INF4820

Hidden Markov Models
The Forward Algorithm
The Viterbi Algorithm

Erik Velldal

University of Oslo

Oct. 06, 2009

Topics for Today

- Quick recap from last time: POS-tagging viewed as Bayesian classification.
- Formal specification of an HMM; $\langle Q, q_0, q_F, A, B \rangle$
- Dynamic Programming
  - The Forward algorithm for computing the HMM probability of an observed sequence of words.
  - The Viterbi algorithm for computing the HMM probability of an unobserved sequence of tags.
- Evaluating a tagger on test data

HMM Tagging as Bayesian Classification

- Given an observed sequence of words $O = (o_1, \ldots, o_T)$, we want to find the most probable sequence of tags $Q = (q_1, \ldots, q_T)$.
- Applying Bayes' Rule, we can state our search problem as

$$\hat{q}_1^T = \arg\max_{q_1^T} P(q_1^T | o_1^T) = \arg\max_{q_1^T} \frac{P(o_1^T | q_1^T) P(q_1^T)}{P(o_1^T)}$$

$$= \arg\max_{q_1^T} P(o_1^T | q_1^T) P(q_1^T)$$

HMM Tagging as Bayesian Classification

- Given an observed sequence of words $O = (o_1, \ldots, o_T)$, we want to find the most probable sequence of tags $Q = (q_1, \ldots, q_T)$.
- Applying Bayes' Rule, we can state our search problem as

$$\hat{q}_1^T = \arg\max_{q_1^T} P(q_1^T | o_1^T) = \arg\max_{q_1^T} \frac{P(o_1^T | q_1^T) P(q_1^T)}{P(o_1^T)}$$
$$= \arg\max_{q_1^T} P(o_1^T | q_1^T) P(q_1^T)$$

- This approach can also be viewed as Noisy-Channel Modeling:
  - Shannon's metaphor: q_1^T is the result of transmitting o_1^n through a noisy channel, i.e. o_1^T is a *scrambled* version of q_1^T.
  - Our task is to model the noise so we can decode the distorted sequence and recover the original source.

A Few Simplifying Assumptions

- Assume the Markov property for $P(q_1^T)$:

$$P(q_1^T) = P(q_1)P(q_2|q_1)P(q_3|q_1, q_2) \dots P(q_n|q_1^{n-1})$$
$$\approx \prod_i P(q_i|q_{i-1})$$

- Two more simplifying assumptions regarding $P(o_1^T|q_1^T)$.
  - Each word is conditionally independent of the other words given the tags, and each word is conditionally independent of all tags but its own:

$$P(o_1^T|q_1^T) = P(o_1|q_1^T)P(o_2|o_1, q_1^T) \dots P(o_n|o_1^{n-1}, q_1^T)$$
$$\approx \prod_i P(o_i|q_i)$$

A Few Simplifying Assumptions

- Assume the Markov property for $P(q_1^T)$:

$$P(q_1^T) = P(q_1)P(q_2|q_1)P(q_3|q_1, q_2) \ldots P(q_n|q_1^{n-1})$$
$$\approx \prod_i P(q_i|q_{i-1})$$

- Two more simplifying assumptions regarding $P(o_1^T|q_1^T)$.
  - Each word is conditionally independent of the other words given the tags, and each word is conditionally independent of all tags but its own:

$$P(o_1^T|q_1^T) = P(o_1|q_1^T)P(o_2|o_1, q_1^T) \ldots P(o_n|o_1^{n-1}, q_1^T)$$
$$\approx \prod_i P(o_i|q_i)$$

- We can now finally formulate the classification problem as:

$$\hat{q}_1^T = \arg\max_{q_1^T} P(q_1^T|o_1^T) \approx \arg\max_{q_1^T} \prod_i P(o_i|q_i)P(q_i|q_{i-1})$$

Supervised Training

Tag Transition Probabilities

Assuming we have a training corpus of previously tagged text, the MLE can be computed from the counts of observed tags:

$$P(q_i|q_{i-1}) = \frac{C(q_{i-1}, q_i)}{C(q_{i-1})}$$

Supervised Training

Tag Transition Probabilities

Assuming we have a training corpus of previously tagged text, the MLE can be computed from the counts of observed tags:

$$P(q_i|q_{i-1}) = \frac{C(q_{i-1}, q_i)}{C(q_{i-1})}$$

Word Likelihoods (AKA Emission Probabilities)

Computed from relative frequencies in the same way: $P(o_i|q_i) = \frac{C(q_i, o_i)}{C(q_i)}$

Supervised Training

Tag Transition Probabilities

Assuming we have a training corpus of previously tagged text, the MLE can be computed from the counts of observed tags:

$$P(q_i|q_{i-1}) = \frac{C(q_{i-1}, q_i)}{C(q_{i-1})}$$

Word Likelihoods (AKA Emission Probabilities)

Computed from relative frequencies in the same way: $P(o_i|q_i) = \frac{C(q_i, o_i)}{C(q_i)}$

Sparse Data Problem

The issues related to MLE / smoothing that we discussed for n-gram models also applies here...

Formal Specification of an HMM: $\langle Q, q_0, q_F, A, B \rangle$

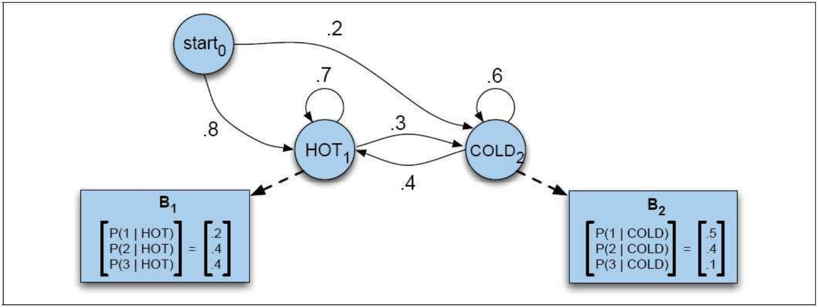
- Q: A set of states $\{q_1, \ldots, q_N\}$
- $B = b_i(o_t)$: Emission probabilities (or observation likelihoods). Represents the probability of state q_i generating observation o_t.
- q_0, q_F: Start state / final state (not associated with observations).
- $A = \begin{pmatrix} a_{11} & \ldots & a_{1N} \\ \vdots & \ddots & \vdots \\ a_{N1} & \ldots & a_{NN} \end{pmatrix}$: Transition probability table.
  - An element a_{ij} records the probability of moving from q_i to q_j, and $\forall i \sum_{j=1}^{N} a_{ij} = 1$.
  - In addition to the ordinary transition probabilities a_{11} through a_{NN}, we also assume a set of probabilities $a_{01}, \ldots, a_{0N}$ out of the start state q_0, and a set of probabilities $a_{1F}, \ldots, a_{NF}$ into the final state q_F.

Likelihood and Decoding

- Let $O = (o_1, o_2, \ldots, o_T)$ be a sequence of observations, where each o_i is member of some vocabulary $V = \{v_1, \ldots, v_L\}$.
- Then, for a given HMM, there are two problems we want to solve:
  1. What is the likelihood of O? (Likelihood)
  2. What is the most probable underlying sequence of hidden variables $Q = (q_1, q_2, \ldots, q_T)$? (Decoding)

The Jason Eisner Ice Cream Problem

Given a sequence of observations O, each o_i corresponding to the number $(1$, 2 or $3)$ of ice creams eaten on a given day, figure out the correct "hidden" sequence Q of weather states (HOT or COLD) which caused Jason to eat the ice cream. (Taken from Jurafsky & Martin, 2009)



Computing the Likelihood (Take One)

- Let's start by assuming that we have actually observed *both* O and Q. The joint probability $P(O, Q)$ can be computed as

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

Computing the Likelihood (Take One)

- Let's start by assuming that we have actually observed *both* O and Q. The joint probability $P(O, Q)$ can be computed as

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

- Problem: We don't actually know the state sequence Q.
- Instead, compute the sum over all possible state sequences, weighted by their probability:

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

Computing the Likelihood (Take One)

- Let's start by assuming that we have actually observed *both* O and Q. The joint probability $P(O, Q)$ can be computed as

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

- Problem: We don't actually know the state sequence Q.
- Instead, compute the sum over all possible state sequences, weighted by their probability:

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

- More problems: For N possible states and T observations, there are a total of N^T possible state sequences. Exponential computational complexity, $O(N^T T)$.

Computing the Likelihood (Take Two)

The Forward Algorithm

- Relies on dynamic programming to reduce the complexity to $O(N^2T)$.
- The trick is to store and reuse the results of intermediate and partial computations.
- This is done by recursively filling the cells of a s.c. trellis structure.

Computing the Likelihood (Take Two)

The Forward Algorithm

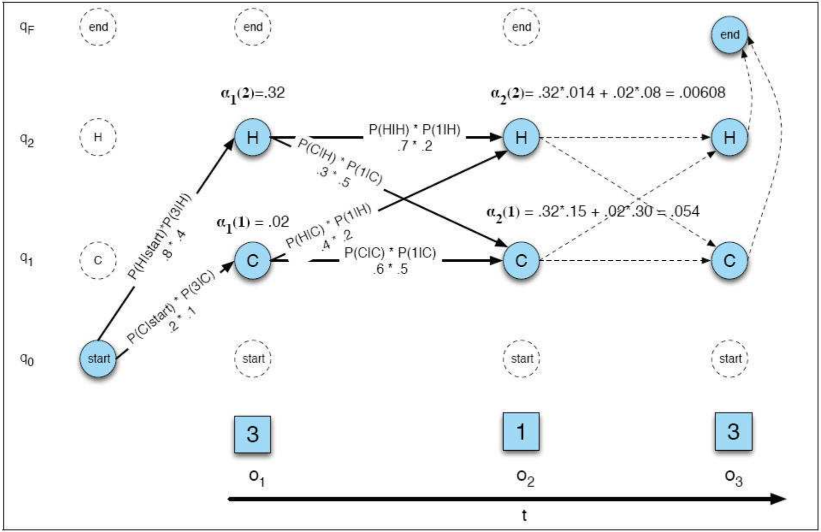
- Relies on dynamic programming to reduce the complexity to $O(N^2T)$.
- The trick is to store and reuse the results of intermediate and partial computations.
- This is done by recursively filling the cells of a s.c. trellis structure.
- A cell $\alpha_t(j)$ in the forward trellis stores the probability of being in state q_j after seeing the t first observations.

$$\alpha_t(j) = P(o_1, \ldots, o_t, q_t = j)$$

- The value of each cell $\alpha_t(j)$ is computed by summing over the probabilities of all possible paths that could lead to that cell.

$$\alpha_t(j) = \sum_{i=1}^{N} \alpha_{t-1}(i)\, a_{ij}\, b_j(o_t)$$

The Forward Trellis for the Ice Cream Problem



The Forward Algorithm

1. Initialization. For each j from 1 to N:

$$\alpha_1(j) = a_{0j}\, b_j(o_1)$$

2. Recursion. For each t from 2 to T, for each j from 1 to N:

$$\alpha_t(j) = \sum_{i=1}^{N} \alpha_{t-1}(i)\, a_{ij}\, b_j(o_t)$$

3. Termination.

$$P(O) = \alpha_T(F) = \sum_{i=1}^{N} \alpha_T(i)\, a_{iF}$$

Decoding

- Extracting the most probable sequence of hidden variables $Q = (q_1, \ldots, q_T)$ considered to be the source of a given sequence of observations $O = (o_1, \ldots, o_T)$.
- For the *ice cream problem* this amounts to finding the most probable sequence of weather states, given what we've seen of Jason's ice cream eating.

Decoding

- Extracting the most probable sequence of hidden variables $Q = (q_1, \ldots, q_T)$ considered to be the source of a given sequence of observations $O = (o_1, \ldots, o_T)$.
- For the *ice cream problem* this amounts to finding the most probable sequence of weather states, given what we've seen of Jason's ice cream eating.
- Just as for the likelihood, the naive approach (computing the probability of each possible state sequence) is not computationally tractable due the exponentially large number of state sequences.
- Again we can reduce the complexity by using a trellis-based dynamic programming technique: The Viterbi algorithm.

The Viterbi Trellis

- Let each cell of the Viterbi trellis $v_t(j)$ represent the probability of our HMM being in state q_j after seeing the first sub-sequence of observations $o_1 \ldots, o_t$ and passing through the most probable sequence of states $q_1, \ldots, q_{t-1}$.

$$v_t(j) = \max_{(q_1,\ldots,q_{t-1})} P(o_1, \ldots, o_t, q_1, \ldots, q_{t-1}, q_t = j)$$

- Moving forward through the trellis, each cell is updated recursively, based on the values of the previously computed cells:

$$v_t(j) = \max_{i=1}^{N} v_{t-1}(i)\, a_{ij}\, b_j(o_t)$$

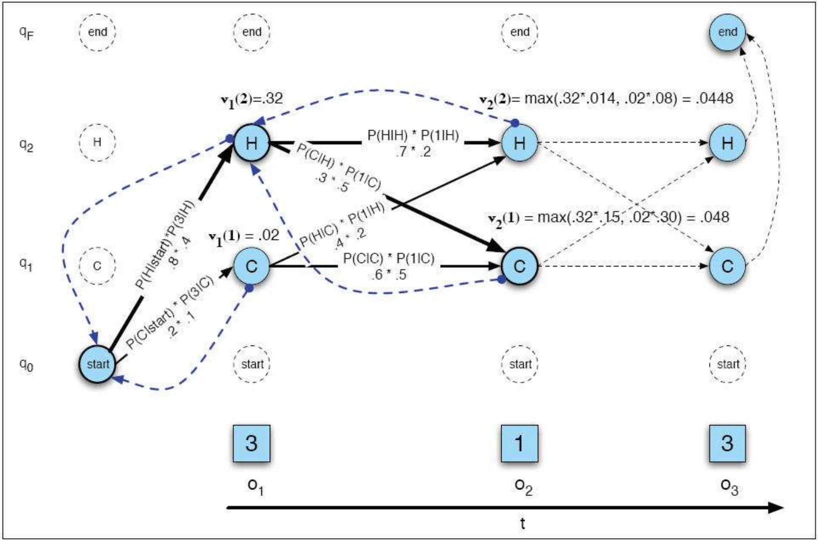
The Backtrace

- So far the Viterbi algorithm is pretty much identical to the Forward algorithm, except that each cell stores the *max* probability (instead of the *sum*) of all the possible paths so far.

The Backtrace

- So far the Viterbi algorithm is pretty much identical to the Forward algorithm, except that each cell stores the *max* probability (instead of the *sum*) of all the possible paths so far.
- But, since we also want to extract the actual state sequence that corresponds to the most probable path, we need to keep track of our path through the trellis.
- Let $bt_t(j)$ denote the backtrace pointer from state q_j at time t, back to the previous node q_{t-1} of the most probable subpath to this node.

The Viterbi Trellis for the Ice Cream Problem



The Viterbi Algorithm

1. Initialization. For each j from 1 to N:

$$v_1(j) = a_{0j}\, b_j(o_1) \text{ and}$$
$$bt_1(j) = 0$$

2. Recursion. For each t from 2 to T, for each j from 1 to N:

$$v_t(j) = \max_{i=1}^{N} v_{t-1}(i)\, a_{ij}\, b_j(o_t) \text{ and}$$
$$bt_t(j) = \arg\max_{i=1}^{N} v_{t-1}(i)\, a_{ij}$$

3. Termination

$$v_T(F) = \max_{i=1}^{N} v_T(i)\, a_{iF} \text{ and}$$
$$bt_T(F) = \arg\max_{i=1}^{N} v_T(i)\, a_{iF}$$

(A Practical Tip)

- When multiplying many small probabilities, we risk getting values that are too close to zero to be represented: Underflow.

(A Practical Tip)

- When multiplying many small probabilities, we risk getting values that are too close to zero to be represented: Underflow.
- It is often helpful to work in "log-space":

$$\log(\max f) = \max(\log f)$$

- Reduces multiplication to addition.

$$\log \prod_i P_i = \sum_i \log P_i$$

(A Practical Tip)

- When multiplying many small probabilities, we risk getting values that are too close to zero to be represented: Underflow.
- It is often helpful to work in "log-space":

$$\log(\max f) = \max(\log f)$$

- Reduces multiplication to addition.

$$\log \prod_i P_i = \sum_i \log P_i$$

- (But beware that $\log(\sum f) \neq \sum(\log f)$, so for situations like the Forward algorithm we can't use the log-space trick. Might want to use scaling instead.)

*Un*supervised Training

- So far we have assumed that we can estimate the relevant probabilities directly from annotated training data.
  - This amounts to what we call supervised training.
- However, we don't always have this luxury.

*Un*supervised Training

- So far we have assumed that we can estimate the relevant probabilities directly from annotated training data.
  - This amounts to what we call supervised training.
- However, we don't always have this luxury.
- HMMs can also be trained unsupervised.
  - The Forward-Backward algorithm is a dynamic programming technique for iteratively computing the probabilities based only on the observations and initial sets of possible states (e.g. from lexicon look-up, in the case of POS tagging).
  - Based on the more general Expectation Maximization (EM) algorithm.

Evaluation

- Using a manually labeled test set as our gold standard, we can compute the accuracy of our model: The percentage of tags in test set that the tagger gets right.

Evaluation

- Using a manually labeled test set as our gold standard, we can compute the accuracy of our model: The percentage of tags in test set that the tagger gets right.
- Compare the accuracy to some reference models: an upper-bound and a baseline.
  - An upper-bound ceiling can be based on e.g. how well humans would do on the task or by assuming an “oracle”.
  - A lower-bound baseline can be based on the accuracy expected by e.g. random choice, always picking the tags with the highest frequency, or applying a unigram model.
- Standard hypothesis tests can be applied to test the statistical significance of any differences.