Hidden Markov Models

CS 780/880 Natural Language Processing Lecture 9

Samuel Carton, University of New Hampshire

Last lecture

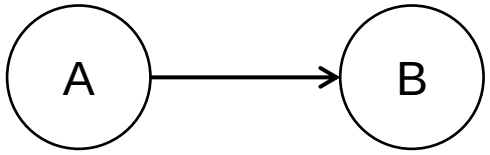
Key idea: Naïve Bayes algorithm for classification

Concepts:

- Bayes rule
- Base rate fallacy
- Naïve Bayes
- Overfitting vs underfitting
- Bias-variance trade-off

Bayesian networks

A collection of random variables linked together by conditional probability relationships



Implies that is dependent on/caused by A.

A Bayesian Network is a **directed acyclic graph** (DAG)

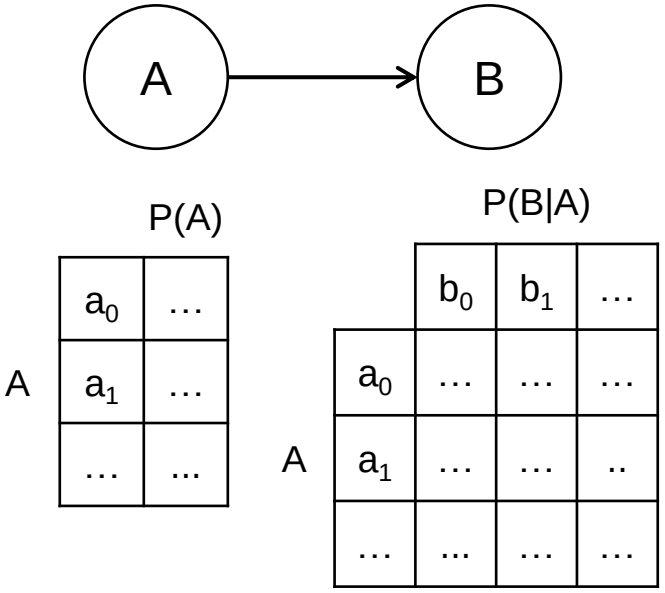
- Edges have a direction and imply causality
- No loops in the graph (No “A causes B causes C causes A” situations)

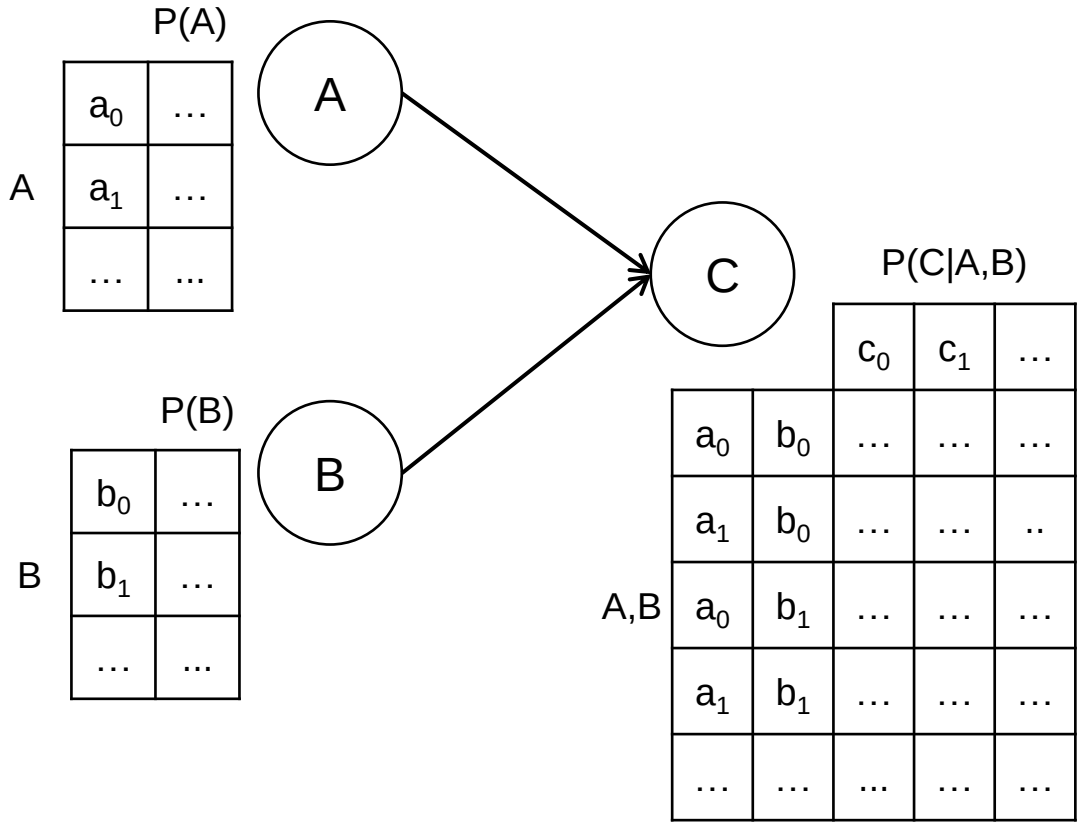
NLP examples:

- N-gram models
- Naïve Bayes model

Parameters of a Bayes Net

Each node in a bayes net is parameterized by its conditional probabilities relative to the nodes that connect to it





Three things we want to do

Learning: Given a dataset, what are the most likely parameters for our model?

- I.e. which parameters would make the data most probable under this model
- But also smoothing, etc.

Inference: Given our (fully-parameterized) model and (optionally) the value of one or more nodes, what is the likelihood of a given set of values for the other nodes?

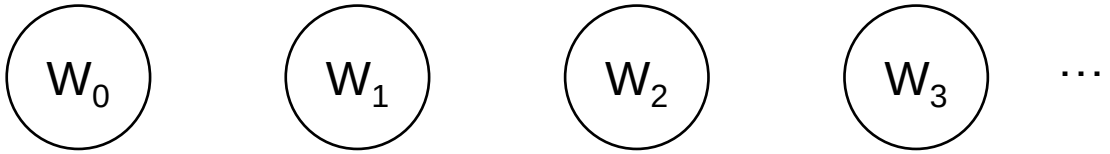
- E.g., what is the likelihood of a given sequence in a bigram model?
- E.g., what is the probability of class c_0 versus c_1 in a Naïve Bayes model, given the words in the text?

Generation: Given our model and (optionally) the values of some nodes, can we generate new values for the other nodes?

- Really just a special case of inference

Unigram model

A collection of freestanding nodes with the same CPT and no dependencies



P(W)

W	
w_0	…
w_1	…
…	...

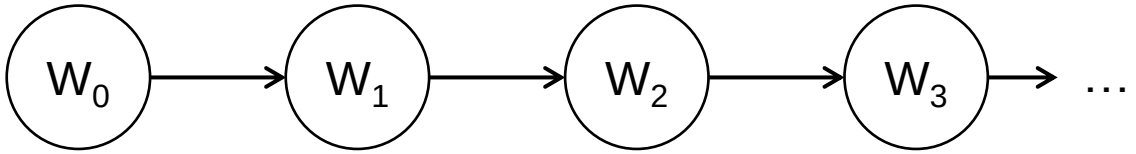
Learning: Count word frequencies within the corpus

Inference: Use product rule of independent probabilities

Generation: Generate each word separately according to CPT

Bigram model

A chain of dependent nodes, all using the same conditional probability table



$P(W_t|W_{t-1})$

W_{t-1}	w_0	w_1	…
w_0	…	…	…
w_1	…	…	..
…	...	…	…

Learning: Count bigram frequencies within the corpus

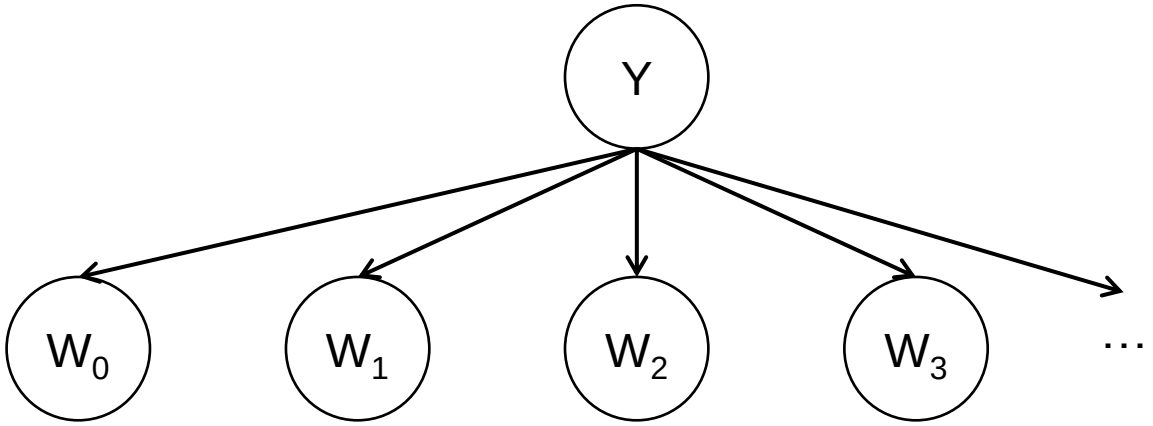
Inference: Use chain rule on consecutive bigrams

Generation: Generate words one at a time depending on preceding word

Can be interpreted as a **Markov Chain**

Naïve bayes model

A unigram model where the word probabilities depend (only) on the class



P(W|Y)

Y		w_0	w_1	…
	y_0	…	…	…
	y_1	…	…	..
	…	...	…	…

Learning: Count unigram frequencies for each class

Inference: Apply Bayes Rule to convert P(W|Y) into P(Y|W)

Generation: Generate class, then generate words independently

- Not really done

Generative story

In NLP, the **generative story** of a corpus of texts is the probabilistic model we suppose was used to produce it.

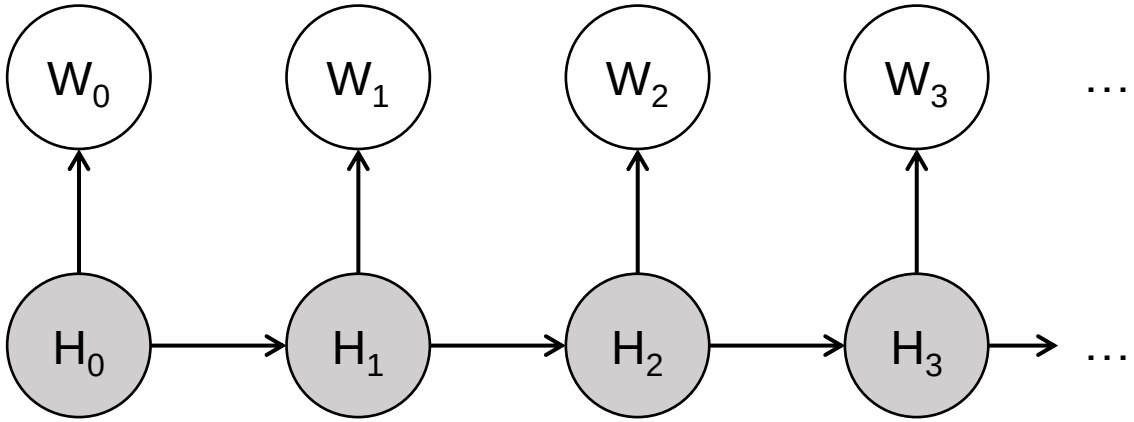
Then we **learn** the most likely parameters for that model based on the corpus

Then we can perform **inference**, as well as **generate** new text.

Hidden Markov Models

Hidden Markov Model

Basic idea: there's a sequence of **latent, hidden states** that are connected together in a Markov Chain, and then words are generated dependent **only** on the state at time *t*.



$P(W_t|H_t)$

H_t	w_0	w_1	…
h_0	…	…	…
h_1	…	…	..
…	…	…	…

Emission matrix

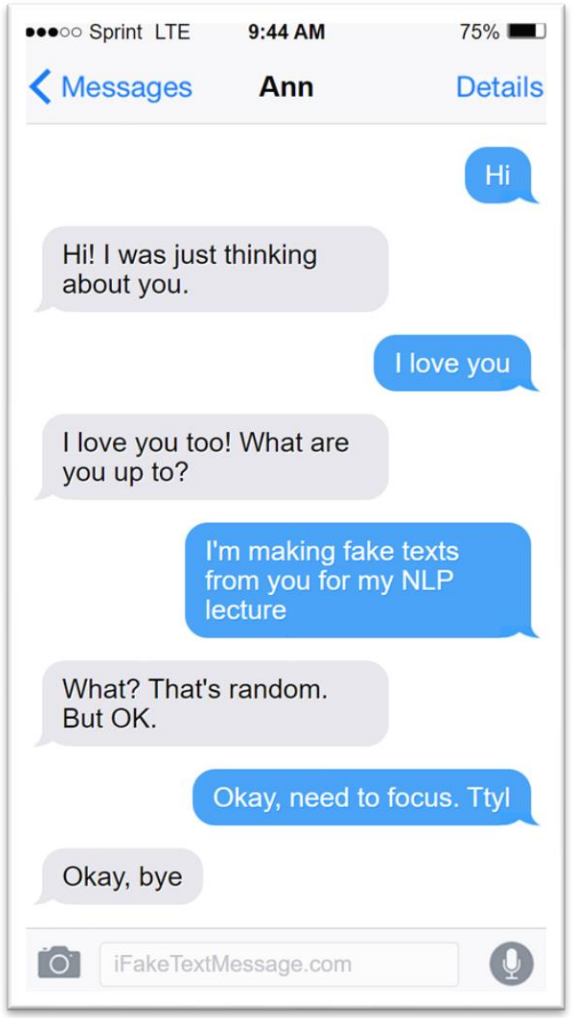
$P(H_t|H_{t-1})$

H_{t-1}	h_0	h_1	…
h_0	…	…	…
h_1	…	…	..
…	…	…	…

Transition matrix

Case study: Interpreting my wife's mood

Clearly upset



Clearly cheerful

Probably still cheerful

Probably upset

Probably still upset

Case study: Interpreting my wife's mood

Clearly cheerful

Probably upset

Maybe cheerful?

Clearly cheerful

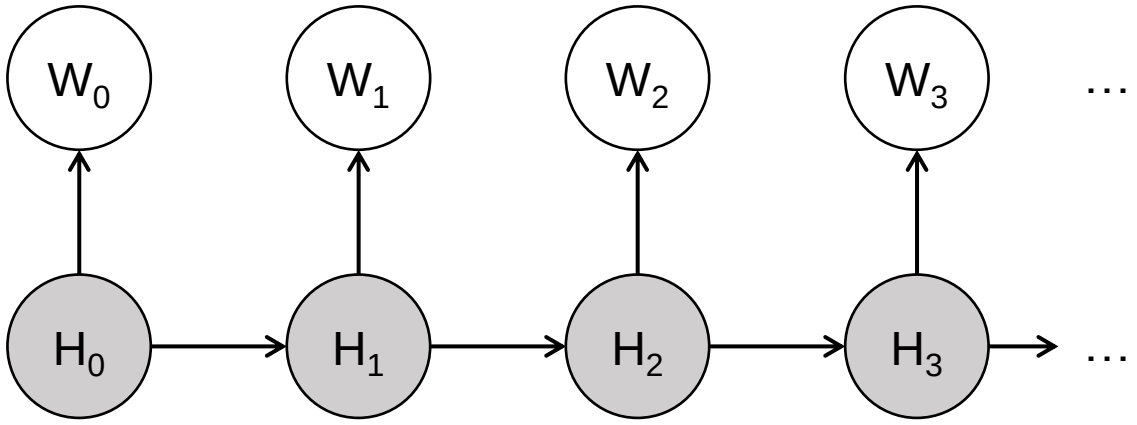
Maybe upset?

Case study: POS tagging

A popular application of HMMs in NLP is part-of-speech tagging

We imagine a generative story where parts-of-speech occur in a Markov chain, and then they emit words conditioned on their value.

i	sentence	you	to	read	this	sentence	.
PP	V	PP	PREP	V	DET	NN	PUNCT



Transition matrix

$P(H_t|H_{t-1})$

H_{t-1} \ H_t	h_0	h_1	…
h_0	…	…	…
h_1	…	…	..
…	…	…	…

Emission matrix

$P(W_t|H_t)$

H_t	w_0	w_1	…
h_0	…	…	…
h_1	…	…	..
…	…	…	…

Things we want to do with HMMs

HMMs diverge from N-gram and Naïve Bayes in requiring complicated algorithms to do certain things with them

Also, multiple scenarios of each operation

Learning
- Labeled
- Unlabeled

Inference
- Likelihood
- Decoding

Generation
- Special case of inference

Speech and Language Processing (Jurafsky & Martin) is a definitive source: https://web.stanford.edu/~jurafsky/slp3/ (Appendix A)

Doing inference on HMMs

Inference: Given the values of some of the nodes, what are the most likely values of other nodes?

Two important inference problems:

- **Likelihood**: given the full model, what is the likelihood of a given output text?
  - **Forward algorithm**
- **Decoding**: given the full model and a particular output text, what was the most likely sequence of hidden states that generated it?
  - **Viterbi algorithm**

Generation is a special case of likelihood inference, because we can just generate new tokens according to their likelihood.

Likelihood

Likelihood: Given an HMM $\lambda = (A,B)$ and an observation sequence O, determine the likelihood $P(O|\lambda)$.

- A is transition matrix, B is emission matrix

POS example: Given the whole model of POS transitions and emission probabilities, what is the overall likelihood of a piece of text?

Forward algorithm

- Dynamic programming algorithm
- $O(N^2T)$ time complexity
  - T is sequence length and N is number of possible states

Forward algorithm

For a particular state sequence Q, it is pretty easy to figure out the likelihood of a given output sequence O:

$$P(O|Q) = \prod_{i=1}^{T} P(o_i|q_i)$$

But: how do we calculate this for every possible state sequence?

Brute force: enumerate every possible state sequence, calculate probability of it (using chain rule), then use chain rule again to calculate P(O)

Key insight: instead of calculating the likelihood of every possible state sequence (which is exponentially large), we can just calculate, for each time step t, the total likelihood of having ended up at each state at that timestep

- And then we can just use the chain rule:

$$P(O) = \sum_{Q} P(O,Q) = \sum_{Q} P(O|Q)P(Q)$$

Forward algorithm

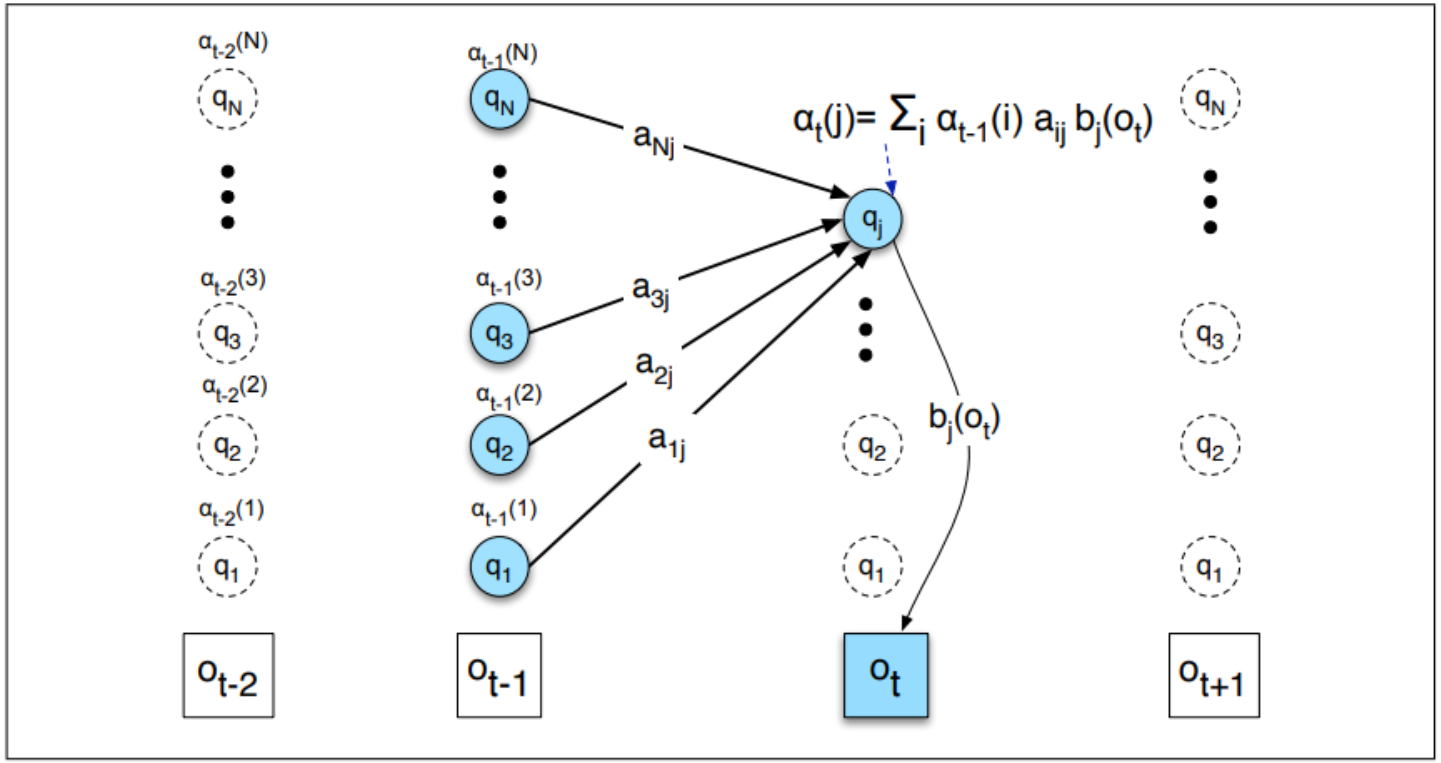


Figure A.6 Visualizing the computation of a single element $\alpha_t(i)$ in the trellis by summing all the previous values α_{t-1}, weighted by their transition probabilities a, and multiplying by the observation probability $b_i(o_t)$. For many applications of HMMs, many of the transition probabilities are 0, so not all previous states will contribute to the forward probability of the current state. Hidden states are in circles, observations in squares. Shaded nodes are included in the probability computation for $\alpha_t(i)$.

Forward algorithm

function FORWARD(*observations* of len T, *state-graph* of len N) **returns** *forward-prob*

create a probability matrix *forward[N,T]*
for each state s **from** 1 **to** N **do** ; initialization step
    $forward[s,1] \leftarrow \pi_s * b_s(o_1)$
for each time step t **from** 2 **to** T **do** ; recursion step
  for each state s **from** 1 **to** N **do**

$$forward[s,t] \leftarrow \sum_{s'=1}^{N} forward[s',t-1] * a_{s',s} * b_s(o_t)$$

$$forwardprob \leftarrow \sum_{s=1}^{N} forward[s,T]$$; termination step

return *forwardprob*

Figure A.7 The forward algorithm, where $forward[s,t]$ represents $\alpha_t(s)$.

Decoding

Decoding: Given as input an HMM $\lambda = (A,B)$ and a sequence of observations $O = o_1,o_2,...,o_T$, find the most probable sequence of states $Q = q_1q_2q_3 ...q_T$

POS example: Given a piece of text, what is the most likely sequence of parts of speech to have generated that text?

Viterbi algorithm

- Another dynamic programming algorithm

Viterbi algorithm

Similar to Forward algorithm

Brute force: enumerate every possible sequence of states, calculate probability of each one (using chain rule), use the chain rule to calculate P(O) given each possible sequence, then use Bayes rule to find the most likely state sequence

Key idea: For each timestep t, for each state j, calculate and store the probability of having ended up in state j at time t… **after following the most likely state sequence prior to t**.

Takes max over previous possible states rather than sum (which is what Forward algorithm does)

Viterbi algorithm

function VITERBI(*observations* of len T,*state-graph* of len N) **returns** *best-path*, *path-prob*

create a path probability matrix *viterbi*$[N,T]$
for each state s **from** 1 **to** N **do** ; initialization step
  viterbi$[s,1] \leftarrow \pi_s * b_s(o_1)$
  backpointer$[s,1] \leftarrow 0$
for each time step t **from** 2 **to** T **do** ; recursion step
  for each state s **from** 1 **to** N **do**
    $viterbi[s,t] \leftarrow \max_{s'=1}^{N} viterbi[s',t-1] * a_{s',s} * b_s(o_t)$
    $backpointer[s,t] \leftarrow \operatorname{argmax}_{s'=1}^{N} viterbi[s',t-1] * a_{s',s} * b_s(o_t)$
$bestpathprob \leftarrow \max_{s=1}^{N} viterbi[s,T]$; termination step
$bestpathpointer \leftarrow \operatorname{argmax}_{s=1}^{N} viterbi[s,T]$; termination step
bestpath $\leftarrow$ the path starting at state *bestpathpointer*, that follows backpointer[] to states back in time
return *bestpath*, *bestpathprob*

Figure A.9 Viterbi algorithm for finding optimal sequence of hidden states. Given an observation sequence and an HMM $\lambda = (A,B)$, the algorithm returns the state path through the HMM that assigns maximum likelihood to the observation sequence.

Learning HMMs

Learning: Given some text data, what are the most likely parameters

Two possible scenarios:

- Labeled
  - Ground-truth examples of both words and hidden states present
  - **Frequency counting**
- Unlabeled
  - Ground-truth examples of words… but not hidden states?
  - **Forward-backward algorithm**

Labeled learning

Given a corpus of text data **and** associated hidden state values, find most likely parameters for transition matrix and emission matrix

POS example:

i	sentence	you	to	read	this	sentence	.
PP	V	PP	PREP	V	DET	NN	PUNCT

i	dug	the	well	very	well	.
PP	V	DET	NN	ADV	ADV	PUNCT

Easy—just count.

- Populate transition matrix by counting POS→POS pairs
- Populate emission matrix by counting POS→Word pairs

Unlabeled learning

Given a corpus of text and **no** associated ground-truth hidden state values, find most likely parameters for transition and emission matrix

POS example:

i	sentence	you	to	read	this	sentence	.
?	?	?	?	?	?	?	?

i	dug	the	well	very	well	.
?	?	?	?	?	?	?

Much harder. Depends on finding implicit patterns in the data where certain words seem to be generated by certain hidden states with a certain transition matrix…

- Ends up being a form of clustering
- Use the **Forward-Backward** algorithm

Forward-backward Algorithm

- Also known as the **Baum-Welch algorithm**
- Special case of the **Expectation-Maximization algorithm**
- Iterative, approximate algorithm (rather than dynamic programming)

Basic idea: start with an initial guess for A and B.

- **Expectation step**: Generate most likely state values given (A, B, and O).
  - Use Viterbi algorithm for this
- **Maximization step**: Calculate most likely parameter values for A and B given generated state values and O
  - Same as labeled learning
- Repeat those two steps until convergence

Why does this work??? Black magic.

Forward-backward algorithm

function FORWARD-BACKWARD(*observations* of len T, *output vocabulary* V, *hidden state set* Q) **returns** *HMM=(A,B)*

initialize A and B

iterate until convergence

E-step

$$\gamma_t(j) = \frac{\alpha_t(j)\beta_t(j)}{\alpha_T(q_F)} \quad \forall\, t \text{ and } j$$

$$\xi_t(i,j) = \frac{\alpha_t(i)\, a_{ij} b_j(o_{t+1})\beta_{t+1}(j)}{\alpha_T(q_F)} \quad \forall\, t,\ i, \text{ and } j$$

M-step

$$\hat{a}_{ij} = \frac{\sum_{t=1}^{T-1} \xi_t(i,j)}{\sum_{t=1}^{T-1}\sum_{k=1}^{N} \xi_t(i,k)}$$

$$\hat{b}_j(v_k) = \frac{\sum_{t=1 s.t.\ O_t=v_k}^{T} \gamma_t(j)}{\sum_{t=1}^{T} \gamma_t(j)}$$

return A, B

Figure A.14 The forward-backward algorithm.

Concluding thoughts

Important intuition: idea of **latent, hidden states** that are responsible for generating the text

Hidden Markov models

- Most direct implementation of this idea
- Need fancy algorithms for learning and inference
- Not incredibly well-supported in Python
  - https://hmmlearn.readthedocs.io/en/latest/
- Largely superseded by RNNs these days
  - BUT… neurosymbolic and Bayesian neural networks are a hot research area: https://www.cs.toronto.edu/~duvenaud/distill_bayes_net/public/
- Good for building intuitions