UPPSALA UNIVERSITET

Basic Text Analysis

Hidden Markov Models

Joakim Nivre

Uppsala University
Department of Linguistics and Philology
joakim.nivre@lingfil.uu.se

Hidden Markov Models

- Markov models are probabilistic sequence models used for problems such as:
  1. Speech recognition
  2. Spell checking
  3. Part-of-speech tagging
  4. Named entity recognition
- A Markov model runs through a sequence of states emitting observable signals
- If the state sequence cannot be determined from the observation sequence, the model is said to be hidden

Hidden Markov Models

- A Markov model consists of five elements:
  1. A finite set of states $Q = \{q^1, \ldots, q^{|Q|}\}$
  2. A finite signal alphabet $\Sigma = \{s^1, \ldots, s^{|\Sigma|}\}$
  3. Initial probabilities $P(q)$ defining the probability of starting in state q (for every $q \in Q$)
  4. Transition probabilities $P(q \mid q')$ defining the probability of going from state q' to state q (for every $(q, q') \in Q^2$)
  5. Emission probabilities $P(s \mid q)$ defining the probability of emitting symbol s in state q (for every $(s, q) \in \Sigma \times Q$)
- Remember: If we add a start state, $P(q) = P(q \mid \text{start})$

Markov Assumptions

- State transitions are assumed to be independent of everything except the current state:

$$P(q_1, \ldots, q_n) = \prod_{i=1}^{n} P(q_i \mid q_{i-1})$$

- Signal emissions are assumed to be independent of everything except the current state:

$$P(q_1, \ldots, q_n, s_1, \ldots, s_n) = P(q_1, \ldots, q_n) \prod_{i=1}^{n} P(s_i \mid q_i)$$

- NB: subscripts on states and signals refer to sequence positions

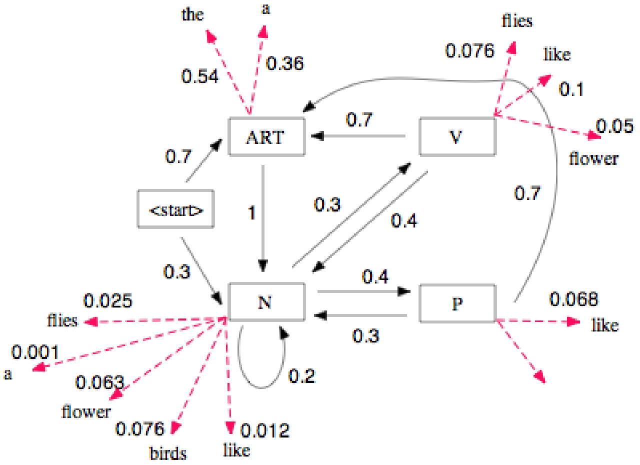
Tagging with an HMM

- Modeling assumptions:
  1. States represent tag *n*-grams
  2. Signals represent words
- Probability model (first-order, unigram states):

$$P(w_1, \ldots, w_n, t_1, \ldots, t_n) = \prod_{i=1}^{n} P(w_i | t_i) P(t_i | t_{i-1})$$

  - In an *n*th-order model, states represent tag *n*-grams
  - This gives an $n+1$-gram language model over tags
- Ambiguity:
  - The same word (signal) generated by different tags (states)
  - Language is ambiguous $\Leftrightarrow$ Markov model is hidden

A Simple First-Order HMM for Tagging



Problems for HMMs

- Decoding = finding the optimal state sequence

$$\operatorname*{argmax}_{q_1,\ldots,q_n} P(q_1,\ldots,q_n,s_1,\ldots,s_n)$$

- Learning = estimating the model parameters

$$\hat{P}(q_j|q_i) \quad \forall q_j, q_i \qquad\qquad \hat{P}(s|q) \quad \forall s, q$$

Decoding

- Given observation sequence $s_1, \ldots, s_n$, compute:

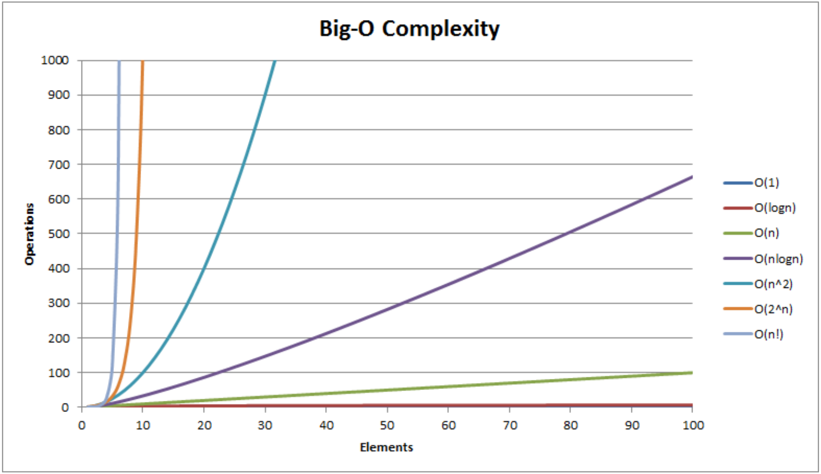
$$\underset{q_1,\ldots,q_n}{\operatorname{argmax}} P(q_1, \ldots, q_n, s_1, \ldots, s_n)$$

- Brute force solution:
  - For every possible state sequence $q_1, \ldots, q_n$
    - Compute $P(q_1, \ldots, q_n, s_1, \ldots, s_n) = \prod_{i=1}^{n} P(q_i|q_{i-1})P(s_i|q_i)$
  - Pick the sequence with highest probability
- Anything wrong with this?

Time Complexity

- How long does it take to compute a solution?
  - Actual running time depends on many factors
  - Time complexity is about how time grows with input size
- Analysis of brute-force algorithm:
  - Computing $P(q_1, \ldots, q_n, s_1, \ldots, s_n)$ requires $2n$ multiplications
  - There are $|Q|^n$ possible state sequences of length n
  - If each multiplication takes c time, $T(n) = c \cdot 2n \cdot |Q|^n$
- In the long run, only the fastest growing factor matters:

$$T(n) = O(|Q|^n)$$


Big-O Complexity
1000
900
800
700
600
500
400
300
200
100
0
Operations
0
10
20
30
40
50
60
70
80
90
100
Elements
O(1)
O(logn)
O(n)
O(nlogn)
O(n^2)
O(2^n)
O(n!)

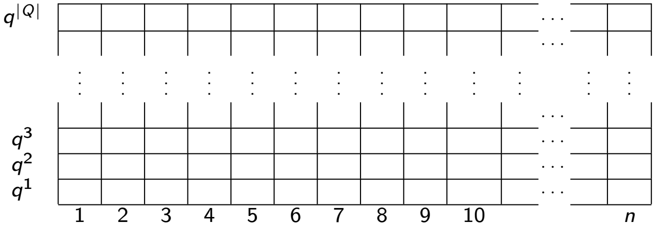
Dynamic Programming

- What is the problem really?
  - The number $|\Omega|^n$ of sequences grows exponentially
  - But the sequences have overlapping subsequences
  - The brute-force algorithm does a lot of unnecessary work
- Key: solution of size n contains solution of size $n-1$

$$\operatorname{argmax}_{q_1,\ldots,q_n} P(q_1,\ldots,q_n,s_1,\ldots,s_n) =$$
$$\operatorname{argmax}_{q_n} \operatorname{argmax}_{q_1,\ldots,q_{n-1}} P(q_1,\ldots,q_{n-1},s_1,\ldots,s_{n-1})P(q_n|q_{n-1})P(s_n|s_n)$$

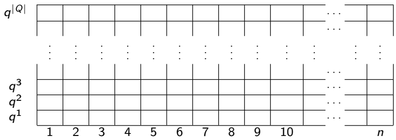
- We can use dynamic programming
  - Create a table for storing partial results
  - Make sure that partial results are available when needed
  - Avoid recomputing the same result more than once

The Trellis



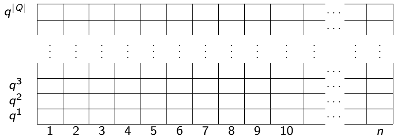
- For HMMs the table is known as the trellis:
  - Every rows corresponds to a state $q \in Q$
  - Every column corresponds to a position i $(1 \leq i \leq n)$
  - The cell q_i represents the best way to reach q at i

The Viterbi Algorithm



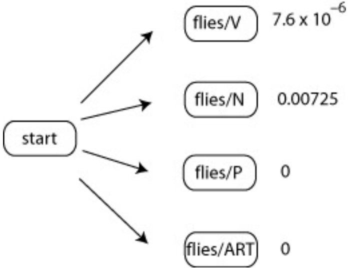
- Goal:
  - Find best cell in column n = best way to reach any state at n
- Algorithm:
  - Fill the table from left to right, column by column
  - For each cell q_i:
    - Add q to all best paths in column $i-1$
    - Keep the one with highest probability
- We need one trellis for probabilities (A) and one for paths (B)

The Viterbi Algorithm

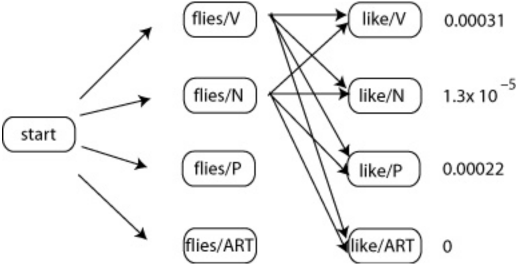


for $i = 1$ **to** n
    for $q = q^1$ **to** $q^{|Q|}$
        for $q' = q^1$ **to** $q^{|Q|}$
            $p \leftarrow A[q', i-1]P(q|q')P(s_i|q)$
            if $p > A[q, i]$ **then**
                $A[q, i] \leftarrow p$
                $B[q, i] \leftarrow q'$
$q^* \leftarrow \max_q A[q, n]$
return $B[q^*, n], B[B[q^*, n], n-1], \ldots$

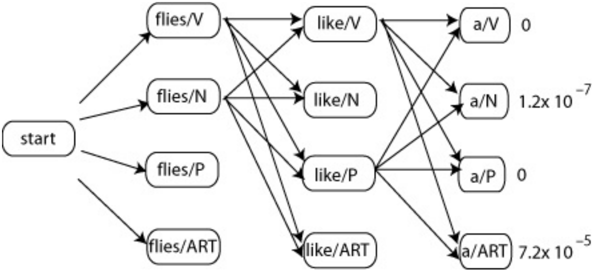
Viterbi Example



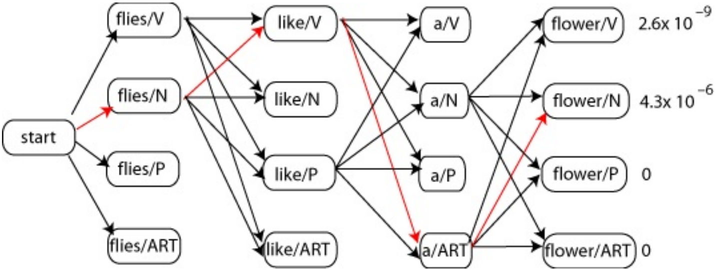
Viterbi Example



Viterbi Example



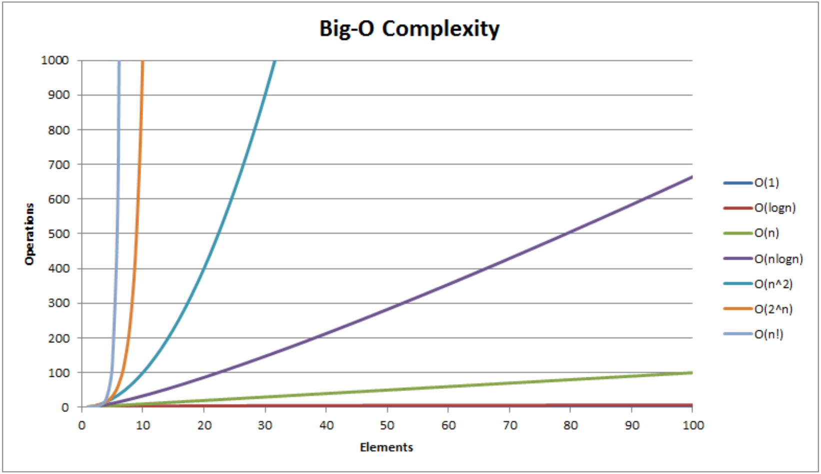
Viterbi Example



Time Complexity

- Analysis of Viterbi algorithm:
  - Filling one cell requires $2|Q|$ multiplications
  - There are $|Q|$ cells in each column
  - There are n columns in the trellis
  - If each multiplication takes c time, $T(n) = 2n|Q|^2$
- Worst-case complexity:

$$T(n) = O(n|Q|^2)$$


Big-O Complexity
1000
900
800
700
600
500
400
300
200
100
0
Operations
0
10
20
30
40
50
60
70
80
90
100
Elements
O(1)
O(logn)
O(n)
O(nlogn)
O(n^2)
O(2^n)
O(n!)

Learning

- Supervised learning:
  - Given a tagged training corpus, we can estimate parameters using (smoothed) relative frequencies
  - This maximizes the joint likelihood of states and signals
  - Transition probabilities can be smoothed like *n*-gram models
  - Emission probabilities need special tricks for unknown words
- Weakly supervised learning:
  - Given a lexicon and an untagged training corpus, we can use Expectation-Maximization to estimate parameters
  - This attempts to maximize the marginal likelihood of the signals (because states are hidden/unobserved)

Maximum Likelihood Estimation

- With labeled data $D = \{(x_1, y_1), \ldots, (x_n, y_n)\}$, we set parameters θ to maximize the joint likelihood of input and output:

$$\underset{\theta}{\operatorname{argmax}} \prod_{1=i}^{n} P_\theta(y_i) P_\theta(x_i | y_i)$$

- With unlabeled data $D = \{x_1, \ldots, x_n\}$, we can instead set parameters θ to maximize the marginal likelihood of the input:

$$\underset{\theta}{\operatorname{argmax}} \prod_{1=i}^{n} \sum_{y \in \Omega_Y} P_\theta(y) P_\theta(x_i | y)$$

- In this case, Y is a hidden variable that is marginalized out

Expectation-Maximization

- Joint MLE is easy – just use relative frequencies
- Marginal MLE is hard – no closed form solution

$$\operatorname*{argmax}_{\theta} \prod_{i=1}^{n} \sum_{y \in \Omega_Y} P_\theta(y) P_\theta(x_i|y) = \ ?$$

- What can we do?
  - Use numerical approximation methods
  - Most common approach: Expectation-Maximization (EM)

Expectation-Maximization

- Basic observations:
  - If we know $f_N(x, y)$, we can find the MLE θ.
  - If we know $f_N(x)$ and the MLE θ, we can derive $f_N(x, y)$.
  - If $P_\theta(y|x) = p$ and $f_N(x) = n$, then $f_N(x, y) = np$.
- Basic idea behind EM:
  1. Start by guessing θ
  2. Compute expected counts $E[f_N(x, y)] = P_\theta(y|x) f_N(x)$
  3. Find MLE θ given expectation $E[f_N(x, y)]$
  4. Repeat steps 2 and 3 until convergence

Supervised Learning

Lexicon:

eat	V
fish	N V
drink	N V
beer	N V

Corpus:

<s> eat/V fish/N </s>
<s> drink/V beer/N </s>

Freqs	N	V	eat	fish	drink	beer
<s>	0	2				
N	0	0	0	1	0	1
V	2	0	1	0	1	0

Probs	N	V	eat	fish	drink	beer
<s>	0.0	1.0				
N	0.5	0.5	0.0	0.5	0.0	0.5
V	1.0	0.0	0.5	0.0	0.5	0.0

EM: First Guess

Lexicon:

eat	V
fish	N V
drink	N V
beer	N V

Corpus:

<s> eat fish </s>
<s> drink beer </s>

Freqs	N	V	eat	fish	drink	beer
<s>	?	?				
N	?	?	?	?	?	?
V	?	?	?	?	?	?

Probs	N	V	eat	fish	drink	beer
<s>	0.5	0.5				
N	0.5	0.5	0.0	0.33	0.33	0.33
V	0.5	0.5	0.33	0.33	0.33	0.0

EM: First E-Step

Lexicon:		
	eat	V
	fish	N V
	drink	N V
	beer	N V

Corpus:

<s> eat/V fish/N </s> : 0.5
<s> eat/V fish/V </s> : 0.5
<s> drink/N beer/N </s> : 0.5
<s> drink/V beer/N </s> : 0.5

Freqs	N	V	eat	fish	drink	beer
<s>	?	?				
N	?	?	?	?	?	?
V	?	?	?	?	?	?

Probs	N	V	eat	fish	drink	beer
<s>	0.5	0.5				
N	0.5	0.5	0.0	0.33	0.33	0.33
V	0.5	0.5	0.33	0.33	0.33	0.0

EM: First E-Step

Lexicon:		
	eat	V
	fish	N V
	drink	N V
	beer	N V

Corpus:

<s> eat/V fish/N </s> : 0.5
<s> eat/V fish/V </s> : 0.5
<s> drink/N beer/N </s> : 0.5
<s> drink/V beer/N </s> : 0.5

Freqs	N	V	eat	fish	drink	beer
<s>	0.5	1.5				
N	0.5	0	0	0.5	0.5	1
V	1	0.5	1	0.5	0.5	0

Probs	N	V	eat	fish	drink	beer
<s>	0.5	0.5				
N	0.5	0.5	0.0	0.33	0.33	0.33
V	0.5	0.5	0.33	0.33	0.33	0.0

EM: First M-Step

Lexicon:

eat	V
fish	N V
drink	N V
beer	N V

Corpus:

<s> eat fish </s>
<s> drink beer </s>

Freqs	N	V	eat	fish	drink	beer
<s>	0.5	1.5				
N	0.5	0	0	0.5	0.5	1
V	1	0.5	1	0.5	0.5	0

Probs	N	V	eat	fish	drink	beer
<s>	0.25	0.75				
N	1.0	0.0	0.0	0.25	0.25	0.5
V	0.67	0.33	0.5	0.25	0.25	0.0

EM: Second E-Step

Lexicon:

eat	V
fish	N V
drink	N V
beer	N V

Corpus:

<s> eat/V fish/N </s> : 0.86
<s> eat/V fish/V </s> : 0.14
<s> drink/N beer/N </s> : 0.33
<s> drink/V beer/N </s> : 0.67

Freqs	N	V	eat	fish	drink	beer
<s>	0.5	1.5				
N	0.5	0	0	0.5	0.5	1
V	1	0.5	1	0.5	0.5	0

Probs	N	V	eat	fish	drink	beer
<s>	0.25	0.75				
N	1.0	0.0	0.0	0.25	0.25	0.5
V	0.67	0.33	0.5	0.25	0.25	0.0

EM: Second E-Step

Lexicon:

eat	V
fish	N V
drink	N V
beer	N V

Corpus:

<s> eat/V fish/N </s> : 0.86
<s> eat/V fish/V </s> : 0.14
<s> drink/N beer/N </s> : 0.33
<s> drink/V beer/N </s> : 0.67

Freqs	N	V	eat	fish	drink	beer
<s>	0.33	1.67				
N	0.33	0	0	0.86	0.33	1
V	1.19	0.14	1	0.14	0.67	0

Probs	N	V	eat	fish	drink	beer
<s>	0.25	0.75				
N	1.0	0.0	0.0	0.25	0.25	0.5
V	0.67	0.33	0.5	0.25	0.25	0.0

EM: Second M-Step

Lexicon:		
	eat	V
	fish	N V
	drink	N V
	beer	N V

Corpus:

<s> eat fish </s>
<s> drink beer </s>

Freqs	N	V	eat	fish	drink	beer
<s>	0.33	1.67				
N	0.33	0	0	0.86	0.33	1
V	1.19	0.14	1	0.14	0.67	0

Probs	N	V	eat	fish	drink	beer
<s>	0.17	0.83				
N	1.0	0.0	0.0	0.39	0.15	0.46
V	0.89	0.11	0.55	0.08	0.37	0.0

EM: Third E-Step

Lexicon:

eat	V
fish	N V
drink	N V
beer	N V

Corpus:

<s> eat/V fish/N </s> : 0.98
<s> eat/V fish/V </s> : 0.02
<s> drink/N beer/N </s> : 0.09
<s> drink/V beer/N </s> : 0.91

Freqs	N	V	eat	fish	drink	beer
<s>	0.33	1.67				
N	0.33	0	0	0.86	0.33	1
V	1.19	0.14	1	0.14	0.67	0

Probs	N	V	eat	fish	drink	beer
<s>	0.17	0.83				
N	1.0	0.0	0.0	0.39	0.15	0.46
V	0.89	0.11	0.55	0.08	0.37	0.0

EM: Third E-Step

Lexicon:

eat	V
fish	N V
drink	N V
beer	N V

Corpus:

<s> eat/V fish/N </s> : 0.98
<s> eat/V fish/V </s> : 0.02
<s> drink/N beer/N </s> : 0.09
<s> drink/V beer/N </s> : 0.91

Freqs	N	V	eat	fish	drink	beer
<s>	0.09	1.91				
N	0.09	0	0	0.98	0.09	1
V	1.89	0.02	1	0.02	0.91	0

Probs	N	V	eat	fish	drink	beer
<s>	0.17	0.83				
N	1.0	0.0	0.0	0.39	0.15	0.46
V	0.89	0.11	0.55	0.08	0.37	0.0

EM: Third M-Step

Lexicon:

eat	V
fish	N V
drink	N V
beer	N V

Corpus:

<s> eat fish </s>
<s> drink beer </s>

Freqs	N	V	eat	fish	drink	beer
<s>	0.09	1.91				
N	0.09	0	0	0.98	0.09	1
V	1.89	0.02	1	0.02	0.91	0

Probs	N	V	eat	fish	drink	beer
<s>	0.05	0.95				
N	1.0	0.0	0.0	0.47	0.04	0.48
V	0.99	0.01	0.52	0.01	0.47	0.0

Convergence

- EM is guaranteed to converge to a local maximum of the likelihood function
  - A local maximum may not be the global maximum
  - Even if we reach the global maximum, it may not be the same as for the supervised model (Why?)
- In general, EM is quite sensitive to initialization

EM for Hidden Markov Models

- Computing expectations:

$$\begin{aligned} E[f_N(q,q')] &= \sum_{i=1}^{n} P(s_1,\ldots,s_n,q_i=q,q_{i-1}=q') \\ E[f_N(s,q)] &= \sum_{i=1}^{n} P(s_1,\ldots,s_{i-1},s_i=s,\ldots,s_n,q_i=q) \end{aligned}$$

- Difficulty:
  - Summing over all possible state sequences
  - The number $|\Omega|^n$ of sequences grows exponentially
- Sounds familiar?
  - We can use dynamic programming again
  - The forward-backward algorithm

Summary

- Markov models are probabilistic sequence models used for part-of-speech tagging and many similar problems
- They can be trained from labeled data using relative frequency estimation or from unlabeled data and a lexicon using EM
- Thanks to the Markov assumptions, computation can be made efficient using dynamic programming:
  - Most probable state sequence – Viterbi
  - Probability of signal sequence – Forward or Backward
  - Expectations for EM – Forward-Backward