



The science of speech recognition

Markov chains are the bedrock of voice recognition technology, and they turn up in plenty of other places too, from simple board games to finance and economics

Once upon a time, in a hypothetical land far, far away, a computer scientist noticed that the weather followed a certain pattern. His first observation was that there were never two sunny days in a row, but that it was equally likely to be cloudy or raining the day after a sunny day. If it was cloudy one day, it was equally likely to be cloudy, raining, or sunny the next. If it rained, however, half the time it would be sunny the next day and half the time it would continue to rain. Indeed, after making these observations he decided to move to Florida – the weather is much better there.

What is a Markov chain

Because he was a computer scientist, he naturally thought of these observations as a state machine with three states: sunny, raining, and cloudy. He drew up the diagram in Figure 1, overleaf where the transitions from state to state are labelled with the probabilities of making the transition. The probabilities of what the weather will be like tomorrow are only dependent on what it's like today.

So, let's suppose that it's sunny today. Tomorrow it will be raining 50 per cent of the time and cloudy 50 per cent of the time, and there's no chance that it'll be sunny. The day after, the probabilities of sun, rain, and cloud will be 0.417, 0.417, and 0.167 respectively. The day after that, they will be 0.264, 0.473, and 0.264. Our mythical computer scientist

then tossed aside his calculator and wrote a small program to calculate the probability of it being sunny, rainy, or cloudy n days in the future, and found that after about eight days the probabilities were always 0.308, 0.462 and 0.231 to three decimal places. (You can now see why he wanted to relocate to somewhere sunnier.) Interestingly, it doesn't matter what the weather is like today: after about eight days the probabilities settle down to the same values. If you like, the probabilities of the weather reach a steady state. If you've studied advanced mathematics, the steady state is known as the eigenvector of the transition probability matrix with eigenvalue one.

Andrey Markov, a Russian mathematician from the late 19th century, was the first to look at these kinds of state machines, although he referred to them as 'chains' (they subsequently became known as Markov chains). The essential properties that define a Markov chain are that it's a random process, stepping through a finite set of individual states, with the probabilities of transitioning from state to state being well defined. The simplest chain is where the transition probabilities from state to state only depend on the current state. In our storybook example, the transition probabilities only depend on what the weather conditions are today; they don't depend on what the weather was like yesterday or last week.

More complex chains can be formed by assuming that the transition probabilities

depend not just on the current state, but also on the previous two or more states.

A Markov chain of order m is one in which the probabilities for transitioning from a state depend on it and the previous $m-1$ states (so the simple Markov chain is of order one).

Real-world uses

So, having defined what a Markov chain is, what can we use it for? Or, more simply, where does it turn up? The first example is from information theory. Claude Shannon, in his seminal paper *A Mathematical Theory of Communication*, used a Markov chain on the English language to show that there's information redundancy in ordinary text (the number of bits of information in English text – its entropy, to give it its official name – is much less than the entropy of a random string and so, waving my information theory hands in the air to whisk away some deep mathematics, English text is compressible).

Let's assume that there are some 40 states in an English language Markov chain – one for each letter and punctuation mark. By analysing a collection of texts (Shakespeare, Dickens and Rushdie, for example), we can discover the transition probabilities to move from state to state (that is, from letter to letter). So, assuming the current letter is a t, the probability of an h, a vowel or another t are much higher than any other letter. Shannon showed that this Markov chain meant that long ▶

- English text has about 1.3 bits of information per character; that is, 6.7 bits of redundancy per byte. A different encoding scheme would lower the number of bits of redundancy and, as it happens, David A Huffman devised a compression algorithm (now known as Huffman encoding) to minimise the information redundancy for a message text. Further results from Shannon's paper are used by cryptanalysts to work out possible plaintexts for an encrypted message. The basic methodology goes like this: because ordinary language has so few bits of information per letter (compared to a random sequence), that redundancy is more likely to 'show' itself through the encrypted message. The more redundancy in the original message, the fewer plaintexts are possible for a given encrypted message. Cryptanalysts use this to try to break encryption. The better encryption programs therefore compress the plaintext message to reduce its redundancy, then encrypt the result.

Text generation

The second example is to use a language Markov chain to generate text. This can then be used as input in prototypes of applications instead of real text. The best Markov chain here is an order-2 chain based on the words in the analysed texts (that is, you calculate the probabilities for the words that can follow two other words in the original texts and those

Spotlight on... The Viterbi algorithm

The Viterbi algorithm, which was devised by electrical engineer Andrew Viterbi in 1967, is a dynamic programming algorithm used to determine the parameters of a hidden Markov model. The algorithm makes several assumptions about the model: first that the model is an order-1 Markov chain with a finite number of states, and second that the observed events (outputs) are in a sequence with the hidden events. The sequence of hidden and observed events is usually assumed to be ordered according to time.

In any Markov chain, there are probably many paths through the state machine that will reach a particular state at a given time t .

While all of those paths are feasible, there's one that's more likely (has a higher probability) than the rest. Therefore, for all states, the algorithm maintains the most likely path to reach each at that time t . Hence, in order to work out the most likely path to reach an observed output at time $t+1$, at the next step in other words, the algorithm uses this previously calculated information in order to produce the most likely path. This is an example of typical dynamic programming approach: break up the original problem into more easily solved sub-problems, then store the 'best' results for each sub-problem and combine them in some fashion to solve the larger problem. ■

form the transition matrix). An example from my tutorial on this topic in PC Plus 272 gave this from *War of the Worlds*: "The war of the adjacent houses. I had left them nearly four weeks ago. A monstrous tripod, higher than the provision and wine shops. A couple of sturdy roughs who had gone northward. I spent that night and a special edition."

Of course, there's nothing stopping you from analysing, say, the nine symphonies of Dvorak into a Markov chain and then

generating a new symphony from that. Since I know next to nothing about the structure of music, I'll leave it at that: a possibility.

The third example of the use of a Markov chain is Google's PageRank algorithm, originally described by Larry Page (the 'Page' in 'PageRank') and Sergey Brin. Here we take the Markov chain built over all the pages on the internet and calculate the probabilities for transitioning from page X to page Y . In essence, a page's PageRank is the probability of ending up on that page after many clicks (in other words, we calculate the eigenvector for the matrix that covers the internet).

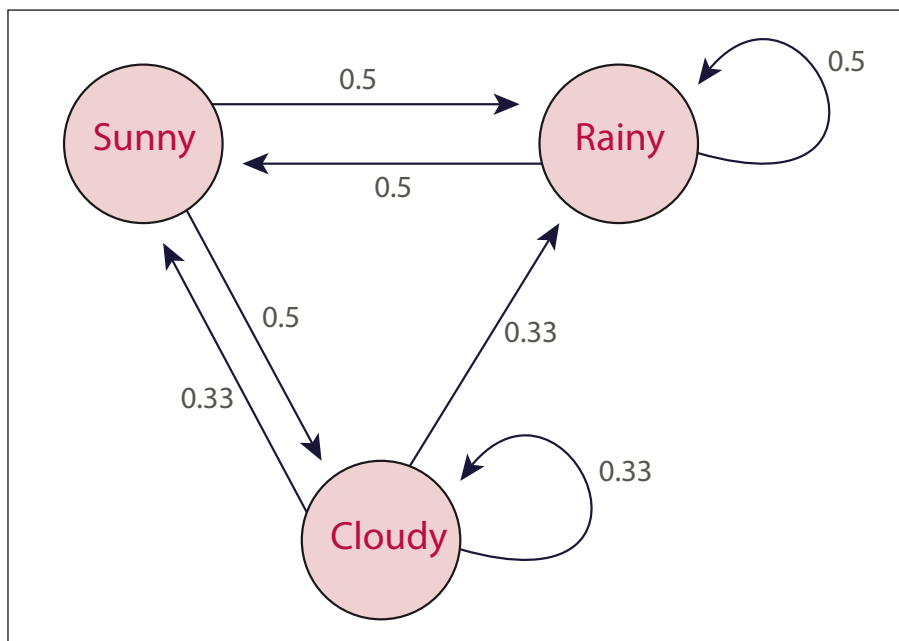
Extending these examples, you can easily visualise the use of Markov chains in finance (to model asset prices), economics (the general equilibrium theory is based on a Markov model) and games (snakes and ladders is a Markov chain, as is any game whose state transitions are determined by purely by dice).

Hidden chains

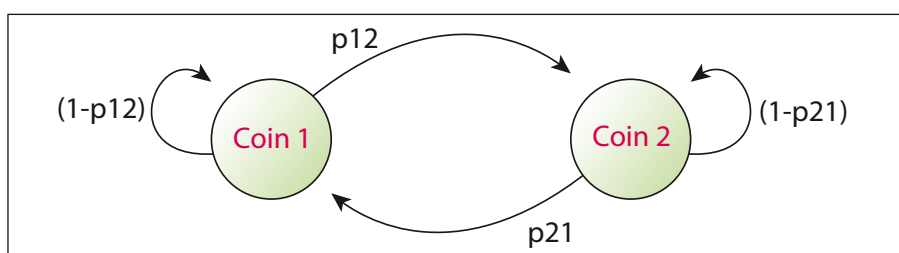
An interesting situation to consider is one where you can only see the outputs of a Markov process, not the states, but where you're certain that there's some kind of chain governing those outputs. Many real-world processes are of this nature – you can see the output of the process without seeing into the black box. These outputs are sometimes known as signals.

In essence, we have a statistical problem: we have to try to discover as much as we can about the black box from the inputs and outputs. We need to throw in a lot of inputs and analyse the outputs statistically to try to discover the parameters governing the behaviour of the black box. The black box is known as a hidden Markov model (or HMM).

In his paper *A Tutorial on HMM and Selected Applications in Speech Recognition*, Lawrence Rabiner invited us to imagine the following thought experiment. You're in a room and can't see anything outside. There's a screen, on which you can see the results of another person doing some kind of coin-flip experiment. The other person won't tell you



▲ Figure 1: A state machine for the weather in a land long ago and far away.



▲ Figure 2: The two-coin state machine for the coin-flip hidden Markov model.

what kind of coin flips he's doing, only that the process is a Markov chain with well-defined probabilities. What can you determine from the sequence of heads and tails you see on the screen? What model can you devise that will mimic the hidden Markov chain?

The first possibility in this situation is that the person is flipping a single, possibly biased, coin. That is, the probability of heads (and therefore tails) is fixed. This is a degenerate Markov chain, in that no knowledge of the current state is required; whether the current state is heads or tails, the probability of getting heads (or tails) is always the same. We have to work out a single probability – the probability of throwing heads. Statistically, it would be approximately the total number of heads divided by the total number of flips, but that measure would be too coarse to explain possible clumping visible in the output data.

The second possibility is that the person is using a two-coin system (figure 2). In such a system, the coins are viewed as separate states and the play goes like this: use some external random process (say a random number generator program) to determine whether to transition from the current coin to the next, then flip the coin for the state you're in and record the result. Here we have to determine not only the bias of each coin (they could be different), but the probabilities to transition from one coin to the other. We have to work out four different probabilities (two coin-flip probabilities and two transition probabilities).

Unknown complexity

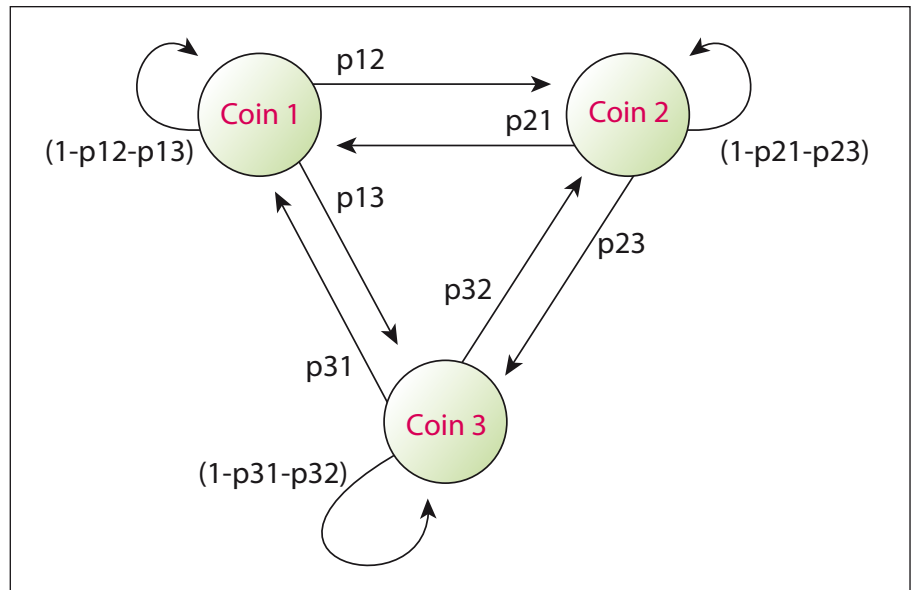
The next possibility is even more complicated: the person is using a three-coin system. Here we'd have to work out nine probabilities for our hidden Markov model (three coin-flip and six transition probabilities). Of course, the hidden Markov model could be even more complex than this, and involve four or more coins.

Even with such a simple experiment, the possibilities multiply quickly. Should we

Random walks

One application of Markov chains is in the study of random walks. These are paths through some defined space where each step made is a random process. The simplest example is the drunkard's walk: imagine that the space is the number line extending infinitely to the negative and positive. Start at zero. At each step, you flip a coin: heads you take a step of -1; tails a step of +1. The resulting path bounces around the number line, much like a drunkard walking unsteadily along a pavement. Refinements include modifying the probabilities at each possible position so that, say, there is a bias towards zero (call it gravity) but the further away you are, the less this bias is felt. Other refinements change the space to a grid or a lattice (so you get a drunkard's walk around Manhattan, for example). All are examples of Markov chains.

The importance of random walks is that they can be used as models of diffusion, for models of stock prices and so on. There's an equity options pricing algorithm called the Black-Scholes model, which uses a Gaussian random walk at its heart. ■



▲ **Figure 3:** The three-coin state machine for the coin-flip hidden Markov model.

assume a three-coin model, when it's possible that a one-coin model is being used? One of the standard algorithms for determining the most likely sequence of states that produce the output stream is the Viterbi algorithm.

There are three basic problems when trying to create an HMM. The first is to determine the probability that a given HMM produced the observed sequence of outputs. If you like, given the three models I described for the coin-toss experiment, which one is more likely to have produced the sequence? Once we've settled on a model, the second problem is to try to determine the 'best possible' state sequence for our model that produced the output sequence. The third is to tweak or optimise the parameters of our selected model in order to better fit the output sequence; in other words, training the model to more favourably recognise the output sequence.

As you will have noticed, Rabiner was looking at HMMs in order to solve a particular problem: that of speech recognition.

With speech recognition, the input is an audio stream. The stream is divided up into small samples of sound (say, 10 milliseconds long). Each of those chunks is passed through various computational filters in order to remove redundancy and to obtain a vector of values that describe the essence of the audio sample. Those vectors are then used to create a Markov chain for each phoneme in the audio stream. Since words are comprised of one or more phonemes, we can then match up sequences of phonemes with sequences of words (in essence, each phoneme is described by a hidden Markov model, and each word is a hidden Markov model of those HMMs).

All this doesn't arrive by magic – the speech recogniser has to be trained. Usually this requires the user to narrate some prepared text so that the HMMs are tuned to that particular person's accents, inflections and so on.

Despite all these algorithms and requirements for training, speech recognisers

Gambler's ruin

One of the properties of a drunkard's walk (see 'Random walks', left) is that if you let the Markov chain run for a long time, you'll hit every point on the number line, infinitely often.

So, let's suppose I'm a gambler with limited funds available. I'm playing a fair game in a casino whose bank has, essentially, unlimited funds. As I play, my pot of money will perform a drunkard's walk (I'll randomly lose some plays and my pot will decrement, and I'll randomly win others to increment my pot). At some point, it's guaranteed that my pot will hit zero and be wiped out. This is what's known as gambler's ruin.

Back in my day, I remember someone proposing the following way to play roulette. Place £1 on red. If you lose, place £2 on red the next game. As long as you're in a losing streak, continue to double your bet on red until you win. Then go back to the original £1 bet and start over. Providing we ignore the possibility of 0 and 00, when you win you will have increased your pot by £1 from the start of that particular losing streak. Of course, the drawback of this particular scheme is that your bets increase exponentially and you are more and more likely to be wiped out. ■

aren't that accurate – I've given up on the phone voice dialler in my Audi, for example. Such systems generally achieve around 80 per cent accuracy, which increases the more limited the word space is. Phone answering systems tend to be very accurate, because the user is speaking digits and words such as 'representative' or 'operator'. Accuracy also improves as the words used become more specialised, and as the recogniser is given more training. Nevertheless, speech recognition is improving all the time as the algorithms and hidden Markov models are refined and tuned. Markov chains are everywhere. **PCP**

*Julian M Bucknall has worked for companies ranging from TurboPower to Microsoft, and is now CTO for Developer Express.
feedback@pcplus.co.uk*