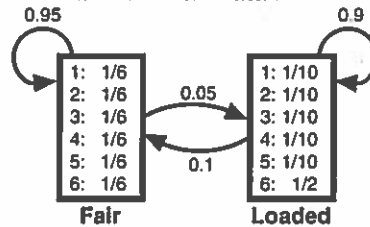


Example: The occasionally dishonest casino, part 1

Let us consider an example from Chapter 1. In a casino they use a fair die most of the time, but occasionally they switch to a loaded die. The loaded die has probability 0.5 of a six and probability 0.1 for the numbers one to five. Assume that the casino switches from a fair to a loaded die with probability 0.05 before each roll, and that the probability of switching back is 0.1. Then the switch between dice is a Markov process. In each state of the Markov process the outcomes of a roll have different probabilities, and thus the whole process is an example of a hidden Markov model. We can draw it like this:



where the emission probabilities $e()$ are shown in the state boxes. □

What is hidden in the above model? If you can just see a sequence of rolls (the sequence of observations) you do not know which rolls used a loaded die and which used a fair one, because that is kept secret by the casino; that is, the *state sequence is hidden*. In a Markov chain you always know exactly in which state a given observation belongs. Obviously the casino wouldn't tell you that they use loaded dice and what the various probabilities are. Yet for this more complicated situation, which we will return to later, it is possible to estimate the probabilities in the above HMM (once you have a suspicion that they use two different dice).

The reason for the name *emission* probabilities is that it is often convenient to think of HMMs as generative models, that generate or emit sequences. For instance we can generate random sequences of rolls from the model of the fair/-loaded dice above by simulating the successive choices of die, then rolls of the chosen die. More generally a sequence can be generated from an HMM as follows: First a state π_1 is chosen according to the probabilities a_{0i} . In that state an observation is emitted according to the distribution e_{π_1} for that state. Then a new state π_2 is chosen according to the transition probabilities $a_{\pi_1 i}$ and so forth. This way a sequence of random, artificial observations are generated. Therefore, we will sometimes say things like $P(x)$ is the probability that x was generated by the model.

It is now easy to write down the joint probability of an observed sequence x and a state sequence π :

$$P(x, \pi) = a_{0\pi_1} \prod_{i=1}^L e_{\pi_i}(x_i) a_{\pi_i \pi_{i+1}}, \quad (3.6)$$

where we require $\pi_{L+1} = 0$. For example, the probability of sequence CGCG being emitted by the state sequence (C_+, G_-, C_-, G_+) in our model is

$$a_{0,C_+} \times 1 \times a_{C_+,G_-} \times 1 \times a_{G_-,C_-} \times 1 \times a_{C_-,G_+} \times 1 \times a_{C_-,0}.$$

Equation (3.6) is the HMM analogue of equation (3.2). However, it is not so useful in practice because in general we do not know the path. In the following sections we describe how to estimate the path, either by finding the most likely one, or alternatively by using an *a posteriori* distribution over states. Then we go on to show how to estimate the parameters for an HMM.

Most probable state path: the Viterbi algorithm

Although it is no longer possible to tell what state the system is in by looking at the corresponding symbol, it is often the sequence of underlying states that we are interested in. To find out what the observation sequence 'means' by considering the underlying states is called *decoding* in the jargon of speech recognition. There are several approaches to decoding. Here we will describe the most common one, called the Viterbi algorithm. It is a dynamic programming algorithm closely related to the ones covered in Chapter 2.

In general there may now be many state sequences that could give rise to any particular sequence of symbols. For example, in our CpG model the state sequences (C_+, G_+, C_+, G_+) , (C_-, G_-, C_-, G_-) and (C_+, G_-, C_+, G_-) would all generate the symbol sequence CGCG. However, they do so with very different probabilities. The third is the product of multiple small probabilities of switching back and forth between the components, and hence is much smaller than the first two. The second is itself significantly smaller than the first because it contains two C to G transitions which are significantly less probable in the '-' component than in the '+' component. Of these three choices, therefore, it is most likely that the sequence CGCG came from a set of '+' states.

A predicted path through the HMM will tell us which part of the sequence is predicted as a CpG island, because we assumed above that each state was assigned to model either CpG islands or other regions. If we are to choose just one path for our prediction, perhaps the one with the highest probability should be chosen,

$$\pi^* = \underset{\pi}{\operatorname{argmax}} P(x, \pi). \quad (3.7)$$

The most probable path π^* can be found recursively. Suppose the probability $v_k(i)$ of the most probable path ending in state k with observation i is known for all the states k . Then these probabilities can be calculated for observation x_{i+1} as

$$v_l(i+1) = e_l(x_{i+1}) \max_k (v_k(i) a_{kl}). \quad (3.8)$$

All sequences have to start in state 0 (the begin state), so the initial condition is

(3.6)

v		C	G	C	G
B	1	0	0	0	0
A_+	0	0	0	0	0
C_+	0	0.13	0	0.012	0
G_+	0	0	0.034	0	0.0032
T_+	0	0	0	0	0
A_-	0	0	0	0	0
C_-	0	0.13	0	0.0026	0
G_-	0	0	0.010	0	0.00021
T_-	0	0	0	0	0

Figure 3.4 For the model of CpG islands shown in Figure 3.3 and the sequence CGCG, this is the resulting table of v . The most probable path is shown with bold face.

that $v_0(0) = 1$. By keeping pointers backwards, the actual state sequence can be found by backtracking. The full algorithm is:

Algorithm: Viterbi

Initialisation ($i = 0$): $v_0(0) = 1, v_k(0) = 0$ for $k > 0$.

Recursion ($i = 1 \dots L$): $v_i(i) = e_i(x_i) \max_k (v_k(i-1)a_{ki})$;
 $\text{ptr}_i(i) = \arg \max_k (v_k(i-1)a_{ki})$.

Termination: $P(x, \pi^*) = \max_k (v_k(L)a_{k0})$;
 $\pi_L^* = \arg \max_k (v_k(L)a_{k0})$.

Traceback ($i = L \dots 1$): $\pi_{i-1}^* = \text{ptr}_i(\pi_i^*)$. ◁

Note that an end state is assumed, which is the reason for a_{k0} in the termination step. If ends are not modelled, this a will disappear.

There are some implementational issues both for the Viterbi algorithm and the algorithms described later. The most severe practical problem is that multiplying many probabilities always yields very small numbers that will give underflow errors on any computer. For this reason the Viterbi algorithm should always be done in log space, i.e. calculating $\log(v_i(i))$, which will make the products become sums and the numbers stay reasonable. This is discussed in Section 3.6.

Figure 3.4 shows the full table of values of v for the sequence CGCG and the CpG island model. When we apply the same algorithm to a longer sequence the derived optimal path π^* will switch between the '+' and the '-' components of the model, and thereby give the precise boundaries of the predicted CpG island regions.

Example: The occasionally dishonest casino, part 2

For a sequence of dice rolls we can now find the most probable path through the model shown on p. 54. A total of 300 random rolls were generated from

Rolls	315116246446644245311321631164152133625144543631656626566666
Die	FFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFF
Viterbi	FFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFF
Rolls	651166453132651245636664631636663162326455236266666625151631
Die	LLLLLLFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFF
Viterbi	LLLLLLFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFF
Rolls	222555441666566563564324364131513465146353411126414626253356
Die	FFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFF
Viterbi	FFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFF
Rolls	36616366646623253441366166116325256246225526525226643535336
Die	LLLLLLLLFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFF
Viterbi	LLLLLLLLLLLLFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFF
Rolls	233121625364414432335163243633665562466662632666612355245242
Die	FFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFF
Viterbi	FFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFF

Figure 3.5 The numbers show 300 rolls of a die as described in the example. Below is shown which die was actually used for that roll (F for fair and L for loaded). Under that the prediction by the Viterbi algorithm is shown.

the model as described earlier. Each roll was generated either with the fair die (F) or the loaded one (L), as shown below the outcome of the roll in Figure 3.5. The Viterbi algorithm was used to predict the state sequence, i.e. which die was used for each of the rolls. Generally, as you can see, the Viterbi algorithm has recovered the state sequence fairly well. $\square$

Exercise

3.4 Show that $\pi^* = \underset{\pi}{\operatorname{argmax}} P(\pi|x)$ is equivalent to (3.7).

The forward algorithm

For Markov chains we calculated the probability of a sequence, $P(x)$, with equation (3.2). The resulting values were used to distinguish between CpG islands and other DNA for instance. We want to be able to calculate this probability for an HMM as well. Because many different state paths can give rise to the same sequence x , we must add the probabilities for all possible paths to obtain the full probability of x ,

$$P(x) = \sum_{\pi} P(x, \pi). \quad (3.9)$$

The number of possible paths π increases exponentially with the length of the sequence, so brute force evaluation of (3.9) by enumerating all paths is not practical. One approach is to use equation (3.6) evaluated at the most probable path π^* obtained in the last section as an approximation to $P(x)$. This implicitly assumes that the only path with significant probability is π^* , a somewhat startling

assumption which however in many cases is surprisingly good. In fact the approximation is unnecessary, because the full probability can itself be calculated by a similar dynamic programming procedure to the Viterbi algorithm, replacing the maximisation steps with sums. This is called the *forward* algorithm.

The quantity corresponding to the Viterbi variable $v_k(i)$ in the forward algorithm is

$$f_k(i) = P(x_1 \dots x_i, \pi_i = k), \quad (3.10)$$

which is the probability of the observed sequence up to and including x_i , requiring that $\pi_i = k$. The recursion equation is

$$f_i(i+1) = e_i(x_{i+1}) \sum_k f_k(i) a_{ki}. \quad (3.11)$$

The full algorithm is:

Algorithm: Forward algorithm

Initialisation ($i = 0$): $f_0(0) = 1$, $f_k(0) = 0$ for $k > 0$.

Recursion ($i = 1 \dots L$): $f_i(i) = e_i(x_i) \sum_k f_k(i-1) a_{ki}$.

Termination: $P(x) = \sum_k f_k(L) a_{k0}$. ◁

Like the Viterbi algorithm, the forward algorithm (and the backward algorithm in the next section) can give underflow errors when implemented on a computer. Again this can be solved by working in log space, although not as elegantly as for Viterbi. Alternatively a scaling method can be used. Both approaches are described in Section 3.6.

As well as their use in the forward algorithm, the quantities $f_k(i)$ have a number of other uses, including those described in the next two sections.

The backward algorithm and posterior state probabilities

The Viterbi algorithm finds the most probable path through the model, but as we remarked at the time, this may not always be the most appropriate basis for further inference about the sequence. We might for instance want to know what the most probable state is for an observation x_i . More generally, we may want the probability that observation x_i came from state k given the observed sequence, i.e. $P(\pi_i = k | x)$. This is the posterior probability of state k at time i when the emitted sequence is known.

Our approach to the posterior probability is a little indirect. We first calculate the probability of producing the entire observed sequence with the i th symbol

being produced by state k :

$$\begin{aligned} P(x, \pi_i = k) &= P(x_1 \dots x_i, \pi_i = k) P(x_{i+1} \dots x_L | x_1 \dots x_i, \pi_i = k) \\ &= P(x_1 \dots x_i, \pi_i = k) P(x_{i+1} \dots x_L | \pi_i = k), \end{aligned} \quad (3.12)$$

the second row following because everything after k only depends on the state at k . The first term in this is recognised as $f_k(i)$ from (3.10) that was calculated by the forward algorithm of the previous section. The second term is called $b_k(i)$,

$$b_k(i) = P(x_{i+1} \dots x_L | \pi_i = k). \quad (3.13)$$

It is analogous to the forward variable, but instead obtained by a backward recursion starting at the end of the sequence:

Algorithm: Backward algorithm

Initialisation ($i = L$): $b_k(L) = a_{k0}$ for all k .

Recursion ($i = L - 1, \dots, 1$): $b_k(i) = \sum_l a_{kl} e_l(x_{i+1}) b_l(i + 1)$.

Termination: $P(x) = \sum_l a_{0l} e_l(x_1) b_l(1)$.

The termination step is rarely needed, because $P(x)$ is usually found by the forward algorithm, and it is just shown for completeness.

Equation (3.12) can now be written as $P(x, \pi_i = k) = f_k(i) b_k(i)$, and from it we obtain the required posterior probabilities by straightforward conditioning,

$$P(\pi_i = k | x) = \frac{f_k(i) b_k(i)}{P(x)}, \quad (3.14)$$

where $P(x)$ is the result of the forward (or backward) calculation.

Example: The occasionally dishonest casino, part 3

In Figure 3.6 the posterior probability for the die being fair is shown for the sequence of rolls shown in Figure 3.5. Notice that the posterior probability does not reflect which die was actually used in some places. This is to be expected, simply because a misleading sequence of rolls can occur at random. $\square$

Posterior decoding

A major use of the $P(\pi_i = k | x)$ is for two alternative forms of decoding in addition to the Viterbi decoding we introduced in the previous section. These are particularly useful when many different paths have almost the same probability as the most probable one, because then it is not well justified to consider only the most probable path.