Sequence Labeling

• Many information extraction problems can be formulated as sequence labeling tasks. Sequential classifiers assign a label to each item in a sequence.

• Sequence labeling methods are appropriate for problems where the label of an item depends on other (typically nearby) items in the sequence.

• Example tasks: part-of-speech tagging, syntactic chunking, named entity recognition, opinion extraction.

• A naive approach would be to consider all possible label sequences and choose the best one. But that is too expensive, we need more efficient methods.

Markov Chains

• A Markov Chain is a finite-state automaton that has a probability associated with each transition (arc) and the input sequence uniquely defines the transitions that can be taken.

• In a first-order Markov chain, the probability of a state depends only on the previous state. The Markov Assumption is:

\[ P(q_i \mid q_1 \ldots q_{i-1}) = P(q_i \mid q_{i-1}) \]

where \( q_i \in Q \) are states:

• The probabilities of all outgoing arcs of a state must sum to 1.

• The Markov chain can be traversed to compute the probability of a sequence of observable events.

A Markov Chain is defined by:

\[ Q = q_1 q_2 \ldots q_n \quad \text{a set of } N \text{ states} \]
\[ A = a_{00} a_{01} \ldots a_{n1} \ldots a_{nm} \quad \text{a transition probability matrix } A, \text{ each } a_{ij} \text{ representing the probability of moving from state } i \text{ to state } j, \text{ s.t. } \sum_{j=1}^{m} a_{ij} = 1 \quad \forall i \]
\[ q_0, q_F \quad \text{a special start state and end (final) state that are not associated with observations} \]

NOTE: not all formulations have an explicit end state.

Example: this Markov chain can compute the probability of a sequence of part-of-speech tags.

Note: no words are modeled!
Hidden Markov Models

- A Hidden Markov Model (HMM) is used to find the most likely sequence of hidden events given a sequence of observable events.
- The input tokens are the observed events.
- The class labels (states) are the hidden events.

For example:
- given a sequence of words, find the most likely sequence of part-of-speech tags or named entity labels
- given acoustic sounds (speech), find the most likely sequence of words

A HMM is defined by:

- \( Q = q_1q_2 \ldots q_N \) a set of \( N \) states
- \( A = a_{ij} \) a transition probability matrix, each \( a_{ij} \) representing the probability of moving from state \( i \) to state \( j \), s.t. \( \sum_{j=1}^{N} a_{ij} = 1 \) \( \forall i \)
- \( O = o_1o_2 \ldots o_T \) a sequence of \( T \) observations, each one drawn from a vocabulary \( V = v_1, v_2, \ldots, v_V \)
- \( B = b_i(o_t) \) a sequence of observation likelihoods, also called emission probabilities, each expressing the probability of an observation \( o_t \) being generated from a state \( i \)
- \( q_0, q_F \) a special start state and end (final) state that are not associated with observations, together with transition probabilities \( a_{01}a_{02}\ldots a_{0n} \) out of the start state and \( a_{1F}a_{2F}\ldots a_{nF} \) into the end state

HMM Example

![HMM Diagram](image)

A hidden Markov model for relating numbers of ice creams eaten by Jason (the observations) to the weather (H or C, the hidden variables).

Using HMMs

We can use an HMM in two ways:

- **Likelihood**: given a sequence of observed events, compute the probability of the observed events, \( P(O) \).
- **Decoding**: given a sequence of observed events, find the most likely sequence of hidden events (labels).

As an example, consider an HMM where words are the observed events and part-of-speech tags are the hidden events:

- **Likelihood**: given a sentence (sequence of words), compute the probability of the sentence.
- **Decoding**: given a sentence, find the most likely sequence of part-of-speech tags for the words.
Decoding: The Viterbi Algorithm

- Given a sequence of observed events $O = o_1, o_2, ... o_N$, find the most probable sequence of states, $Q = q_1, q_2, ... q_N$. For NLP, the observations are usually words and the states are class labels.

- Computing the probability of every possible state sequence by enumerating them would be expensive: $N^T$, where $T = |Q|$.

- The Viterbi algorithm uses a dynamic programming solution, which is $O(N \times T^2)$.

- The algorithm sequentially processes the input, sweeping through all possible states for each word and identifying the best prior state sequence leading to each one.

- Identifying the best prior sequence leading to the current word is sufficient because of the Markov assumption.

The Viterbi Algorithm

```
function VITERBI(observations of len T, state-graph of len N) returns best-path
create a path probability matrix viterbi[N+2,T]
for each state s from 1 to N do
  viterbi[s,1] ← $a_{0,s} \times b_s(o_1)$
  backpointer[s,1] ← 0
for each time t from 2 to T do
  for each state s from 1 to N do
    viterbi[s,t] ← max (s' = 1 to N) viterbi[s',t-1] * $a_{s',s} \times b_s(o_t)$
    backpointer[s,t] ← arg max (s' = 1 to N) viterbi[s',t-1] * $a_{s',s}$
  viterbi[qf,T] ← max (s = 1 to N) viterbi[s,T] * $a_{s,qf}$
  backpointer[qf,T] ← arg max (s = 1 to N) viterbi[s,T] * $a_{s,qf}$
return the backtrack path by following backpointers to states back in time from backpointer[qf,T]
```

Viterbi Definitions

- $v_{i-1}(i)$: the previous Viterbi path probability from the previous time step
- $a_{ij}$: the transition probability from previous state $q_i$ to current state $q_j$
- $b_j(o_t)$: the state observation likelihood of the observation symbol $o_t$ given the current state $j$

For part-of-speech tagging, we used the following:

- $a_{ij} = P(\text{tag}_j | \text{tag}_i)$ which were POS bigram probabilities
- $b_j(o_t) = P(\text{word}_t | \text{tag}_j)$ which were lexical generation probabilities

The Likelihood of an Input Sequence

- The likelihood of an input sequence (observed events) given a specific sequence of hidden events is:
  
  \[ P(O|Q) = \prod_{i=1}^{T} P(o_i | q_i) \]

- The joint likelihood of an input sequence (observed events) and a specific sequence of hidden events is:

  \[ P(O,Q) = P(O|Q) \times P(Q) = \prod_{i=1}^{n} P(o_i | q_i) \times \prod_{i=1}^{n} P(q_i | q_{i-1}) \]

- So we can compute the likelihood of the input sequence by summing over all sequences of hidden states:

  \[ P(O) = \sum_Q P(O,Q) = \sum_Q P(O|Q)P(Q) \]
The Forward Probability

- The **forward probability**, \( \alpha_t(j) \), is the probability of being in state \( j \) after seeing the first \( t \) observed events.

- Intuitively, it is the sum of the probabilities over all possible state sequences that could lead to state \( j \) for \( o_1 ... o_t \).

\[
\alpha_t(j) = P(o_1, o_2, ... o_t, q_t = j)
\]

- We sequentially process the input, sweeping over each possible state for \( o_t \). For each \( q_t \), we compute the sum over all possible prior states of: (the forward probability of the prior state) * (the transition between the states) * (the emission probability of \( o_t \)).

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

The Backward Algorithm

- The **backward probability**, \( \beta_t(i) \), is the probability of seeing observations \( t+1 \) to \( T \), given that you’re in state \( i \) at time \( t \).

- Intuitively, it is the sum of the probabilities over all possible state sequences that could follow from state \( i \) for \( o_{t+1} ... o_T \).

\[
\beta_t(i) = P(o_{t+1}, o_{t+2}, ... o_T \mid q_t = i)
\]

- We sequentially process the input backwards from \( q_F \), sweeping over each possible preceding state. For each \( q_t \), we compute the sum over all following states of: (the transition between the states) * (the emission probability of \( o_{t+1} \)) * (the backward probability of the following state).

\[
\beta_t(i) = \sum_{j=1}^{N} a_{ij} b_j(o_{t+1}) \beta_{t+1}(j)
\]

Computing Backward Probabilities

\[
\beta_t(i) = P(o_{t+1}, o_{t+2} ... o_T \mid q_t = i, \lambda)
\]

It is computed inductively in a similar manner to the forward algorithm.

1. **Initialization:**

\[
\beta_T(i) = a_{q_F} \quad 1 \leq i \leq N
\]

2. **Recursion** (again since states 0 and \( q_F \) are non-emitting):

\[
\beta_t(i) = \sum_{j=1}^{N} a_{ij} b_j(o_{t+1}) \beta_{t+1}(j) \quad 1 \leq i \leq N, 1 \leq t < T
\]

3. **Termination:**

\[
P(O \mid \lambda) = \alpha_T(q_F) = \beta_T(q_0) = \sum_{j=1}^{N} a_{0j} b_j(o_1) \beta_t(j)
\]

Note that this is similar to the Viterbi algorithm, but sums over the products, instead of taking the Max.
The Forward-Backward Algorithm

• The forward-backward algorithm (also known as the Baum-Welch algorithm) can be used to learn the parameters (transition probabilities $A$ and emission probabilities $B$) of a HMM.

• We typically estimate probabilities by counting (maximum likelihood estimate), but the hidden events may not be available.

• The key idea behind the forward-backward algorithm is that we can iteratively estimate the counts and probabilities. Beginning with some initial values, the probabilities gradually evolve based on the data.

Estimating Transition Probabilities

To estimate the $A$ values, intuitively we want:

$$\hat{a}_{ij} = \frac{\text{expected number of transitions from state } i \text{ to state } j}{\text{expected number of transitions from state } i}$$

We define the probability of being in state $i$ at time $t$, and state $j$ at time $t+1$:

$$\xi_{t}(i, j) = P(q_t = i, q_{t+1} = j | O, \lambda)$$

We estimate this using the forward and backward probabilities (for before state $i$ and after state $j$, respectively):

$$\hat{\xi}_{t}(i, j) = \frac{\alpha_{t}(i) a_{ij} b_{j}(q_{t+1}) \beta_{t+1}(j)}{\alpha_{t}(q_k)}$$

Estimating Emission Probabilities

To estimate the $B$ values, intuitively we want:

$$\hat{b}_{j}(v_k) = \frac{\text{expected number of times in state } j \text{ and observing symbol } v_k}{\text{expected number of times in state } j}$$

We define the probability of being in state $j$ at time $t$:

$$\gamma_{t}(j) = P(q_t = j | O, \lambda)$$

We estimate this using the forward and backward probabilities (for before state $i$ and after state $j$, respectively):

$$\gamma_{t}(j) = \frac{\alpha_{t}(j) \beta_{t}(j)}{P(O | \lambda)}$$

Formulas for Estimations

Using the previous definitions, we can estimate the transition and emission probabilities as follows:

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

$\rightarrow$ expected # transitions from $q_i$ to $q_j$

$$\hat{b}_{j}(v_k) = \frac{\sum_{t=1}^{T} \mathbb{1}_{t,s,q_t=v_k} \gamma_{t}(j)}{\sum_{t=1}^{T} \gamma_{t}(j)}$$

$\rightarrow$ expected times in $q_j$ with $v_k$

$\rightarrow$ expected times in $q_i$
The 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: estimate probabilities based on current parameters
\[ \gamma(t, j) = \frac{\alpha_t(i) \beta_t(j)}{\alpha_t(i)} \quad \forall t \text{ and } j \]
\[ \zeta_t(i, j) = \frac{\alpha_t(i) a_{ij} \beta_{t+1}(j)}{\alpha_t(i) \beta_t(j)} \quad \forall t, i, \text{ and } j \]

M-step: re-estimate parameter values
\[ \hat{a}_{ij} = \frac{\sum_{t=1}^{T-1} \zeta_t(i, j)}{\sum_{t=1}^{T-1} \sum_{k=1}^{N} \zeta_t(i, k)} \]
\[ \hat{b}_{j}(\omega) = \frac{\sum_{t=1}^{T} \delta_{t}(\omega) \gamma(j)}{\sum_{j=1}^{J} \sum_{t=1}^{T} \gamma(j)} \]

return A, B
```

Initialization and Convergence

- The forward-backward algorithm can be very sensitive to the initial parameter settings.
- The algorithm works best if the initial probabilities are set based on prior knowledge, rather than randomly. For example, the HMM structure may be hand-crafted, illegal transitions should have zero probability, etc.
- A small amount of labeled data can be used to obtain reasonable initial parameter settings.
- The algorithm is run over the data until the values converge. It is guaranteed to converge to a local maximum, but not a global maximum.

Summary

- HMMs are widely used across many applications because of their generality and effectiveness.
- The Viterbi algorithm makes decoding with an HMM quite efficient.
- The key benefit of HMMs is that they compute the probability of a sequence as a whole, capturing local dependencies.
- The transition and emission probabilities can be estimated from labeled and unlabeled data.
- A limitation of HMMs, however, is that they represent a specific generative model and cannot use arbitrary features.