Machine Learning and Bayesian Inference

Dr Sean Holden

Computer Laboratory, Room FC06

Telephone extension 63725

Email: sbh11@cl.cam.ac.uk

www.cl.cam.ac.uk/∼sbh11/

Part IV

Inference through time

Hidden Markov models (HMMs)

Probabilistic reasoning through time

Probabilistic reasoning *through time*.

A fundamental idea throughout the AI courses has been that an agent should keep track of the *state of the environment*:

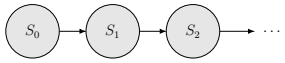
- The environment's state *changes over time*.
- The knowledge of *how the state changes* may be *uncertain*.
- The agent's *perception* of the state of the environment *may be uncertain*.

States and evidence

We model the (unobservable) state of the environment as follows:

- We use a *sequence*
$$(S_0, S_1, S_2, \ldots)$$
of *sets* of *random variables* (RVs).
- Each S_t is a *set* of RVs
$$S_t = \{S_t^{(1)}, \ldots, S_t^{(n)}\}$$
denoting the state of the environment at time t, where $t = 0, 1, 2, \ldots$.

Think of the state as changing over time.



States and evidence

At each time t there is also an *observable* set
$$E_t = \{E_t^{(1)}, \ldots, E_t^{(m)}\}$$
of random variables denoting the *evidence that an agent obtains about the state* at time t.

As usual capitals denote RVs and lower case denotes actual values. So actual values for the assorted RVs are denoted
$$S_t = \{s_t^{(1)}, \ldots, s_t^{(n)}\} = s_t$$
$$E_t = \{e_t^{(1)}, \ldots, e_t^{(m)}\} = e_t$$

Stationary and Markov processes

As t can in principle increase without bound we now need some simplifying assumptions.

1. *Assumption 1*: We deal with *stationary processes*—probability distributions do not change over time.
2. *Assumption 2*: We deal with *Markov processes*

$$\Pr(S_t|S_{0:t-1}) = \Pr(S_t|S_{t-1})$$

where $S_{0:t-1} = (S_0, S_1, \ldots, S_{t-1})$.

(Strictly speaking this is a *first order Markov Process*, and we'll only consider these.)

$\Pr(S_t|S_{t-1})$ is called the *transition model*.

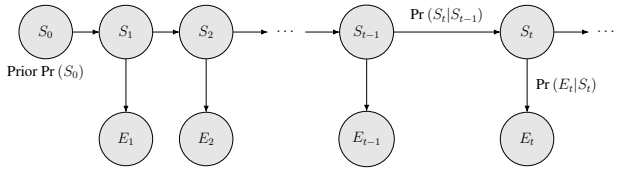
Stationary and Markov processes

Assumption 3: We assume that evidence only depends on the current state

$$\Pr(E_t|S_{0:t}, E_{1:t-1}) = \Pr(E_t|S_t)$$

Then

$\Pr(E_t|S_t)$ is called the *sensor model*.



$\Pr(S_0)$ is the *prior probability* of the starting state. We need this as there has to be some way of getting the process started.

The full joint distribution

Given:

1. The prior $\Pr(S_0)$.
2. The transition model $\Pr(S_t|S_{t-1})$.
3. The sensor model $\Pr(E_t|S_t)$.

along with the assumptions of stationarity and the assumptions of independence we have

$$\Pr(S_0, S_1, \ldots, S_t, E_1, E_2, \ldots, E_t) = \Pr(S_0) \prod_{i=1}^{t} \Pr(S_i|S_{i-1}) \Pr(E_i|S_i).$$

Example: 2008, paper 9, question 5

A friend of mine likes to climb on the roofs of Cambridge. To make a good start to the coming week, he climbs on a Sunday with probability 0.98. Being concerned for his own safety, he is less likely to climb today if he climbed yesterday, so

$$\Pr(\texttt{climb today}|\texttt{climb yesterday}) = 0.4.$$

If he did not climb yesterday then he is very unlikely to climb today, so

$$\Pr(\texttt{climb today}|\neg\texttt{climb yesterday}) = 0.1.$$

Unfortunately, he is not a very good climber, and is quite likely to injure himself if he goes climbing, so

$$\Pr(\texttt{injury}|\texttt{climb today}) = 0.8$$

whereas

$$\Pr(\texttt{injury}|\neg\texttt{climb today}) = 0.1.$$

Performing inference

There are *four basic inference tasks* that we might want to perform.

In each of the following cases, assume that *we have observed the evidence*

$$E_{1:t} = e_{1:t}.$$

Filtering: Deduce what state we might now be in by computing

$$\Pr(S_t|e_{1:t}).$$

Prediction: Deduce what state we might be in some time in the future by computing

$$\Pr(S_{t+T}|e_{1:t}) \text{ for some } T > 0.$$

Smoothing: Deduce what state we might have been in at some point in the past by computing

$$\Pr(S_t|e_{1:T}) \text{ for } 0 \leq t < T.$$

Find the most likely explanation: Deduce the most likely sequence of states so far by computing

$$\operatorname*{argmax}_{s_{1:t}} \Pr(s_{1:t}|e_{1:t}).$$

Filtering

We want to compute $\Pr(S_t|e_{1:t})$. This is often called the *forward message* and denoted

$$f_{1:t} = \Pr(S_t|e_{1:t})$$

for reasons that are about to become clear.

Remember that S_t is an RV and so $f_{1:t}$ is a *probability distribution* containing a probability for each possible value of S_t.

It turns out that this can be done in a simple manner with a *recursive estimation*. Obtain the result at time $t+1$:

1. using the result from time t and...
2. ...incorporating new evidence e_{t+1}.

$$f_{1:t+1} = g(e_{t+1}, f_{1:t})$$

for a suitable function g that we'll now derive.

Filtering

Step 1:

Project the current state distribution forward

$$\begin{aligned}\Pr(S_{t+1}|e_{1:t+1}) &= \Pr(S_{t+1}|e_{1:t}, e_{t+1}) \\ &= c\Pr(e_{t+1}|S_{t+1}, e_{1:t}) \Pr(S_{t+1}|e_{1:t}) \\ &= c\underbrace{\Pr(e_{t+1}|S_{t+1})}_{\text{Sensor model}}\underbrace{\Pr(S_{t+1}|e_{1:t})}_{\text{Needs more work}}\end{aligned}$$

where as usual c is a constant that normalises the distribution. Here,

- The first line does nothing but split $e_{1:t+1}$ into e_{t+1} and $e_{1:t}$.
- The second line is an application of *Bayes' theorem*.
- The third line uses *assumption 3* regarding sensor models.

Filtering

Step 2:

To obtain $\Pr(S_{t+1}|e_{1:t})$

$$\begin{aligned}\Pr(S_{t+1}|e_{1:t}) &= \sum_{s_t} \Pr(S_{t+1}, s_t|e_{1:t}) \\ &= \sum_{s_t} \Pr(S_{t+1}|s_t, e_{1:t}) \Pr(s_t|e_{1:t}) \\ &= \sum_{s_t} \underbrace{\Pr(S_{t+1}|s_t)}_{\text{Transition model}} \underbrace{\Pr((s_t|e_{1:t}))}_{\text{Available from previous step}}.\end{aligned}$$

Here,

- The first line uses marginalisation.
- The second line uses the basic equation $\Pr(A, B) = \Pr(A|B)\Pr(B)$.
- The third line uses *assumption 2* regarding transition models.

Filtering

Pulling it all together

$$\Pr(S_{t+1}|e_{1:t+1}) = c\underbrace{\Pr(e_{t+1}|S_{t+1})}_{\text{Sensor model}} \sum_{s_t} \underbrace{\Pr(S_{t+1}|s_t)}_{\text{Transition model}} \underbrace{\Pr(s_t|e_{1:t})}_{\text{From previous step}}.$$

This will be shortened to

$$f_{1:t+1} = c\text{FORWARD}(e_{t+1}, f_{1:t})$$

Here

- $f_{1:t}$ is a shorthand for $\Pr(S_t|e_{1:t})$.
- $f_{1:t}$ is often interpreted as a *message* being passed forward.
- The process is started using the *prior*.

Prediction

Prediction is somewhat simpler as

$$\begin{aligned}\underbrace{\Pr(S_{t+T+1}|e_{1:t})}_{\text{Prediction at } t+T+1} &= \sum_{s_{t+T}} \Pr(S_{t+T+1}, s_{t+T}|e_{1:t}) \\ &= \sum_{s_{t+T}} \Pr(S_{t+T+1}|s_{t+T}, e_{1:t}) \Pr(s_{t+T}|e_{1:t}) \\ &= \sum_{s_{t+T}} \underbrace{\Pr(S_{t+T+1}|s_{t+T})}_{\text{Transition model}} \underbrace{\Pr(s_{t+T}|e_{1:t})}_{\text{Prediction at } t+T}.\end{aligned}$$

However we do not get to make accurate predictions arbitrarily far into the future!

Smoothing

For smoothing, we want to calculate $\Pr(S_t|e_{1:T})$ for $0 \leq t < T$.

Again, we can do this in two steps.

Step 1:

$$\begin{aligned}\Pr(S_t|e_{1:T}) &= \Pr(S_t|e_{1:t}, e_{t+1:T}) \\ &= c\Pr(S_t|e_{1:t}) \Pr(e_{t+1:T}|S_t, e_{1:t}) \\ &= c\Pr(S_t|e_{1:t}) \Pr(e_{t+1:T}|S_t) \\ &= cf_{1:t}b_{t+1:T}.\end{aligned}$$

Here

- $f_{1:t}$ is the forward message defined earlier.
- $b_{t+1:T}$ is a shorthand for $\Pr(e_{t+1:T}|S_t)$ to be regarded as *a message being passed backward*.

Smoothing

Step 2:

$$\begin{aligned}b_{t+1:T} = \Pr(e_{t+1:T}|S_t) &= \sum_{s_{t+1}} \Pr(e_{t+1:T}, s_{t+1}|S_t) \\ &= \sum_{s_{t+1}} \Pr(e_{t+1:T}|s_{t+1}) \Pr(s_{t+1}|S_t) \\ &= \sum_{s_{t+1}} \Pr(e_{t+1}, e_{t+2:T}|s_{t+1}) \Pr(s_{t+1}|S_t) \\ &= \sum_{s_{t+1}} \underbrace{\Pr(e_{t+1}|s_{t+1})}_{\text{Sensor model}} \underbrace{\Pr(e_{t+2:T}|s_{t+1})}_{b_{t+2:T}} \underbrace{\Pr(s_{t+1}|S_t)}_{\text{Transition model}} \\ &= \text{BACKWARD}(e_{t+1:T}, b_{t+2:T}).\end{aligned}$$

This process is initialised with

$$b_{t+1:t} = \Pr(e_{T+1:T}|S_T) = (1, \ldots, 1)$$

The forward-backward algorithm

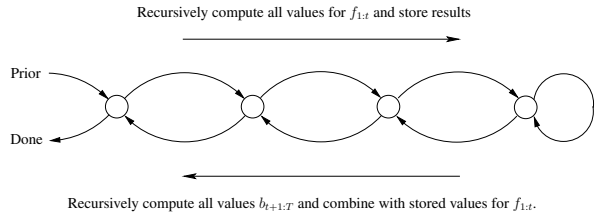
So: our original aim of computing $\Pr(S_t|e_{1:T})$ can be achieved using:

- A recursive process working from time 1 to time t.
- A recursive process working from time T to time $t+1$.

This results in a process that is $O(T)$ given the evidence $e_{1:T}$ and smooths for a *single* point at time t.

To smooth at *all* points $1:T$ we can easily repeat the process obtaining $O(T^2)$.

Alternatively a very simple example of *dynamic programming* allows us to smooth at all points in $O(T)$ time.



Computing the most likely sequence: the Viterbi algorithm

In *computing the most likely sequence* the aim is to obtain $\operatorname{argmax}_{s_{1:t}} \Pr(s_{1:t}|e_{1:t})$.

Earlier we derived the joint distribution for all relevant variables

$$\Pr(S_0, S_1, \ldots, S_t, E_1, E_2, \ldots, E_t) = \Pr(S_0) \prod_{i=1}^{t} \Pr(S_i|S_{i-1}) \Pr(E_i|S_i) .$$

We therefore have

$$\begin{aligned} &\boxed{\max_{s_{1:t}} \Pr(s_{1:t}, S_{t+1}|e_{1:t+1})} \\ &= c \max_{s_{1:t}} \Pr(e_{t+1}|S_{t+1}) \Pr(S_{t+1}|s_t) \Pr(s_{1:t}|e_{1:t}) \\ &= c\Pr(e_{t+1}|S_{t+1}) \max_{s_t} \left\{ \Pr(S_{t+1}|s_t) \boxed{\max_{s_{1:t-1}} \Pr(s_{1:t-1}, s_t|e_{1:t})} \right\} . \end{aligned}$$

This looks *a bit fierce*.

Computing the most likely sequence: the Viterbi algorithm

There is however a way to visualise it that leads to a *dynamic programming* algorithm called the *Viterbi algorithm*.

Step 1: Simplify the notation.

- Assume there are n states $s_1, \ldots, s_n$ and m possible observations $e_1, \ldots, e_m$ at any given time.
- Denote $\Pr(S_t = s_j|S_{t-1} = s_i)$ by $p_{i,j}(t)$.
- Denote $\Pr(e_t|S_t = s_i)$ by $q_i(t)$.

It's important to remember in what follows that the *observations are known* but that we're *maximising over all possible state sequences*.

Computing the most likely sequence: the Viterbi algorithm

The equation we're interested in is now of the form

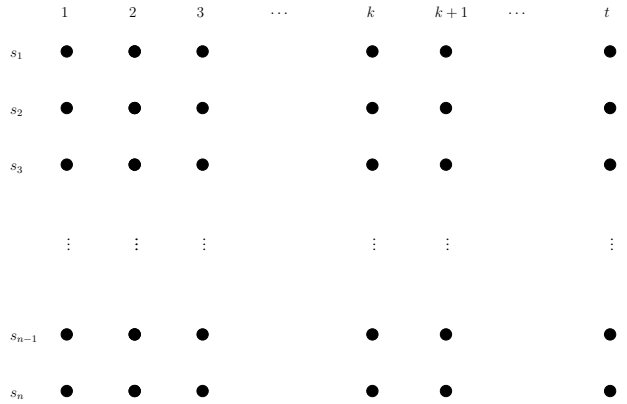
$$P = \prod_{t=1}^{T} p_{i,j}(t) q_i(t).$$

It is in fact a *function* of *any given sequence of states*.

(The prior $\Pr(S_0)$ has been dropped out for the sake of clarity, but is easy to put back in in what follows.)

Computing the most likely sequence: the Viterbi algorithm

Step 2: Make a grid: columns denote time and rows denote state.



Computing the most likely sequence: the Viterbi algorithm

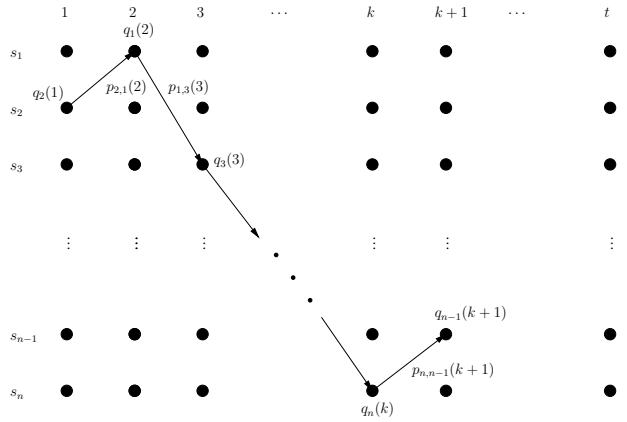
Step 3: Label the nodes:

- Say at time t the *actual observation* was e_t. Then label the node for s_i in column t with the value $q_i(t)$.
- Any *sequence of states* through time is now a *path through the grid*. So for any transition from s_i at time $t-1$ to s_j at time t label the transition with the value $p_{i,j}(t)$.

In the following diagrams we can often just write $p_{i,j}$ and q_i because the time is clear from the diagram.

So for instance...

Computing the most likely sequence: the Viterbi algorithm

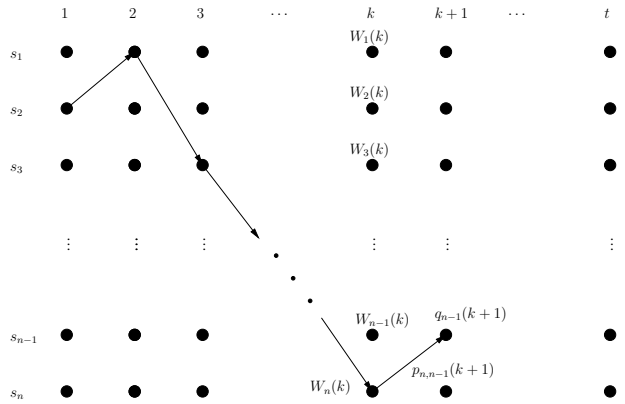


Computing the most likely sequence: the Viterbi algorithm

- The value of $P = \prod_{t=1}^{T} p_{i,j}(t)q_i(t)$ for any path through the grid is just the product of the corresponding labels that have been added.
- But we don't want to find the maximum by looking at all the possible paths because this would be time-consuming.
- The *Viterbi algorithm* computes the maximum by moving from one column to the next updating as it goes.
- Say you're at column k and *for each node* m in that column you know the *highest value* for the product to this point over *any possible path*. Call this:

$$W_m(k) = \max_{s_{1:k}} \prod_{t=1}^{k} p_{i,j}(t)q_i(t).$$

Computing the most likely sequence: the Viterbi algorithm



Computing the most likely sequence: the Viterbi algorithm

Here is the key point: you only need to know

- The values $W_i(k)$ for $i = 1, \ldots, n$ at time k.
- The numbers $p_{i,j}(k+1)$.
- The numbers $q_i(k+1)$.

to compute the values $W_i(k+1)$ for the next column $k+1$.

This is because

$$W_i(k+1) = \max_j W_j(k) p_{j,i}(k+1) q_i(k+1).$$

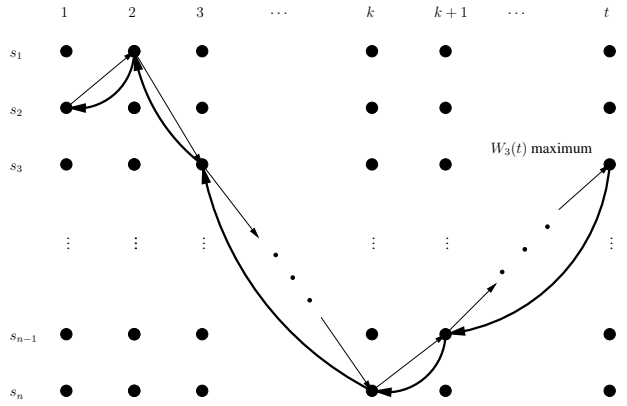
Computing the most likely sequence: the Viterbi algorithm

Once you get to the column for time t:

- The node with the largest value for $W_i(t)$ tells you the largest possible value of P.
- Provided you stored *the path taken to get there* you can *work backwards* to find *the corresponding sequence of states*.

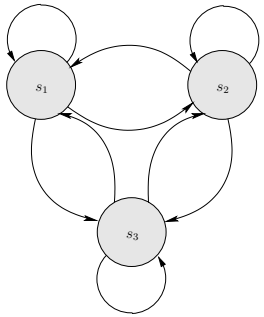
This is the *Viterbi algorithm*.

Computing the most likely sequence: the Viterbi algorithm



Hidden Markov models

Now for a specific case: hidden Markov models (HMMs). Here we have a *single*, *discrete* state variable S_i taking values $s_1, s_2, \ldots, s_n$. For example, with $n = 3$ we might have



	$\Pr(S_{t+1}\|S_t = s_1)$	$\Pr(S_{t+1}\|S_t = s_2)$	$\Pr(S_{t+1}\|S_t = s_3)$
s_1	0.3	0.2	0.2
s_2	0.1	0.6	0.3
s_3	0.6	0.2	0.5

Hidden Markov models

In this simplified case the conditional probabilities $\Pr(S_{t+1}|S_t)$ can be represented using the matrix

$$S_{ij} = \Pr(S_{t+1} = s_j | S_t = s_i)$$

or for the example on the previous slide

$$\mathbf{S} = \begin{bmatrix} 0.3 & 0.1 & 0.6 \\ 0.2 & 0.6 & 0.2 \\ 0.2 & 0.3 & 0.5 \end{bmatrix} = \begin{bmatrix} \Pr(s_1|s_1) & \Pr(s_2|s_1) & \cdots & \Pr(s_n|s_1) \\ \Pr(s_1|s_2) & \Pr(s_2|s_2) & \cdots & \Pr(s_n|s_2) \\ \vdots & \vdots & \ddots & \vdots \\ \Pr(s_1|s_n) & \Pr(s_2|s_n) & \cdots & \Pr(s_n|s_n) \end{bmatrix}.$$

To save space, I am abbreviating $\Pr(S_{t+1} = s_i | S_t = s_j)$ to $\Pr(s_i|s_j)$.

Hidden Markov models

The computations we're making are always conditional on some actual observations $e_{1:T}$.

For each t we can therefore use the sensor model to define a further matrix $\mathbf{E}_t$:

- $\mathbf{E}_t$ is square and diagonal (all off-diagonal elements are 0).
- The ith element of the diagonal is $\Pr(e_t|S_t = s_i)$.

So in our present example with 3 states, there will be a matrix

$$\mathbf{E}_t = \begin{bmatrix} \Pr(e_t|s_1) & 0 & 0 \\ 0 & \Pr(e_t|s_2) & 0 \\ 0 & 0 & \Pr(e_t|s_3) \end{bmatrix}$$

for each $t = 1, \ldots, T$.

Hidden Markov models

In the general case the equation for filtering was

$$\Pr(S_{t+1}|e_{1:t+1}) = c\Pr(e_{t+1}|S_{t+1}) \sum_{s_t} \Pr(S_{t+1}|s_t) \Pr(s_t|e_{1:t})$$

and the message $f_{1:t}$ was introduced as a representation of $\Pr(S_t|e_{1:t})$.

In the present case we can define $f_{1:t}$ to be the vector

$$f_{1:t} = \begin{bmatrix} \Pr(s_1|e_{1:t}) \\ \Pr(s_2|e_{1:t}) \\ \vdots \\ \Pr(s_n|e_{1:t}) \end{bmatrix}.$$

Key point: *the filtering equation now reduces to nothing but matrix multiplication*.

What does matrix multiplication do?

What does matrix multiplication do? *It computes weighted summations*:

$$\mathbf{Ab} = \begin{bmatrix} a_{1,1} & a_{1,2} & \cdots & a_{1,m} \\ a_{2,1} & a_{2,2} & \cdots & a_{2,m} \\ \vdots & \vdots & \ddots & \vdots \\ a_{n,1} & a_{n,2} & \cdots & a_{n,m} \end{bmatrix} \begin{bmatrix} b_1 \\ b_2 \\ \vdots \\ b_m \end{bmatrix} = \begin{bmatrix} \sum_{i=1}^m a_{1,i} b_i \\ \sum_{i=1}^m a_{2,i} b_i \\ \vdots \\ \sum_{i=1}^m a_{n,i} b_i \end{bmatrix}.$$

So the point at the end of the last slide shouldn't come as a big surprise!

Hidden Markov models

Now, note that if we have n states

$$\begin{aligned} \mathbf{S}^T f_{1:t} &= \begin{bmatrix} \Pr(s_1|s_1) & \cdots & \Pr(s_1|s_n) \\ \Pr(s_2|s_1) & \cdots & \Pr(s_2|s_n) \\ \vdots & \ddots & \vdots \\ \Pr(s_n|s_1) & \cdots & \Pr(s_n|s_n) \end{bmatrix} \begin{bmatrix} \Pr(s_1|e_{1:t}) \\ \Pr(s_2|e_{1:t}) \\ \vdots \\ \Pr(s_n|e_{1:t}) \end{bmatrix} \\ &= \begin{bmatrix} \Pr(s_1|s_1)\Pr(s_1|e_{1:t}) + \cdots + \Pr(s_1|s_n)\Pr(s_n|e_{1:t}) \\ \Pr(s_2|s_1)\Pr(s_1|e_{1:t}) + \cdots + \Pr(s_2|s_n)\Pr(s_n|e_{1:t}) \\ \vdots \\ \Pr(s_n|s_1)\Pr(s_1|e_{1:t}) + \cdots + \Pr(s_n|s_n)\Pr(s_n|e_{1:t}) \end{bmatrix} \\ &= \begin{bmatrix} \sum_s \Pr(s_1|s)\Pr(s|e_{1:t}) \\ \sum_s \Pr(s_2|s)\Pr(s|e_{1:t}) \\ \vdots \\ \sum_s \Pr(s_n|s)\Pr(s|e_{1:t}) \end{bmatrix}. \end{aligned}$$

Hidden Markov models

And taking things one step further

$$\begin{aligned} \mathbf{E}_{t+1}\mathbf{S}^T f_{1:t} &= \begin{bmatrix} \Pr(e_{t+1}|s_1) & & 0 \\ & \ddots & \\ 0 & & \Pr(e_{t+1}|s_n) \end{bmatrix} \begin{bmatrix} \sum_s \Pr(s_1|s)\Pr((s|e_{1:t})) \\ \sum_s \Pr(s_2|s)\Pr((s|e_{1:t})) \\ \vdots \\ \sum_s \Pr(s_n|s))\Pr(s|e_{1:t}) \end{bmatrix} \\ &= \begin{bmatrix} \Pr(e_{t+1}|s_1)\sum_s \Pr(s_1|s)\Pr(s|e_{1:t})) \\ \Pr(e_{t+1}|s_2)\sum_s \Pr(s_2|s)\Pr(s|e_{1:t}) \\ \vdots \\ \Pr(e_{t+1}|s_n)\sum_s \Pr(s_n|s)\Pr(s|e_{1:t}) \end{bmatrix}. \end{aligned}$$

Compare this with the equation for filtering

$$\Pr(S_{t+1}|e_{1:t+1}) = c\Pr(e_{t+1}|S_{t+1}) \sum_{s_t} \Pr(S_{t+1}|s_t)\Pr(s_t|e_{1:t}).$$

Hidden Markov models

Comparing the expression for $\mathbf{E}_{t+1}\mathbf{S}^T f_{1:t}$ with the equation for filtering we see that

$$f_{1:t+1} = c\mathbf{E}_{t+1}\mathbf{S}^T f_{1:t}$$

and a similar equation can be found for b

$$b_{t+1:T} = \mathbf{S}\mathbf{E}_{t+1} b_{t+2:T}.$$

Exercise: derive this.

The fact that these can be expressed simply using only multiplication of vectors and matrices allows us to make an improvement to the forward-backward algorithm.

Hidden Markov models

The *forward-backward* algorithm works by:

- Moving up the sequence from 1 to T, computing and storing values for f.
- Moving down the sequence from T to 1 computing values for b and *combining* them with the stored values for f using the equation

$$\Pr(S_t|e_{1:T}) = cf_{1:t}b_{t+1:T}.$$

Now in our simplified HMM case we have

$$f_{1:t+1} = c\mathbf{E}_{t+1}\mathbf{S}^T f_{1:t}$$

or multiplying through by $(\mathbf{E}_{t+1}\mathbf{S}^T)^{-1}$ and re-arranging

$$f_{1:t} = \frac{1}{c}(\mathbf{S}^T)^{-1}(\mathbf{E}_{t+1})^{-1} f_{1:t+1}.$$

Hidden Markov models

So as long as:

- We know the *final* value for f.
- $\mathbf{S}^T$ has an inverse.
- Every observation has non-zero probability in every state.

We *don't* have to store T different values for f—we just work through, discarding intermediate values, to obtain the last value and then work backward.