Hidden Markov Model – The Forward Algorithm

A poker player can be in one of three states: Honest, Cunning or Liar. Those are known as the “hidden states”.

The player makes a statement and switches to another state based on the following transition matrix.

	H	C	L
H	50%	40%	10%
C	10%	80%	10%
L	10%	60%	30%

The statement can be either True or False – the probability depends on the state

	T	F
H	90%	10%
C	50%	50%
L	10%	90%

The Forward algorithm answer the question: Given a model what is the probability of a specific sequence.

For example: what is the probability of FFF?

While it is the “forward algorithm” and indeed is evaluated in the “forward” direction it is easier to understand the principle if we think of the “backward” direction, just like local string matching.

H			P_3 (H)
C			P_3 (C)
L			P_3 (L)
	F	F	F

P_3 (H)_ probability that the model generated FFF and ended up in state H

P_3 (C)_ probability that the model generated FFF and ended up in state C

	T	F
H	90%	10%
C	50%	50%
L	10%	90%

H			P_3 (H)
C			P_3 (C)
L			P_3 (L)
	F	F	F

The probability of generating FFF is

$P_3(H) + P_3(C) + P_3(L)$

	H	C	L
H	50%	40%	10%
C	10%	80%	10%
L	10%	60%	30%

	T	F
H	90%	10%
C	50%	50%
L	10%	90%

H		P_2 (H)	P_3 (H)
C		P_2 (C)	P_3 (C)
L		P_2 (L)	P_3 (L)
	F	F	F

$$P_3(H) = P_2(H) * 0.5 * 0.1 + P_2(C) * 0.1 * 0.1 + P_2(L) * 0.1 * .1$$

	H	C	L
H	50%	40%	10%
C	10%	80%	10%
L	10%	60%	30%

	T	F
H	90%	10%
C	50%	50%
L	10%	90%

H	0.05		
C	0.125		
L	0.225		
	F	F	F

Fill first column using the initial distribution and emission probabilities,

0.05 = 0.5 * 0.1
0.125 = 0.25 * 0.5

	H	C	L
H	50%	40%	10%
C	10%	80%	10%
L	10%	60%	30%

	T	F
H	90%	10%
C	50%	50%
L	10%	90%

H	0.05	0.006	0.00234
C	0.125	0.1275	0.07515
L	0.225	0.0765	0.03267
	F	F	F

0.006 = 0.1 * (0.05 * 0.5 + 0.125 * 0.1 + 0.225 * 0.1)
0.1275 = .5 * (0.05 * .4 + 0.125 * .8 + .225 * .6)

0.11 = 0.00234 + 0.07515 + 0.03267

Forward probabilities for a 2000 states sequence

0	1	2	3	4	5	6	7	8	9
0.01670	0.04950	0.00433	0.01150	0.00100	0.00029	0.00014	0.00065	0.00006	0.00002
0.35400	0.17900	0.08350	0.04100	0.01920	0.00941	0.00478	0.00246	0.00115	0.00056
0.11300	0.00708	0.02250	0.00155	0.00514	0.00320	0.00174	0.00010	0.00031	0.00019

What do you notice here?

It gets much worse later on ...

1075	1076								
3.46e-323	4.94e-324	4.94e-324	0	0	0	0	0	0	0
5.93e-323	2.96e-323	1.48e-323	4.94e-324	0	0	0	0	0	0
0	9.88e-324	0	0	0	0	0	0	0	0

1075	1076								
3.46e-323	4.94e-324	4.94e-324	0	0	0	0	0	0	0
5.93e-323	2.96e-323	1.48e-323	4.94e-324	0	0	0	0	0	0
0	9.88e-324	0	0	0	0	0	0	0	0

The probability of any specific sequence is very low. Remember that even with the minimal alphabet size of 2 there are 2^{2000} possible sequences of length 2000.

This is where we hit the limits of standard floating point representation in a modern computer.

How do computers represent floating point numbers?

As $M \times 2^E$, where M is the mantissa and E is the exponent.

Even with double precision (64 bits) the exponent is typically only 11 bits, which gives a range of $[2^{-1022}, 2^{1023}]$, approximately $10^{-308,}\ 10^{+308}$

So, what can we do?

The standard trick is to transform the numbers and work in the LOG scale.

That way we can represent very small or very large numbers.

Instead of multiplication/division we use addition/subtraction.

But we still need to add those numbers!

	H	C	L
H	50%	40%	10%
C	10%	80%	10%
L	10%	60%	30%

	T	F
H	90%	10%
C	50%	50%
L	10%	90%

H	-2.995		
C	-2.079	-2.059	
L	-1.491		
	F	F	F

Table entries are log(p).
log(0.05) = -2.995

This is how we compute normally

0.1275 = .5 * (0.05 * .4 + 0.125 * .8 + .225 * .6)

Now we need

Log(.5) + [(log(0.05) + log(0.4)) ♣ (log(0.125) + log(.8)) ♣ (log(0.225) + log(.6))

The club sign means addition in log space, where the answer is in log units -

a ♣ b = log(exp(a) + exp(b))

$$a \clubsuit b = \log(e^a + e^b) = \log(e^a) + \log(e^0 + e^{b-a}) =$$
$$a + \log(1 + e^{b-a})$$

So, if a and b are of similar magnitude the loss of accuracy is small.

We pay a price in computation time, 3 +-, one log and one exponentiation instead of one addition.

The backward probabilities

H		P_2 (H)	
C	P_1(C)	P_2 (C)	
L		P_2 (L)	
	F	F	F

P_2(H) = probability of generating the last F given that state is H at time 2

P_1(C) = probability of generating FF given that state is C at time 1

	H	C	L
H	50%	40%	10%
C	10%	80%	10%
L	10%	60%	30%

	T	F
H	90%	10%
C	50%	50%
L	10%	90%

H			1
C			1
L			1
	F	F	F

	H	C	L
H	50%	40%	10%
C	10%	80%	10%
L	10%	60%	30%

	T	F
H	90%	10%
C	50%	50%
L	10%	90%

H	0.16	0.34	1
C	0.2556	0.5	1
L	0.31	0.58	1
	F	F	F

0.34 = .5 * .1 * 1 + .4 * .5 * 1 + .1 * .9 * 1

0.31 = .1 * .1 * .34 + .6 * .5 * .5 + .3 * .9 * .58

	H	C	L
H	50%	40%	10%
C	10%	80%	10%
L	10%	60%	30%

	T	F
H	90%	10%
C	50%	50%
L	10%	90%

H	0.16	0.34	1
C	0.2556	0.5	1
L	0.31	0.58	1
	F	F	F

0.11 = .5 * .1 * 0.16 + .25 * .5 * 0.2556 + .25 * .9 * .31

For the overall probability multiply by initial distribution and emission for first output and sum over that. We should get the same value!

The “decoding” - Viterbi algorithm

What is the most probable sequence of internal states for a given HMM and an output sequence?

The “decoding” - Viterbi algorithm

What is the most probable sequence of internal states for a given HMM and an output sequence?

The Joint probability – probability of generating a specific output using a specific sequence of internal states.

$$P(\text{Output},\text{State} \mid \text{HMM})$$

We look for the sequence of states which maximizes the above.

The forward/backward probabilities were used to calculate the sum of the joint probability over all possible states.

	H	C	L
H	50%	40%	10%
C	10%	80%	10%
L	10%	60%	30%

	T	F
H	90%	10%
C	50%	50%
L	10%	90%

For example,

$P([T,F,T] , [H,C,L] \mid HMM) = 0.5 * 0.9 * 0.4 * 0.5 * 0.1 * 0.1$

$\pi_a \quad E(H,T) \quad P(H \rightarrow C) \quad E(C,F) \quad ...$

Finding the most likely decoding is very similar to how we compute the forward probabilities. The added nuisance is keeping track of the internal states.

H			P_3 (H)
C			P_3 (C)
L			P_3 (L)
	F	F	F

P_3(H) is the highest joint probability possible when ending on state H.

The filling of the table is similar to the forward probabilities, only we take the maximum instead of the sum.

	H	C	L
H	50%	40%	10%
C	10%	80%	10%
L	10%	60%	30%

	T	F
H	90%	10%
C	50%	50%
L	10%	90%

H	0.05		
C	0.125		
L	0.225		
	F	F	F

Start the same way as for forward – start probability times emission.

	H	C	L
H	50%	40%	10%
C	10%	80%	10%
L	10%	60%	30%

	T	F
H	90%	10%
C	50%	50%
L	10%	90%

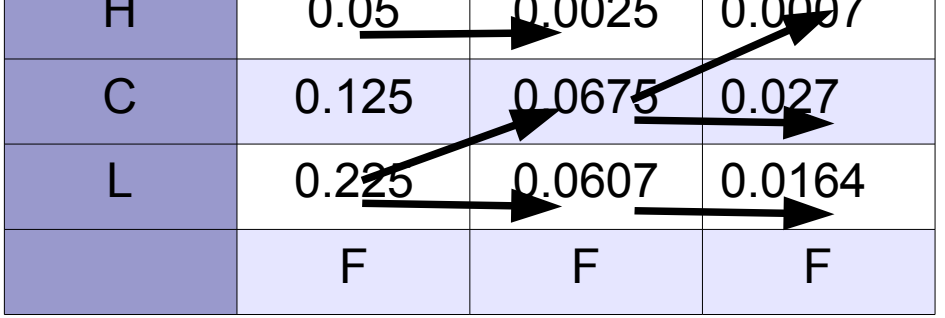
H	0.05	0.0025	0.0007
C	0.125	0.0675	0.027
L	0.225	0.0607	0.0164
	F	F	F

Take $P_2(C)$ – We can get there from H (0.05*0.4), from C (0.125 * .8) or from L (0.225 * .6). Highest is (0.135=0.225*.6), so we fill the cell with 0.135 * .5 (.5 emission of F in state C)

	H	C	L
H	50%	40%	10%
C	10%	80%	10%
L	10%	60%	30%

	T	F
H	90%	10%
C	50%	50%
L	10%	90%

H	0.05	0.0025	0.0007
C	0.125	0.0675	0.027
L	0.225	0.0607	0.0164
	F	F	F



The most likely internal state is (L,C,C) with a joint probability of 0.027

Baum-Welsh: Training an HMM

What does training (or learning) means?

Baum-Welsh: Training an HMM

What does training (or learning) means?

We have a model and we wish to estimate its parameters based on actual data.

Unfortunately, unlike the previous cases it is not possible to provide a small, easy to understand example of training.

Why?

We need enough data to estimate the parameters. In our very small example we have 11 (2 + 6 + 3) parameters.

So, we need long runs.

To estimate the start probability we need multiple runs.

And the calculations are tedious.

Still, I was hoping to show the estimates used in the Baum-Welsh method for the true model in our little example.

It turns out not all HMM's are created equal.

My arbitrary constructed example is bad in almost every way – the numbers are not even close to the real ones!

The transition probabilities matter.

	H	C	L
H	50%	40%	10%
C	10%	80%	10%
L	10%	60%	30%

	T	F
H	90%	10%
C	50%	50%
L	10%	90%

Basically in this case the input does not matter!

That means no estimation is possible.

out	Joint probability	Viterbi path
TTTT	0.0410	HHHH
TTTF	0.0182	HHHC
TTFT	0.0162	HHCC
TTFF	0.0162	HHCC
TFTT	0.0144	HCCC
TFTF	0.0144	HCCC
TFFT	0.0144	HCCC
TFFF	0.0144	HCCC
FTTT	0.0108	LCCC
FTTF	0.0108	LCCC
FTFT	0.0108	LCCC
FTFF	0.0108	LCCC
FFTT	0.0108	LCCC
FFTF	0.0108	LCCC
FFFT	0.0108	LCCC
FFFF	0.0108	LCCC

We need to change the matrix to something like below,

	H	C	L
H	80%	10%	10%
C	10%	80%	10%
L	10%	10%	80%

	T	F
H	90%	10%
C	50%	50%
L	10%	90%

TTTT	0.1680	HHHH
TTTF	0.0210	HHHL
TTFT	0.0187	HHHH
TTFF	0.0210	HHLL
TFTT	0.0187	HHHH
TFTF	0.0080	CCCC
TFFT	0.0080	CCCC
TFFF	0.0210	HLLL
FTTT	0.0187	HHHH
FTTF	0.0080	CCCC
FTFT	0.0080	CCCC
FTFF	0.0093	LLLL
FFTT	0.0105	HLHH
FFTF	0.0093	LLLL
FFFT	0.0105	HLLH
FFFF	0.0840	LLLL