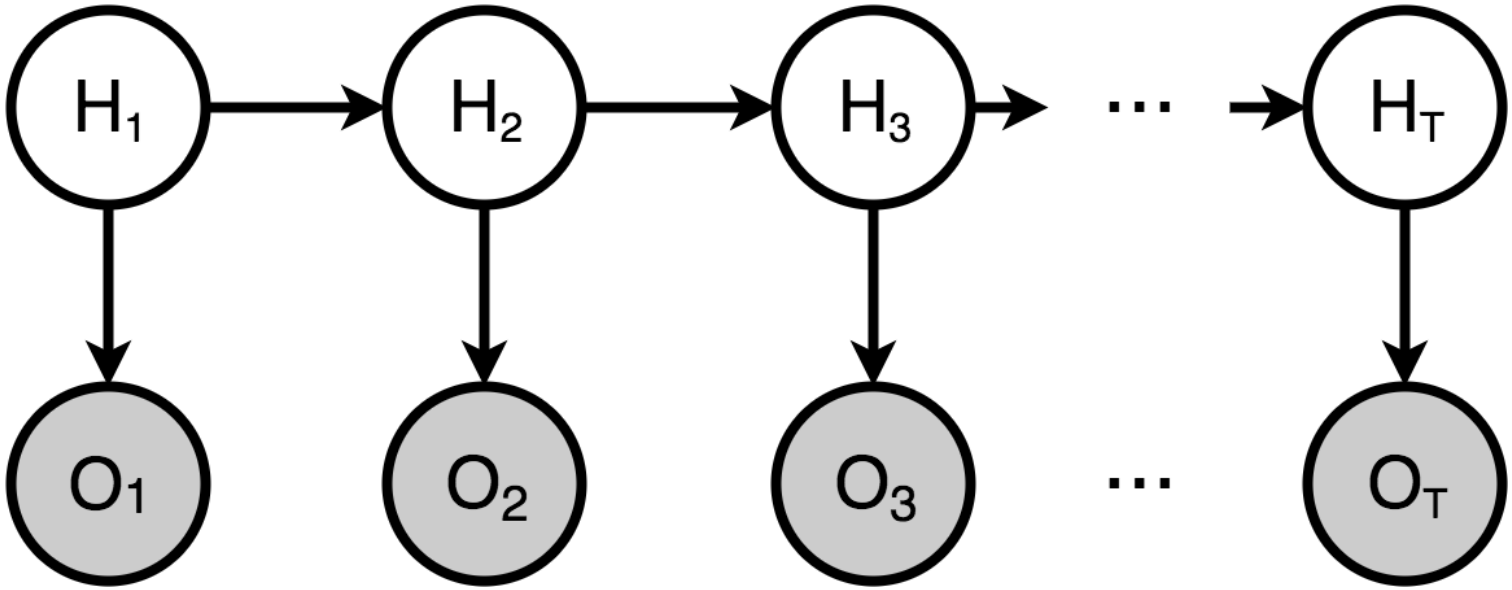
DSCI 552, Machine Learning for Data Science

University of Southern California

M. R. Rajati, PhD

Lesson 11
Hidden Markov Models



There's discrek # of states. There's a probability of moving to any other state. We forget where we've been, and only know where we are. This is a Markov chain.

Hidden Markov Models

What is a hidden Markov model (HMM)?

A machine learning technique and…

…a discrete hill climb technique

Two for the price of one!

Where are HMMs used?

Speech recognition, information security, and too many other things to list

Q: Why are HMMs so useful?

A: Widely applicable and ***efficient algorithms***

Markov Chain

Markov chain

- "Memoryless random process"
- Transitions depend only on current state (Markov chain of order 1)…
- …and transition probability matrix

Markov Chain

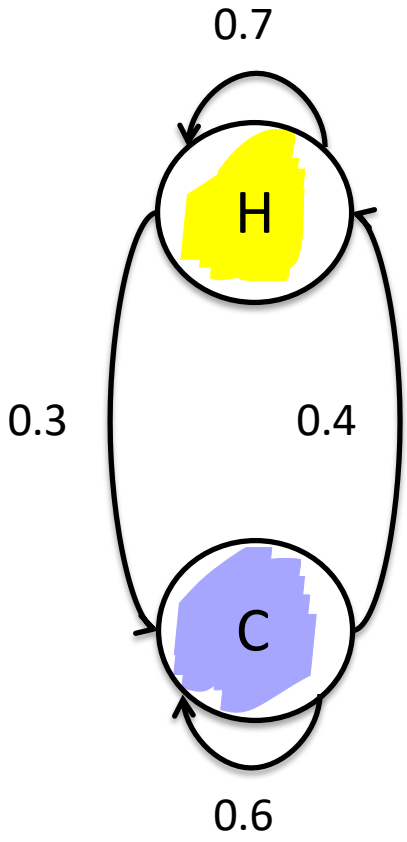
Suppose we're interested in average annual temperature

Only consider Hot and Cold

From recorded history, obtain probabilities for…

Year-to-year transitions

Based on thermometer readings for "recent" years

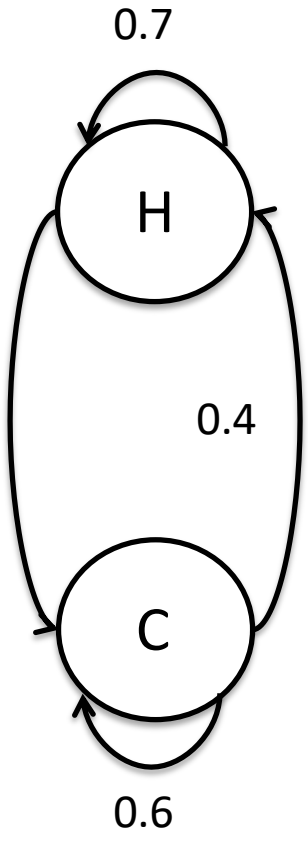


Markov Chain

Transition probability matrix

Matrix is denoted as A

$$\begin{array}{c} \\ H \\ C \end{array} \begin{array}{c} \begin{array}{cc} H & C \end{array} \\ \begin{bmatrix} 0.7 & 0.3 \\ 0.4 & 0.6 \end{bmatrix} \end{array}$$



Note, A is "row stochastic"

$$A = \begin{bmatrix} 0.7 & 0.3 \\ 0.4 & 0.6 \end{bmatrix} \begin{array}{l} = 1 \\ = 1 \end{array}$$

So, each element of A is between 0 and 1 and each row satisfies the definition of a discrete probability distribution, thus the elements of any given row sum to 1.

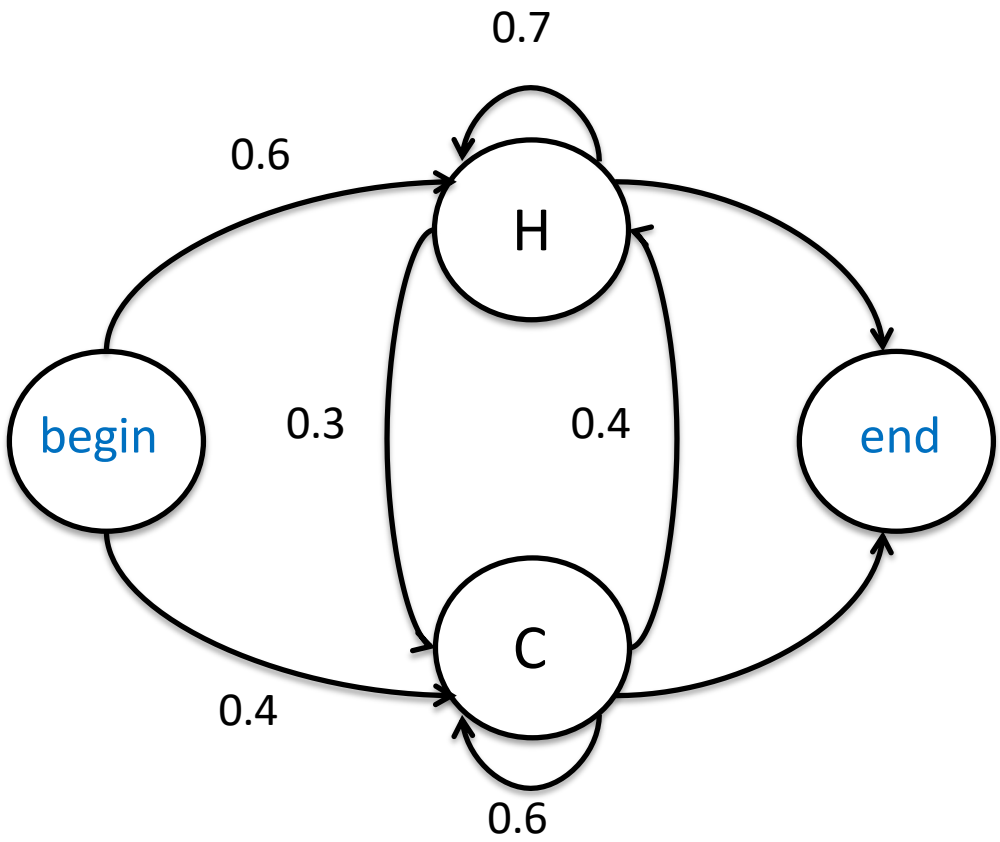
Markov Chain

Can also include begin, end states
Matrix for begin state denoted π

In this example,

$$\pi = \begin{bmatrix} 0.6 & 0.4 \end{bmatrix}$$

Note that π is also row stochastic



Hidden Markov Model

HMM includes a Markov chain/ process

- But the Markov process is "hidden", i.e., we can't directly observe the Markov process
- Instead, observe things that are probabilistically related to hidden states
- It's as if there is a "curtain" between Markov chain and the observations

HMM Example

Consider H/C annual temp example
Suppose we want to know H or C annual temperature in distant past

- Before thermometers were invented
- Note, we only distinguish between H and C

We assume transition between Hot and Cold years is same as today

Then the A matrix is known

HMM Example

Temps in past follow a Markov process

But, we cannot observe temperature in past

We find evidence that tree ring size is related to temperature

Looking at historical data, we find this holds

We only consider 3 tree ring sizes

Small, Medium, Large (S, M, L, respectively)

Measure tree ring sizes and recorded temperatures to determine relationship

HMM Example

We find that tree ring sizes and temperature related by

$$\begin{array}{c} \\ H \\ C \end{array} \begin{array}{c} \begin{array}{ccc} S & M & L \end{array} \\ \begin{bmatrix} 0.1 & 0.4 & 0.5 \\ 0.7 & 0.2 & 0.1 \end{bmatrix} \end{array}$$

This is known as the B matrix

The matrix B is also row stochastic

$$B = \begin{bmatrix} 0.1 & 0.4 & 0.5 \\ 0.7 & 0.2 & 0.1 \end{bmatrix}$$

HMM Example

Can we now find H/C temps in past?
We cannot measure (observe) temps
But we can measure tree ring sizes…
…and tree ring sizes related to temps
    By probabilities in the B matrix
Can we say something intelligent about temps over some interval in the past?

HMM Notation

A lot of notation is required

Notation may be the most difficult part…

$$
\begin{aligned}
T &= \text{the length of the observation sequence} \\
N &= \text{the number of states in the model} \\
M &= \text{the number of observation symbols} \\
Q &= \{q_0, q_1, \ldots, q_{N-1}\} = \text{the states of the Markov process} \\
V &= \{0, 1, \ldots, M-1\} = \text{set of possible observations} \\
A &= \text{the state transition probabilities} \\
B &= \text{the observation probability matrix} \\
\pi &= \text{the initial state distribution} \\
\mathcal{O} &= (\mathcal{O}_0, \mathcal{O}_1, \ldots, \mathcal{O}_{T-1}) = \text{observation sequence.}
\end{aligned}
$$

HMM Notation

Note that for simplicity, observations taken from V = {0,1,…,M-1}

That is, $\mathcal{O}_i \in V \text{ for } i = 0, 1, \ldots, T-1$

The matrix A = $\{a_{ij}\}$ is N x N, where

$$a_{ij} = P(\text{state } q_j \text{ at } t+1 \,|\, \text{state } q_i \text{ at } t)$$

The matrix B = $\{b_j(k)\}$ is N x M, where

$$b_j(k) = P(\text{observation } k \text{ at } t \,|\, \text{state } q_j \text{ at } t)$$

HMM Example

Consider our temperature example…

What are the possible observations?

V = {0,1,2}, corresponding to S, M, L

What are states of Markov process?

Q = {H,C}

What are A,B, π, and T?

A,B, π on previous slides

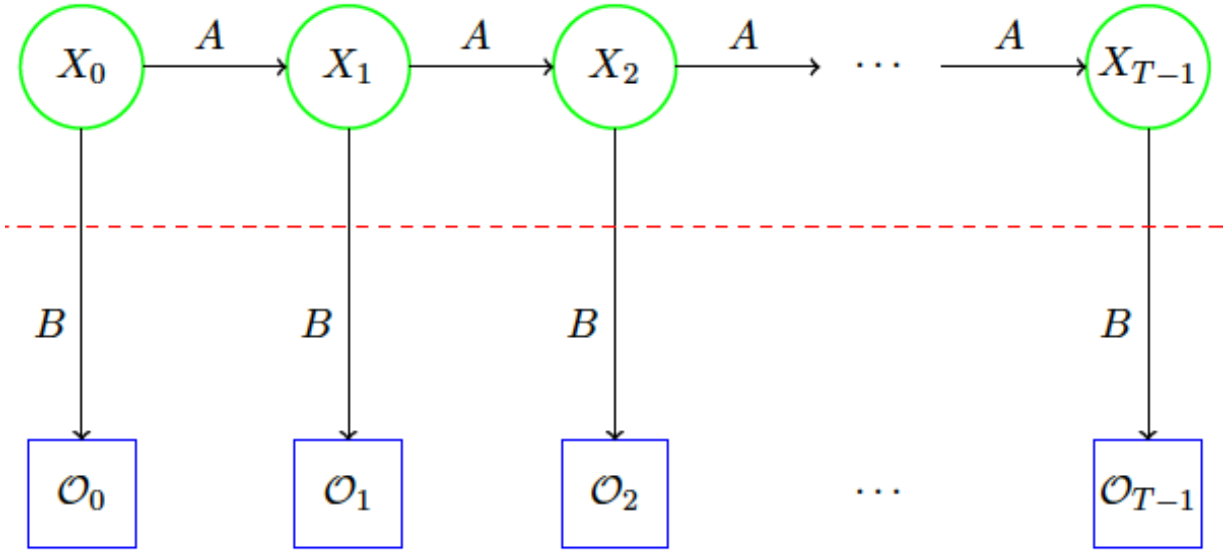
T is number of tree rings measured

What are N and M?

N = 2 and M = 3

Generic HMM

Generic view of HMM



HMM defined by A,B, and π
We denote HMM "model" as λ = (A,B,π)

HMM Example

Suppose that we observe tree ring sizes
For a 4 year period of interest: S,M,S,L
Then $\mathcal{O} = (\mathcal{O}_0, \mathcal{O}_1, \mathcal{O}_2, \mathcal{O}_3)$ = (0, 1, 0, 2)
Most likely (hidden) state sequence?
That is, most likely X = (X_0, X_1, X_2, X_3)
Let π_{X0} be prob. of starting in state x_0
Note $b_{x_0}(\mathcal{O}_0)$ prob. of initial observation
And $a_{X0,X1}$ is prob. of transition X_0 to X_1
And so on…

HMM Example

$$\pi = \begin{bmatrix} 0.6 & 0.4 \end{bmatrix}$$

$$\begin{array}{c} \\ H \\ C \end{array} \begin{array}{c} \begin{array}{ccc} S & M & L \end{array} \\ \begin{bmatrix} 0.1 & 0.4 & 0.5 \\ 0.7 & 0.2 & 0.1 \end{bmatrix} \end{array} \qquad \begin{array}{c} \\ H \\ C \end{array} \begin{array}{c} \begin{array}{cc} H & C \end{array} \\ \begin{bmatrix} 0.7 & 0.3 \\ 0.4 & 0.6 \end{bmatrix} \end{array}$$

Bottom line?

We can compute $P(X,O)$ for any X

For $X = (X_0, X_1, X_2, X_3)$ we have $P(X,O)$

$$= \pi_{x_0} b_{x_0}(\mathcal{O}_0) a_{x_0,x_1} b_{x_1}(\mathcal{O}_1) a_{x_1,x_2} b_{x_2}(\mathcal{O}_2) a_{x_2,x_3} b_{x_3}(\mathcal{O}_3)$$

Suppose we observe (0,1,0,2), then what is probability of, say, HHCC? S,M,S,L

Plug into formula above to find $P(HHCC,0,1,0,2)$

$$= 0.6(0.1)(0.7)(0.4)(0.3)(0.7)(0.6)(0.1) = 0.000212.$$

HMM Example

Do same for all 4-state sequences
We find that the winner is…
CCCH
Not so fast!

state	probability	normalized probability
$HHHH$	.000412	.042787
$HHHC$	.000035	.003635
$HHCH$	.000706	.073320
$HHCC$	.000212	.022017
$HCHH$	.000050	.005193
$HCHC$	.000004	.000415
$HCCH$	.000302	.031364
$HCCC$	.000091	.009451
$CHHH$	.001098	.114031
$CHHC$	.000094	.009762
$CHCH$	.001882	.195451
$CHCC$	.000564	.058573
$CCHH$	.000470	.048811
$CCHC$	.000040	.004154
$CCCH$	.002822	.293073
$CCCC$	.000847	.087963

Sum = $P(O)$ $P(X|O)=P(X,O)/P(O)$
divided by

HMM Example

aka Maximum Likelihood

The ***path*** CCCH scores the highest

In dynamic programming (DP), we find highest scoring path

But, in HMM we maximize ***expected number of correct states***

Sometimes called “EM algorithm”

For “Expectation Maximization”

How does HMM work in this example?

exam likely

EM solution

HMM Example

For first position…

Sum probabilities for all paths that have H in 1st position, compare to sum of probabilities for paths with C in 1st position: biggest wins

Repeat for each position and we find

	element			
	0	1	2	3
$P(H\|O)$	0.188182	0.519576	0.228788	0.804029
$P(C\|O)$	0.811818	0.480424	0.771212	0.195971

CHCH … different solution

HMM Example

Note: We could use both $P(X,O)$ and $P(X|O)$ to decide the best. Especially in paper and pencil problems, it is easier to use the un-normalized $P(X,O)$

	element			
	0	1	2	3
$P(H\|O)$	0.188182	0.519576	0.228788	0.804029
$P(C\|O)$	0.811818	0.480424	0.771212	0.195971

HMM Example

So, HMM solution gives us CHCH

While DP solution is CCCH

Which solution is better?

Neither solution is better!

Just using different definitions of “best”

	element			
	0	1	2	3
$P(H\|O)$	0.188182	0.519576	0.228788	0.804029
$P(C\|O)$	0.811818	0.480424	0.771212	0.195971

HMM Example

	element			
	0	1	2	3
$P(H\|O)$	0.188182	0.519576	0.228788	0.804029
$P(C\|O)$	0.811818	0.480424	0.771212	0.195971

So, HMM solution gives us CHCH

While DP solution is CCCH

Which solution is better?

Neither solution is better!

Just using different definitions of “best”

HMM Paradox?

HMM maximizes expected number of correct states

Whereas DP chooses “best” overall path

Possible for HMM to choose a “path” that is impossible

Say now it's CMH... C→M→H only... HMM gives CMHMCHM

but this is impossible

Could be a transition probability of 0

Cannot get impossible path with DP

Is this a flaw with HMM?

No, it’s a feature

Ex: DNA : ATCG

Observe a disease in multiple people, set up HMM, gives us an ATCG sequence, but it won't actually have an equivalent gene in humans... but it's similar enough to give a hint

Probability of Observations

Table computed for

$$\mathcal{O} = (\mathcal{O}_0, \mathcal{O}_1, \mathcal{O}_2, \mathcal{O}_3)$$
$$= (0,1,0,2)$$

For this sequence,

$$P(\mathcal{O}) = .000412 + .000035 + .000706 + \ldots + .000847$$
$$= \text{left to the reader}$$

Similarly for other observations $\mathcal{O}$

$\sum_x P(X,\mathcal{O})$ is prob. of $(0,1,0,2)$

state	probability	normalized probability
$HHHH$	.000412	.042787
$HHHC$	.000035	.003635
$HHCH$	.000706	.073320
$HHCC$	.000212	.022017
$HCHH$	.000050	.005193
$HCHC$	.000004	.000415
$HCCH$	.000302	.031364
$HCCC$	.000091	.009451
$CHHH$	.001098	.114031
$CHHC$	.000094	.009762
$CHCH$	.001882	.195451
$CHCC$	.000564	.058573
$CCHH$	.000470	.048811
$CCHC$	.000040	.004154
$CCCH$	.002822	.293073
$CCCC$	.000847	.087963

Probability of Observations

If this calculation is made for all possible 4-observation sequences then the sum of the resulting probabilities (not the normalized probabilities) will be 1.

state	probability	normalized probability
$HHHH$	.000412	.042787
$HHHC$	.000035	.003635
$HHCH$	.000706	.073320
$HHCC$	.000212	.022017
$HCHH$	.000050	.005193
$HCHC$	.000004	.000415
$HCCH$	.000302	.031364
$HCCC$	.000091	.009451
$CHHH$	.001098	.114031
$CHHC$	.000094	.009762
$CHCH$	.001882	.195451
$CHCC$	.000564	.058573
$CCHH$	.000470	.048811
$CCHC$	.000040	.004154
$CCCH$	.002822	.293073
$CCCC$	.000847	.087963

$3^4 = 81$

$\sum_{0} \sum_{x} p(y,o) = 1$

Law of total probability

HMM Model

An HMM is defined by the three matrices, A, B, and π
Note that M and N are implied, since they are the dimensions of matrices
So, we denote an HMM "model" as

$$\lambda = (A,B,\pi)$$

A : M x M

B : N x M (i think)

The Three Problems

HMMs used to solve 3 problems:

Problem 1: Given a model $\lambda = (A,B,\pi)$ and observation sequence O, find $P(O|\lambda)$ We saw this earlier

That is, we can ***score*** an observation sequence to see how well it fits a given model

Final material finished after these 3 problems

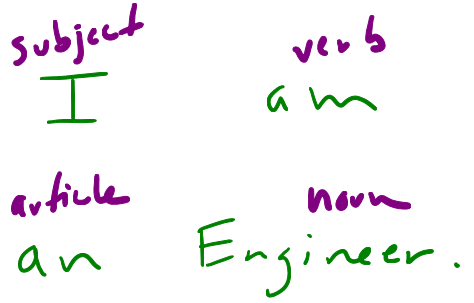
The Three Problems

HMMs used to solve 3 problems

Problem 2: Given $\lambda = (A,B,\pi)$ and O, find an optimal state sequence (in HMM sense)

Uncover hidden part (like previous example)

In many applications in NLP, the solution to Problem 2 is crucial. Example: Finding the grammatical roles of words in a sentence. They're our hidden states.



The Three Problems

HMMs used to solve 3 problems

Problem 3: Given O, N, and M, find the model λ that maximizes probability of O

Find A B π (I think)

That is, ***train*** a model to fit observations

HMMs in Practice

Often, HMMs used as follows:

Given an observation sequence…

- Assume that (hidden) Markov process exists

Train a model based on observations

- That is, solve Problem 3
- "Best" N can be found by trial and error

Then given a sequence of observations, score it versus the model we trained

- This is Problem 1: high score implies similar to training data, low score says it's not

HMMs in Practice

In this sense, HMM is a “machine learning” technique

- To train a model, we do not need to specify anything except the parameter N
- “Best” N often found by trial and error

So, we don’t need to “think” too much

- Just train HMM and then use it
- Fortunately, there are efficient algorithms for HMMs

The Three Solutions

We give detailed solutions to 3 problems

- Note: We must find ***efficient*** solutions

The three problems:

- **Problem 1**: Score an observation sequence versus a given model
- **Problem 2**: Given a model, "uncover" hidden part
- **Problem 3**: Given an observation sequence, train a model

Recall that we considered example for 2 and 1, but direct solutions are **very inefficient**

The Three Solutions
Appendix: Algorithmic Details

Solution 1

Score observations versus a given model

Given model $\lambda = (A,B,\pi)$ and observation sequence $O=(O_0,O_1,\ldots,O_{T-1})$, find $P(O|\lambda)$

Denote hidden states as

$$X = (X_0, X_1, \ldots, X_{T-1})$$

Then from definition of B,

$$P(O|X,\lambda)=b_{X0}(O_0)\, b_{X1}(O_1) \ldots b_{XT-1}(O_{T-1})$$

And from definition of A and π,

$$P(X|\lambda)=\pi_{X0}\, a_{X0,X1}\, a_{X1,X2} \ldots a_{XT-2,XT-1}$$

Solution 1

Elementary conditional probability fact:

$$P(O,X|\lambda) = P(O|X,\lambda)\, P(X|\lambda)$$

Sum over all possible state sequences X,

$$P(O|\lambda) = \Sigma\, P(O,X|\lambda) = \Sigma\, P(O|X,\lambda)\, P(X|\lambda)$$

$$= \Sigma \pi_{X0} b_{X0}(O_0) a_{X0,X1} b_{X1}(O_1) \ldots a_{XT-2,XT-1} b_{XT-1}(O_{T-1})$$

This “works” but way too costly

Requires about $2TN^T$ multiplications

Why?

There should be a better way…

Forward Algorithm

Instead, use ***forward algorithm***

Or “alpha pass”

For t = 0,1,…,T-1 and i=0,1,…,N-1, let

$$\alpha_t(i) = P(O_0,O_1,\ldots,O_t,X_t=q_i|\lambda)$$

Probability of “partial observation” to t, and Markov process is in state q_i at step t

Can be computed recursively, efficiently

Forward Algorithm

Let $\alpha_0(i) = \pi_i b_i(O_0)$ for $i = 0,1,\ldots,N-1$

For $t = 1,2,\ldots,T-1$ and $i=0,1,\ldots,N-1$, let

$$\alpha_t(i) = \left(\Sigma \alpha_{t-1}(j) a_{ji}\right) b_i(O_t)$$

Where the sum is from $j = 0$ to $N-1$

From definition of $\alpha_t(i)$ we see

$$P(O|\lambda) = \Sigma \alpha_{T-1}(i)$$

Where the sum is from $i = 0$ to $N-1$

This requires only N^2T multiplications

Forward Algorithm

Note: The score $P(O|\lambda)$ is dependent on the length of the observation sequence. Consequently, to compare scores for sequences of different length, we can normalize to a per observation score, that is, score = $\Sigma\alpha_{T-1}(i) / T$.

Forward Algorithm

In a nutshell, $\alpha_T(i)$ is the probability that the sequence of observations up to time T correspond to the state being the i^{th} state at time T, given the model λ.

We compute $\alpha_t(i)$ recursively using $\alpha_{t-1}(j)$'s.

Solution 2

Given a model, find hidden states

- Given $\lambda = (A,B,\pi)$ and O, find an optimal state sequence
- Recall that optimal means "maximize expected number of correct states"
- In contrast, DP finds best scoring path

For temp/tree ring example, solved this

- But hopelessly inefficient approach

A better way: ***backward algorithm***

- Or "beta pass"

Backward Algorithm

For $t = 0,1,\ldots,T-1$ and $i = 0,1,\ldots,N-1$, let $\beta_t(i) = P(O_{t+1},O_{t+2},\ldots,O_{T-1}|X_t=q_i,\lambda)$

Probability of partial observation from t to end and Markov process in state q_i at step t

Analogous to the forward algorithm

As with forward algorithm, this can be computed recursively and efficiently

$\beta_t(i) = P(O_{t+1},O_{t+2},\ldots,O_{T-1}|X_t=q_i,\lambda)$

Backward Algorithm

Let $\beta_{T-1}(i) = 1$ for $i = 0,1,\ldots,N-1$

For $t = T-2,T-3, \ldots,1$ and $i = 0,1,\ldots,N-1$, let

$$\beta_t(i) = \Sigma a_{ij} b_j(O_{t+1}) \beta_{t+1}(j)$$

Where the sum is from $j = 0$ to $N-1$

Solution 2

For t = 1,2,…,T-1 and i=0,1,…,N-1 define

$\gamma_t(i) = P(X_t=q_i|O,\lambda)$

(The probability of being in state i at time t)

Most likely state at t is q_i that maximizes $\gamma_t(i)$

$$\widetilde{X}_t = \max_i \gamma_t(i).$$

Note that $\gamma_t(i) = \alpha_t(i)\beta_t(i)/P(O|\lambda)$

And recall $P(O|\lambda) = \Sigma\alpha_{T-1}(i)$

Solution 2

The bottom line?

Forward algorithm solves Problem 1

Forward/backward algorithms solve Problem 2 by computing $\gamma_t(i)$ for each state at each time step (using both forward and backward paths) and choosing the state that maximizes it.

$$\tilde{X}_t = \max_i \gamma_t(i).$$

Solution 2

Why is it necessary to normalize gamma by dividing by P(O | lambda)? Because these probabilities are computed assuming the observation sequence is known (i.e., given O), as opposed to being computed relative to the larger probability space.

Solution 3

Train a model: Given O, N, and M, find λ that maximizes probability of O

We'll iteratively adjust $\lambda = (A,B,\pi)$ to better fit the given observations O

- The size of matrices are fixed (N and M)
- But elements of matrices can change

It is nice that this works…

- …and amazing that it's efficient!

Solution 3

For t=0,1,…,T-2 and i,j in {0,1,…,N-1}, define "di-gammas" as

$$\gamma_t(i,j) = P(X_t=q_i, X_{t+1}=q_j|O,\lambda)$$

Note $\gamma_t(i,j)$ is prob of being in state q_i at time t and transiting to state q_j at t+1

Then $\gamma_t(i,j) = \alpha_t(i)a_{ij}b_j(O_{t+1})\beta_{t+1}(j)/P(O|\lambda)$

And $\gamma_t(i) = \Sigma\gamma_t(i,j)$

Where sum is from j = 0 to N – 1

Baum-Welch Model Re-estimation

Given di-gammas and gammas…

For $i = 0,1,\ldots,N-1$ let $\pi_i = \gamma_0(i)$

For $i = 0,1,\ldots,N-1$ and $j = 0,1,\ldots,N-1$

$$a_{ij} = \Sigma\gamma_t(i,j)/\Sigma\gamma_t(i)$$

Where both sums are from t = 0 to T-2

For $j = 0,1,\ldots,N-1$ and $k = 0,1,\ldots,M-1$

$$b_j(k) = \Sigma\gamma_t(j)/\Sigma\gamma_t(j)$$

Both sums from from t = 0 to T-1 but only t for which $O_t = k$ are counted in numerator

Baum-Welch Model Re-estimation

Why does this work?

For the given observation sequence O, the sum $\Sigma\gamma_t(i)$ gives us the current best estimate for the total probability of being in state i, while $\Sigma\gamma_t(i,j)$ gives us the total probability of transitioning from state i to state j. Hence, the ratio $\Sigma\gamma_t(i,j)/\Sigma\gamma_t(i)$ enables us to re-estimate a_{ij} based on the current model parameters and observation sequence.

Solution 3

To summarize…

1. Initialize $\lambda = (A,B,\pi)$
2. Compute $\alpha_t(i)$, $\beta_t(i)$, $\gamma_t(i,j)$, $\gamma_t(i)$
3. Re-estimate the model $\lambda = (A,B,\pi)$
4. If $P(O|\lambda)$ increases by more than ε, goto 2 (where ε is small)

Solution 3

Some fine points…

Model initialization

- If we have a good guess for $\lambda = (A,B,\pi)$ then we can use it for initialization
- If not, let $\pi_i \approx 1/N$, $a_{i,j} \approx 1/N$, $b_j(k) \approx 1/M$
- Subject to row stochastic conditions
- But, do ***not*** initialize to exactly uniform values

Stopping conditions

- Stop after some number of iterations and/or…
- Stop if increase in $P(O|\lambda)$ is too small

HMM as Discrete Hill Climb

Algorithm on previous slides shows that HMM is a “discrete hill climb”

HMM consists of discrete parameters

- Specifically, the elements of the matrices

And re-estimation process improves model by modifying parameters

- So, process “climbs” toward improved model
- This happens in a high-dimensional space

Dynamic Programming: The Viterbi Algorithm

Brief detour…

For $\lambda = (A,B,\pi)$ as above, it's easy to define a dynamic program (DP)

Executive summary:

DP is forward algorithm, with "sum" replaced by "max"

Dynamic Programming: The Viterbi Algorithm

Let $\delta_0(i) = \pi_i\, b_i(O_0)$ for $i=0,1,\ldots,N-1$

For $t=1,2,\ldots,T-1$ and $i=0,1,\ldots,N-1$ compute

$$\delta_t(i) = \max\left(\delta_{t-1}(j)a_{ji}\right)b_i(O_t)$$

Where the max is over j in $\{0,1,\ldots,N-1\}$

Note that at each t, the DP computes best path for each state, up to that point

So, probability of best path is $\max \delta_{T-1}(j)$

This max gives the highest probability

Not the best path, for that, see next slide

Dynamic Programming: The Viterbi Algorithm

- To determine optimal path
  - While computing deltas, keep track of pointers to previous state
  - When finished, construct optimal path by tracing back points

Dynamic Programming: The Viterbi Algorithm

For example, consider temp example: recall that we observe (0,1,0,2)

$$\pi = \begin{bmatrix} 0.6 & 0.4 \end{bmatrix}$$

$$\begin{array}{c} \\ H \\ C \end{array} \begin{array}{c} \begin{array}{ccc} S & M & L \end{array} \\ \begin{bmatrix} 0.1 & 0.4 & 0.5 \\ 0.7 & 0.2 & 0.1 \end{bmatrix} \end{array} \qquad \begin{array}{c} \\ H \\ C \end{array} \begin{array}{c} \begin{array}{cc} H & C \end{array} \\ \begin{bmatrix} 0.7 & 0.3 \\ 0.4 & 0.6 \end{bmatrix} \end{array}$$

Probabilities for path of length 1:

$$P(H) = \pi_0 b_0(0) = 0.6(0.1) = 0.06 \text{ and } P(C) = \pi_1 b_1(0) = 0.4(0.7) = 0.28.$$

These are the only “paths” of length 1

Dynamic Programming: The Viterbi Algorithm

Probabilities for each path of length 2

$$
\begin{aligned}
P(HH) &= 0.06(0.7)(0.4) = 0.0168 \\
P(HC) &= 0.06(0.3)(0.2) = 0.0036 \\
P(CH) &= 0.28(0.4)(0.4) = 0.0448 \\
P(CC) &= 0.28(0.6)(0.2) = 0.0336
\end{aligned}
$$

Best path of length 2 ending with H is CH

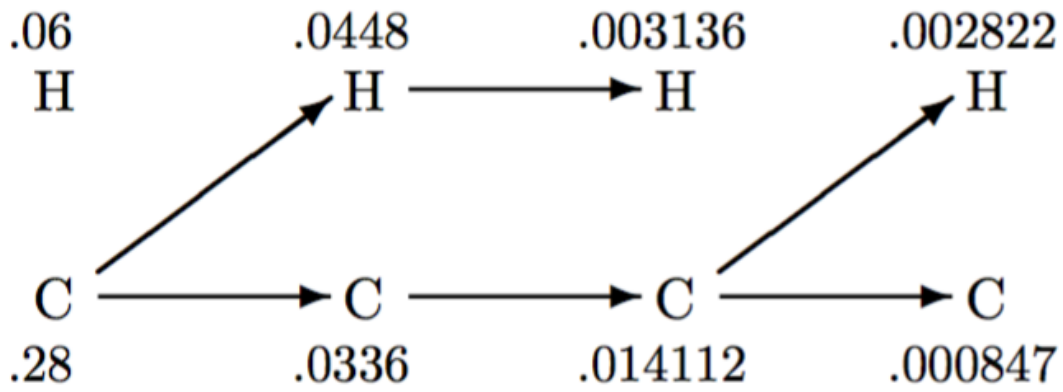
Best path of length 2 ending with C is CC

$$\delta_t(i) = \max \left(\delta_{t-1}(j) a_{ji} \right) b_i(O_t)$$

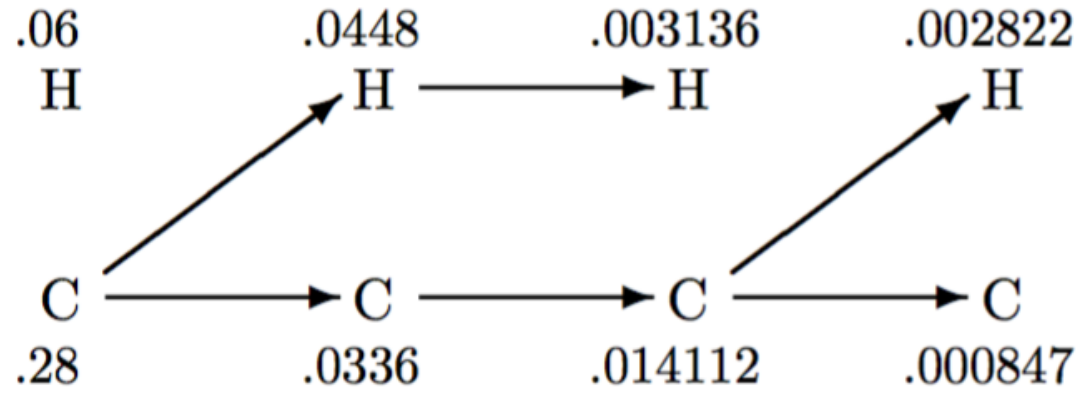
Dynamic Program

Continuing, we compute best path ending at H and C at each step

And save pointers: why?



Dynamic Program



Best final score is .002822

And thanks to pointers, best path is CCCH

But what about underflow?

A serious problem in bigger cases

Dynamic Program

https://www.youtube.com/watch?v=6JVqutwtzmo&t=4s

Underflow Resistant DP

Common trick to prevent underflow:

Instead of multiplying probabilities…

…add logarithms of probabilities

Why does this work?

Because log(xy) = log x + log y

Adding logs does not tend to 0

Note that these logs are negative…

…and we must avoid 0 probabilities

Underflow Resistant DP

Underflow resistant DP algorithm:

Let $\delta_0(i) = \log(\pi_i\, b_i(O_0))$
for $i=0,1,\ldots,N-1$

For $t=1,2,\ldots,T-1$ and $i=0,1,\ldots,N-1$ compute

$$\delta_t(i) = \max\left(\delta_{t-1}(j) + \log(a_{ji}) + \log(b_i(O_t))\right)$$

Where the max is over j in $\{0,1,\ldots,N-1\}$

And ***score*** of best path is max $\delta_{T-1}(j)$

As before, must also keep track of paths

HMM Scaling

Trickier to prevent underflow in HMM

We consider solution 3

Since it includes solutions 1 and 2

Recall for t = 1,2,…,T-1, i=0,1,…,N-1,

$$\alpha_t(i) = \left(\Sigma \alpha_{t-1}(j) a_{j,i}\right) b_i(O_t)$$

The idea is to normalize alphas so that they sum to 1

Algorithm on next slide

HMM Scaling

Given $\alpha_t(i) = \left(\Sigma\alpha_{t-1}(j)a_{j,i}\right)b_i(O_t)$

Let $a_0(i) = \alpha_0(i)$ for i=0,1,…,N-1

Let $c_0 = 1/\Sigma a_0(j)$

For i = 0,1,…,N-1, let $\alpha_0(i) = c_0 a_0(i)$

This takes care of t = 0 case

Algorithm continued on next slide…

HMM Scaling

For t = 1,2,…,T-1 do the following:
For i = 0,1,…,N-1,

$$a_t(i) = \left(\Sigma \alpha_{t-1}(j) a_{j,i}\right) b_i(O_t)$$

Let $c_t = 1/\Sigma a_t(j)$
For i = 0,1,…,N-1 let $\alpha_t(i) = c_t a_t(i)$

HMM Scaling

Easy to show $\alpha_t(i) = c_0 c_1 \ldots c_t \, \alpha_t(i)$ (♯)

Simple proof by induction

So, $c_0 c_1 \ldots c_t$ is scaling factor at step t

Also, easy to show that

$\alpha_t(i) = \alpha_t(i) / \Sigma \alpha_t(j)$

Which implies $\Sigma a_{T-1}(i) = 1$ (♯♯)

HMM Scaling

By combining (♯) and (♯♯), we have

$$1 = \Sigma \alpha_{T-1}(i) = c_0 c_1 \ldots c_{T-1} \Sigma \alpha_{T-1}(i)$$

$$= c_0 c_1 \ldots c_{T-1} P(O|\lambda)$$

Therefore, $P(O|\lambda) = 1 / c_0 c_1 \ldots c_{T-1}$

To avoid underflow, we compute

$$\log P(O|\lambda) = -\Sigma \log(c_j)$$

Where sum is from j = 0 to T-1

HMM Scaling

Similarly, scale betas as $c_t\beta_t(i)$

For re-estimation,

- Compute $\gamma_t(i,j)$ and $\gamma_t(i)$ using original formulas, but with scaled alphas, betas

This gives us new values for $\lambda = (A,B,\pi)$

"Easy exercise" to show re-estimate is exact when scaled alphas and betas used

Also, $P(O|\lambda)$ cancels from formula

- Use $\log P(O|\lambda) = -\Sigma \log(c_j)$ to decide if iterate improves

All Together Now

Complete pseudo code for Solution 3

Given: $(O_0,O_1,\ldots,O_{T-1})$ and N and M

Initialize: $\lambda = (A,B,\pi)$

A is NxN, B is NxM and π is 1xN

$\pi_i \approx 1/N$, $a_{ij} \approx 1/N$, $b_j(k) \approx 1/M$, each matrix row stochastic, but not uniform

Initialize:

maxIters = max number of re-estimation steps

iters = 0

oldLogProb = $-\infty$

Forward Algorithm

Forward algorithm
With scaling

```
// compute α_0(i)
c_0 = 0
for i = 0 to N − 1
    α_0(i) = π(i) b_i(O_0)
    c_0 = c_0 + α_0(i)
next i

// scale the α_0(i)
c_0 = 1/c_0
for i = 0 to N − 1
    α_0(i) = c_0 α_0(i)
next i

// compute α_t(i)
for t = 1 to T − 1
    c_t = 0
    for i = 0 to N − 1
        α_t(i) = 0
        for j = 0 to N − 1
            α_t(i) = α_t(i) + α_{t−1}(j) a_ji
        next j
        α_t(i) = α_t(i) b_i(O_t)
        c_t = c_t + α_t(i)
    next i

    // scale α_t(i)
    c_t = 1/c_t
    for i = 0 to N − 1
        α_t(i) = c_t α_t(i)
    next i
next t
```

Backward Algorithm

Backward algorithm or “beta pass”

With scaling

Note: same scaling factor as alphas

// Let $\beta_{T-1}(i) = 1$ scaled by c_{T-1}
for $i = 0$ to $N - 1$
    $\beta_{T-1}(i) = c_{T-1}$
next i

// β-pass
for $t = T - 2$ to 0 by -1
    for $i = 0$ to $N - 1$
        $\beta_t(i) = 0$
        for $j = 0$ to $N - 1$
            $\beta_t(i) = \beta_t(i) + a_{ij} b_j(\mathcal{O}_{t+1}) \beta_{T+1}(j)$
        next j
        // scale $\beta_t(i)$ with same scale factor as $\alpha_t(i)$
        $\beta_t(i) = c_t \beta_t(i)$
    next i
next t

Gammas

Using scaled alphas and betas

for $t = 0$ to $T - 2$
    $\texttt{denom} = 0$
    for $i = 0$ to $N - 1$
        for $j = 0$ to $N - 1$
            $\texttt{denom} = \texttt{denom} + \alpha_t(i) a_{ij} b_j(\mathcal{O}_{t+1}) \beta_{t+1}(j)$
        next j
    next i
    for $i = 0$ to $N - 1$
        $\gamma_t(i) = 0$
        for $j = 0$ to $N - 1$
            $\gamma_t(i,j) = (\alpha_t(i) a_{ij} b_j(\mathcal{O}_{t+1}) \beta_{t+1}(j))/\texttt{denom}$
            $\gamma_t(i) = \gamma_t(i) + \gamma_t(i,j)$
        next j
    next i
next t

// Special case for $\gamma_{T-1}(i)$
$\texttt{denom} = 0$
for $i = 0$ to $N - 1$
    $\texttt{denom} = \texttt{denom} + \alpha_{T-1}(i)$
next i
for $i = 0$ to $N - 1$
    $\gamma_{T-1}(i) = \alpha_{T-1}(i)/\texttt{denom}$
next i

Re-Estimation

Again, using scaled gammas
So formulas unchanged

```
// re-estimate π
for i = 0 to N − 1
    π_i = γ_0(i)
next i

// re-estimate A
for i = 0 to N − 1
    for j = 0 to N − 1
        numer = 0
        denom = 0
        for t = 0 to T − 2
            numer = numer + γ_t(i, j)
            denom = denom + γ_t(i)
        next t
        a_ij = numer/denom
    next j
next i

// re-estimate B
for i = 0 to N − 1
    for j = 0 to M − 1
        numer = 0
        denom = 0
        for t = 0 to T − 2
            if(O_t == j) then
                numer = numer + γ_t(i)
            end if
            denom = denom + γ_t(i)
        next t
        b_i(j) = numer/denom
    next j
next i
```

Stopping Criteria

Check that probability increases

- In practice, want logProb > oldLogProb + ε

And don't exceed max iterations

```
// Compute log[P(O | λ)]

logProb = 0
for i = 0 to T − 1
    logProb = logProb + log(c_i)
next i
logProb = −logProb

// To iterate or not to iterate, that is the question...

iters = iters + 1
if (iters < maxIters and logProb > oldLogProb) then
    oldLogProb = logProb
    goto α-pass
else
    output λ = (π, A, B)
end if
```

References

M. Stamp, A revealing introduction to hidden Markov models
L.R. Rabiner, A tutorial on hidden Markov models and selected applications in speech recognition
R.L. Cave & L.P. Neuwirth, Hidden Markov models for English