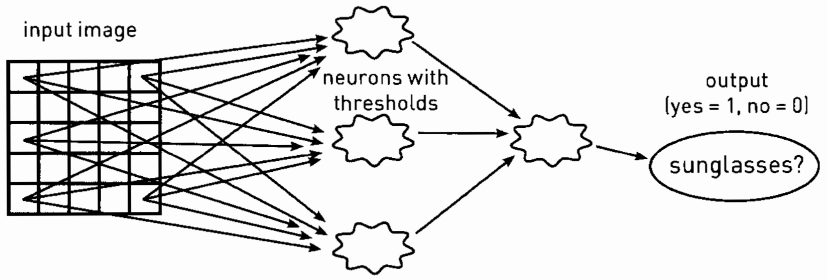
At this point, it would be well worth your while to look back at the decision tree for the umbrella problem on page 90. It turns out that the decision tree and the neural network produce exactly the same results when given the same inputs. For this very simple, artificial problem, the decision tree is probably a more appropriate representation. But we will next look at a much more complex problem that demonstrates the true power of neural networks.

A Neural Network for the Sunglasses Problem

As an example of a realistic problem that can be successfully solved using neural networks, we'll be tackling a task called the "sunglasses problem." The input to this problem is a database of low-resolution photographs of faces. The faces in the database appear in a variety of configurations: some of them look directly at the camera, some look up, some look to the left or right, and some are wearing sunglasses. The figure above shows some examples.

We are deliberately working with low-resolution images here, to make our neural networks easy to describe. Each of these images is, in fact, only 30 pixels wide and 30 pixels high. As we will soon see, however, a neural network can produce surprisingly good results with such coarse inputs.

Neural networks can be used to perform standard face recognition on this face database—that is, to determine the identity of the person in a photograph, regardless of whether the person is looking at the camera or disguised with sunglasses. But here, we will attack an easier problem, which will demonstrate the properties of neural networks more clearly. Our objective will be to decide whether or not a given face is wearing sunglasses.



A neural network for the sunglasses problem.

The figure above shows the basic structure of the network. This figure is schematic, since it doesn't show every neuron or every connection in the actual network used. The most obvious feature is the single output neuron on the right, which produces a 1 if the input image contains sunglasses and a 0 otherwise. In the center of the network, we see three neurons that receive signals directly from the input image and send signals on to the output neuron. The most complicated part of the network is on the left, where we see the connections from the input image to the central neurons. Although all the connections aren't shown, the actual network has a connection from every pixel in the input image to every central neuron. Some quick arithmetic will show you that this leads to a rather large number of connections. Recall that we are using low-resolution images that are 30 pixels wide and 30 pixels high. So even these images, which are tiny by modern standards, contain $30 \times 30 = 900$ pixels. And there are three central neurons, leading to a total of $3 \times 900 = 2700$ connections in the left-hand layer of this network.

How was the structure of this network determined? Could the neurons have been connected differently? The answer is yes, there are many different network structures that would give good results for the sunglasses problem. The choice of a network structure is often based on previous experience of what works well. Once again, we see that working with pattern recognition systems requires insight and intuition.

Unfortunately, as we shall soon see, each of the 2700 connections in the network we have chosen needs to be "tuned" in a certain way before the network will operate correctly. How can we possibly handle this complexity, which involves tuning thousands of different connections? The answer will turn out to be that the tuning is done automatically, by learning from training examples.

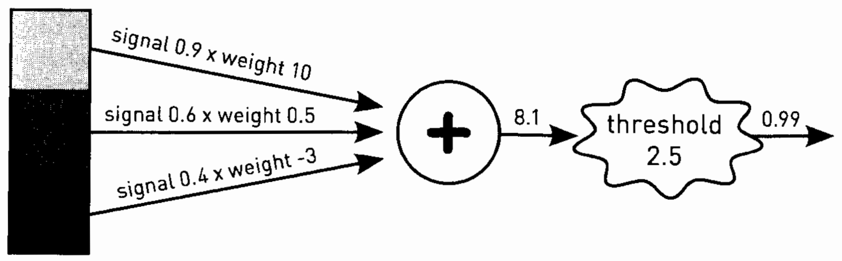
Adding Weighted Signals

As mentioned earlier, our network for the umbrella problem used a basic version of artificial neural networks. For the sunglasses problem, we'll be adding three significant enhancements.

Enhancement 1: *Signals can take any value between 0 and 1 inclusive.* This contrasts with the umbrella network, in which the input and output signals were restricted to equal 0 or 1 and could not take any intermediate values. In other words, signal values in our new network can be, for example, 0.0023 or 0.755. To make this concrete, think about our sunglasses example. The brightness of a pixel in an input image corresponds to the signal value sent over that pixel's connections. So a pixel that is perfectly white sends a value of 1, whereas a perfectly black pixel sends a value of 0. The various shades of gray result in corresponding values between 0 and 1.

Enhancement 2: *Total input is computed from a weighted sum.* In the umbrella network, neurons added up their inputs without altering them in any way. In practice, however, neural networks take into account that every connection can have a different strength. The strength of a connection is represented by a number called the connection's *weight.* A weight can be any positive or negative number. Large positive weights (e.g., 51.2) represent strong excitatory connections—when a signal passes through a connection like this, its downstream neuron is likely to fire. Large negative weights (e.g., −121.8) represent strong inhibitory connections—a signal on this type of connection will probably cause the downstream neuron to remain idle. Connections with small weights (e.g., 0.03 or −0.0074) have little influence on whether their downstream neurons fire. (In reality, a weight is defined as "large" or "small" only in comparison to other weights, so the numerical examples given here only make sense if we assume they are on connections to the same neuron.) When a neuron computes the total of its inputs, each input signal is multiplied by the weight of its connection before being added to the total. So large weights have more influence than small ones, and it is possible for excitatory and inhibitory signals to cancel each other out.

Enhancement 3: *The effect of the threshold is softened.* A threshold no longer clamps its neuron's output to be either fully on (i.e., 1) or fully off (i.e., 0); the output can be any value between 0 and 1 inclusive. When the total input is well below the threshold, the output is close to 0, and when the total input is well above the threshold, the output is close to 1. But a total input near the threshold can produce an intermediate output value near 0.5. For example, consider a neuron with threshold 6.2. An input of 122 might produce an output of 0.995, since the input is much greater than the threshold. But an input of 6.1 is close to the threshold and might produce an output of 0.45. This effect occurs at all neurons, including the final output neuron. In our sunglasses application, this means that output values near 1 strongly suggest the presence of sunglasses, and output values near 0 strongly suggest their absence.



Signals are multiplied by a connection weight before being summed.

The figure above demonstrates our new type of artificial neuron with all three enhancements. This neuron receives inputs from three pixels: a bright pixel (signal 0.9), a medium-bright pixel (signal 0.6), and a darker pixel (signal 0.4). The weights of these pixels' connections to the neuron happen to be 10, 0.5, and −3, respectively. The signals are multiplied by the weights and then added up, which produces a total incoming signal for the neuron of 8.1. Because 8.1 is significantly larger than the neuron's threshold of 2.5, the output is very close to 1.

Tuning a Neural Network by Learning

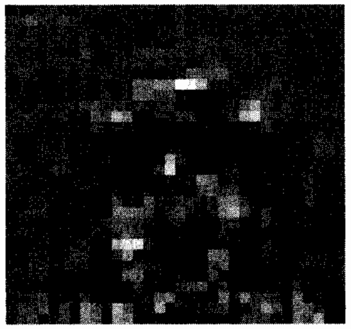
Now we are ready to define what it means to tune an artificial neural network. First, every connection (and remember, there could be many thousands of these) must have its weight set to a value that could be positive (excitatory) or negative (inhibitory). Second, every neuron must have its threshold set to an appropriate value. You can think of the weights and thresholds as being small dials on the network, each of which can be turned up and down like a dimmer on an electric light switch.

To set these dials by hand would, of course, be prohibitively time-consuming. Instead, we can use a computer to set the dials during a learning phase. Initially, the dials are set to random values. (This may seem excessively arbitrary, but it is exactly what professionals do in real applications.) Then, the computer is presented with its first training sample. In our application, this would be a picture of a person who may or may not be wearing sunglasses. This sample is run through the network, which produces a single output value between 0 and 1. However, because the sample is a *training* sample, we know the "target" value that the network should ideally produce. The key trick is to alter the network slightly so that its output is closer to the desired target value. Suppose, for example, that the first training sample happens to contain sunglasses. Then the target value is 1. Therefore, every dial in the entire network is adjusted by a tiny amount, in the direction that will move the network's output value toward the target of 1. If the first training sample did not contain sunglasses, every dial would be moved a tiny amount in the opposite direction, so that the output value moves toward the target 0. You can probably see immediately how this process continues. The network is presented with each training sample in turn, and every dial is adjusted to improve the performance of the network. After running through all of the training samples many times, the network typically reaches a good level of performance and the learning phase is terminated with the dials at the current settings.

The details of how to calculate these tiny adjustments to the dials are actually rather important, but they require some math that is beyond the scope of this book. The tool we need is multivariable calculus, which is typically taught as a mid-level college math course. Yes, math *is* important! Also, note that the approach described here, which experts call "stochastic gradient descent," is just one of many accepted methods for training neural networks.

All these methods have the same flavor, so let's concentrate on the big picture: the learning phase for a neural network is rather laborious, involving repeated adjustment of all the weights and thresholds until the network performs well on the training samples. However, all this can be done automatically by a computer, and the result is a network that can be used to classify new samples in a simple and efficient manner.

Let's see how this works out for the sunglasses application. Once the learning phase has been completed, every one of the several thousand connections from the input image to the central neurons has been assigned a numerical weight. If we concentrate on the connections from all pixels to just one of the neurons (say, the top one), we can visualize these weights in a very convenient way, by transforming them into an image. This visualization of the weights is shown in the figure on the next page, for just one of the central neurons. For this particular visualization, strong excitatory connections (i.e., with large positive weights) are white, and strong inhibitory connections

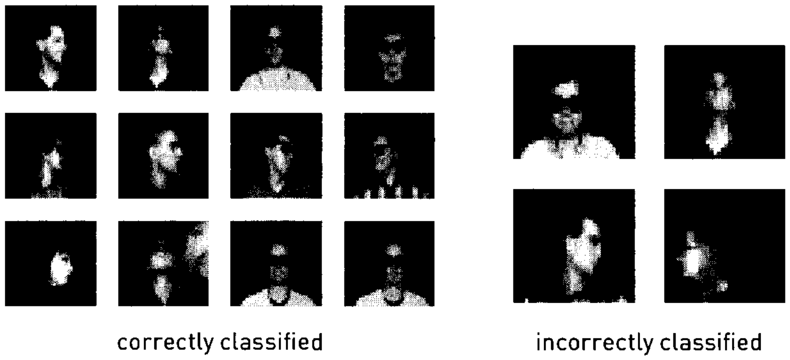


Weights (i.e., strengths) of inputs to one of the central neurons in the sunglasses network.

(i.e., with large negative weights) are black. Various shades of gray are used for connections of intermediate strength. Each weight is shown in its corresponding pixel location. Take a careful look at the figure. There is a very obvious swath of strong inhibitory weights in the region where sunglasses would typically appear—in fact, you could almost convince yourself that this image of weights actually contains a picture of some sunglasses. We might call this a "ghost" of sunglasses, since they don't represent any particular sunglasses that exist.

The appearance of this ghost is rather remarkable when you consider that the weights were not set using any human-provided knowledge about the typical color and location of sunglasses. The *only* information provided by humans was a set of training images, each with a simple "yes" or "no" to specify whether sunglasses were present. The ghost of sunglasses emerged automatically from the repeated adjustment of the weights in the learning phase.

On the other hand, it's clear that there are plenty of strong weights in other parts of the image, which should—in theory—have no impact on the sunglasses decision. How can we account for these meaningless, apparently random, connections? We have encountered here one of the most important lessons learned by artificial intelligence researchers in the last few decades: it is possible for seemingly intelligent behavior to emerge from seemingly random systems. In a way, this should not be surprising. If we had the ability to go into our own brains and analyze the strength of the connections between the neurons, the vast majority would appear random. And yet, when acting as an ensemble, these ramshackle collections of connection strengths produce our own intelligent behavior!



Results from the sunglasses network. Source: Tom Mitchell, *Machine Learning*, McGraw-Hill (1998). Used with permission.

Using the Sunglasses Network

Now that we are using a network that can output any value between 0 and 1, you may be wondering how we get a final answer—is the person wearing sunglasses or not? The correct technique here is surprisingly simple: an output above 0.5 is treated as "sunglasses," while an output below 0.5 yields "no sunglasses."

To test our sunglasses network, I ran an experiment. The face database contains about 600 images, so I used 400 images for learning the network and then tested the performance of the network on the remaining 200 images. In this experiment, the final accuracy of the sunglasses network turned out to be around 85%. In other words, the network gives a correct answer to the question "is this person wearing sunglasses?" on about 85% of images that it has never seen before. The figure above shows some of the images that were classified correctly and incorrectly. It's always fascinating to examine the instances on which a pattern recognition algorithm fails, and this neural network is no exception. One or two of the incorrectly classified images in the right panel of the figure are genuinely difficult examples that even a human might find ambiguous. However, there is at least one (the top left image in the right panel) that appears, to us humans, to be absolutely obvious—a man staring straight at the camera and clearly wearing sunglasses. Occasional mysterious failures of this type are not at all unusual in pattern recognition tasks.

Of course, state-of-the-art neural networks could achieve much better than 85% correctness on this problem. The focus here has been on using a simple network, in order to understand the main ideas involved.

PATTERN RECOGNITION: PAST, PRESENT, AND FUTURE

As mentioned earlier, pattern recognition is one aspect of the larger field of artificial intelligence, or AI. Whereas pattern recognition deals with highly variable input data such as audio, photos, and video, AI includes more diverse tasks, including computer chess, online chat-bots, and humanoid robotics.

AI started off with a bang: at a conference at Dartmouth College in 1956, a group of ten scientists essentially founded the field, popularizing the very phrase "artificial intelligence" for the first time. In the bold words of the funding proposal for the conference, which its organizers sent to the Rockefeller Foundation, their discussions would "proceed on the basis of the conjecture that every aspect of learning or any other feature of intelligence can in principle be so precisely described that a machine can be made to simulate it."

The Dartmouth conference promised much, but the subsequent decades delivered little. The high hopes of researchers, perennially convinced that the key breakthrough to genuinely "intelligent" machines was just over the horizon, were repeatedly dashed as their prototypes continued to produce mechanistic behavior. Even advances in neural networks did little to change this: after various bursts of promising activity, scientists ran up against the same brick wall of mechanistic behavior.

Slowly but surely, however, AI has been chipping away at the collection of thought processes that might be defined as uniquely human. For years, many believed that the intuition and insight of human chess champions would beat any computer program, which must necessarily rely on a deterministic set of rules rather than intuition. Yet this apparent stumbling block for AI was convincingly eradicated in 1997, when IBM's Deep Blue computer beat world champion Garry Kasparov.

Meanwhile, the success stories of AI were gradually creeping into the lives of ordinary people too. Automated telephone systems, servicing customers through speech recognition, became the norm. Computer-controlled opponents in video games began to exhibit human-like strategies, even including personality traits and foibles. Online services such as Amazon and Netflix began to recommend items based on automatically inferred individual preferences, often with surprisingly pleasing results.

Indeed, our very perceptions of these tasks have been fundamentally altered by the progress of artificial intelligence. Consider a task that, in 1990, indisputably required the intelligent input of humans, who would actually be paid for their expertise: planning the itinerary of a multistop plane trip. In 1990, a good human travel agent could make a huge difference in finding a convenient and low-cost itinerary. By 2010, however, this task was performed better by computers than humans. Exactly how computers achieve this would be an interesting story in itself, as they do use several fascinating algorithms for planning itineraries. But even more important is the effect of the systems on our *perception* of the task. I would argue that by 2010, the task of planning an itinerary was perceived as purely mechanistic by a significant majority of humans—in stark contrast to the perception 20 years earlier.

This gradual transformation of tasks, from apparently intuitive to obviously mechanistic, is continuing. Both AI in general and pattern recognition in particular are slowly extending their reach and improving their performance. The algorithms described in this chapter—nearest-neighbor classifiers, decision trees, and neural networks—can be applied to an immense range of practical problems. These include correcting fat-fingered text entry on cell phone virtual keyboards, helping to diagnose a patient's illness from a complex battery of test results, recognizing car license plates at automated toll booths, and determining which advertisement to display to a particular computer user—to name just a few. Thus, these algorithms are some of the fundamental building blocks of pattern recognition systems. Whether or not you consider them to be truly "intelligent," you can expect to see a lot more of them in the years ahead.

7

Data Compression: Something for Nothing

Emma was gratified, and would soon have shewn no want of words, if the sound of Mrs Elton's voice from the sitting-room had not checked her, and made it expedient to compress all her friendly and all her congratulatory sensations into a very, very earnest shake of the hand.

—JANE AUSTEN, *Emma*

We're all familiar with the idea of *compressing* physical objects: when you try to fit a lot of clothes into a small suitcase, you can squash the clothes so that they are small enough to fit even though they would overflow the suitcase at their normal size. You have *compressed* the clothes. Later, you can *decompress* the clothes after they come out of a suitcase and (hopefully) wear them again in their original size and shape.

Remarkably, it's possible to do exactly the same thing with information: computer files and other kinds of data can often be compressed to a smaller size for easy storage or transportation. Later, they are decompressed and used in their original form.

Most people have plenty of disk space on their own computers and don't need to bother about compressing their own files. So it's tempting to think that compression doesn't affect most of us. But this impression is wrong: in fact, compression is used behind the scenes in computer systems quite often. For example, many of the messages sent over the internet are compressed without the user even knowing it, and almost all software is downloaded in compressed form—this means your downloads and file transfers are often several times quicker than they otherwise would be. Even your voice gets compressed when you speak on the phone: telephone companies can achieve a vastly superior utilization of their resources if they compress voice data before transporting it.

Compression is used in more obvious ways, too. The popular ZIP file format employs an ingenious compression algorithm that will