Outline

Algorithms for Hidden Markov Models
  Main HMM Algorithms
  HMM as Parser
  Viterbi Algorithm for HMMs
  HMM as Language Model
Hidden Markov Model

Model $\theta = \left\{ \begin{array}{l} \pi_i \quad \text{probability of starting at state } i \\ a_{i,j} \quad \text{probability of transition from state } i \text{ to state } j \\ b_i(o) \quad \text{probability of output } o \text{ at state } i \end{array} \right.$

- HMM as parser: compute the best sequence of states for a given observation sequence.
- HMM as language model: compute probability of given observation sequence.
- HMM as learner: given a corpus of observation sequences, learn its distribution, i.e. learn the parameters of the HMM from the corpus.
  - Learning from a set of observations with the sequence of states provided (states are not hidden) [Supervised Learning]
  - Learning from a set of observations without any state information. [Unsupervised Learning]
HMM as Parser

\[ \pi = \begin{pmatrix} A & N \\ 0.25 & 0.75 \end{pmatrix} \]

\[ a = \begin{pmatrix} a_{i,j} & A & N \\ N & 0.5 & 0.5 \\ A & 0.0 & 1.0 \end{pmatrix} \]

\[ b = \begin{pmatrix} b_i(o) & A & N \\ clown & 0.0 & 0.4 \\ killer & 0.0 & 0.3 \\ problem & 0.0 & 0.3 \\ crazy & 1.0 & 0.0 \end{pmatrix} \]

The task: for a given observation sequence find the most likely state sequence.
HMM as Parser

- Find most likely sequence of states for *killer clown*
- Score every possible sequence of states: AA, AN, NN, NA
  - \( P(\text{killer clown, AA}) = \pi_A \cdot b_A(\text{killer}) \cdot a_{A,A} \cdot b_A(\text{clown}) = 0.0 \)
  - \( P(\text{killer clown, AN}) = \pi_A \cdot b_A(\text{killer}) \cdot a_{A,N} \cdot b_N(\text{clown}) = 0.0 \)
  - \( P(\text{killer clown, NN}) = \pi_N \cdot b_N(\text{killer}) \cdot a_{N,N} \cdot b_N(\text{clown}) = 0.75 \cdot 0.3 \cdot 0.5 \cdot 0.4 = 0.045 \)
  - \( P(\text{killer clown, NA}) = \pi_N \cdot b_N(\text{killer}) \cdot a_{N,A} \cdot b_A(\text{clown}) = 0.0 \)
- Pick the state sequence with highest probability (NN=0.045).

HMM as Parser

- As we have seen, for input of length 2, and a HMM with 2 states there are \(2^2\) possible state sequences.
- In general, if we have \(q\) states and input of length \(T\) there are \(q^T\) possible state sequences.
- Using our example HMM, for input *killer crazy clown problem* we will have \(2^4\) possible state sequences to score.
- Our naive algorithm takes exponential time to find the best state sequence for a given input.
- The **Viterbi algorithm** uses dynamic programming to provide the best state sequence with a time complexity of \(q^2 \cdot T\).
Viterbi Algorithm for HMMs

- For input of length $T$: $o_1, \ldots, o_T$, we want to find the sequence of states $s_1, \ldots, s_T$
- Each $s_t$ in this sequence is one of the states in the HMM.
- So the task is to find the most likely sequence of states:
  \[
  \arg\max_{s_1, \ldots, s_T} P(o_1, \ldots, o_T, s_1, \ldots, s_T)
  \]
- The Viterbi algorithm solves this by creating a table $V[s, t]$ where $s$ is one of the states, and $t$ is an index between $1, \ldots, T$. 
Viterbi Algorithm for HMMs

Consider the input *killer crazy clown problem*

So the task is to find the most likely sequence of states:

\[ \arg\max_{s_1, s_2, s_3, s_4} P(killer\ crazy\ clown\ problem, s_1, s_2, s_3, s_4) \]

A sub-problem is to find the most likely sequence of states for *killer crazy clown*:

\[ \arg\max_{s_1, s_2, s_3} P(killer\ crazy\ clown, s_1, s_2, s_3) \]

In our example there are two possible values for \( s_4 \):

\[
\max_{s_1, \ldots, s_4} P(killer\ crazy\ clown\ problem, s_1, s_2, s_3, s_4) = \max \left\{ \begin{array}{l}
\max_{s_1, s_2, s_3} P(killer\ crazy\ clown\ problem, s_1, s_2, s_3, N), \\
\max_{s_1, s_2, s_3} P(killer\ crazy\ clown\ problem, s_1, s_2, s_3, A)
\end{array} \right\}
\]

Similarly:

\[
\arg\max_{s_1, \ldots, s_3} P(killer\ crazy\ clown, s_1, s_2, s_3) = \arg\max_{N, V} \left\{ \max_{s_1, s_2} P(killer\ crazy\ clown, s_1, s_2, N), \right. \\
\left. \max_{s_1, s_2} P(killer\ crazy\ clown, s_1, s_2, A) \right\}
\]
Viterbi Algorithm for HMMs

- Putting them together:
  
  \[ P(\text{killer crazy clown problem}, s_1, s_2, s_3, N) = \]
  
  \[ \max \{ P(\text{killer crazy clown}, s_1, s_2, N) \cdot a_{N,N} \cdot b_N(\text{problem}), \]
  
  \[ P(\text{killer crazy clown}, s_1, s_2, A) \cdot a_{A,N} \cdot b_N(\text{problem}) \} \]

  \[ P(\text{killer crazy clown problem}, s_1, s_2, s_3, A) = \]
  
  \[ \max \{ P(\text{killer crazy clown}, s_1, s_2, N) \cdot a_{N,A} \cdot b_A(\text{problem}), \]
  
  \[ P(\text{killer crazy clown}, s_1, s_2, A) \cdot a_{A,A} \cdot b_A(\text{problem}) \} \]

- The best score is given by:
  
  \[ \max_{s_1, \ldots, s_4} P(\text{killer crazy clown problem}, s_1, s_2, s_3, s_4) = \]
  
  \[ \max_{N,A} \left\{ \max_{s_1, s_2, s_3} P(\text{killer crazy clown problem}, s_1, s_2, s_3, N), \right. \]
  
  \[ \left. \max_{s_1, s_2, s_3} P(\text{killer crazy clown problem}, s_1, s_2, s_3, A) \right\} \]

Viterbi Algorithm for HMMs

- Provide an index for each input symbol:
  
  1:killer 2:crazy 3:clown 4:problem

  \[ V[N, 3] = \max_{s_1, s_2} P(\text{killer crazy clown}, s_1, s_2, N) \]
  
  \[ V[N, 4] = \max_{s_1, s_2, s_3} P(\text{killer crazy clown problem}, s_1, s_2, s_3, N) \]

- Putting them together:
  
  \[ V[N, 4] = \max \{ V[N, 3] \cdot a_{N,N} \cdot b_N(\text{problem}), \]
  
  \[ V[A, 3] \cdot a_{A,N} \cdot b_N(\text{problem}) \} \]

  \[ V[A, 4] = \max \{ V[N, 3] \cdot a_{N,A} \cdot b_A(\text{problem}), \]
  
  \[ V[A, 3] \cdot a_{A,A} \cdot b_A(\text{problem}) \} \]

- The best score for the input is given by:
  
  \[ \max \{ V[N, 4], V[A, 4] \} \]

- To extract the best sequence of states we backtrack (same trick as obtaining alignments from minimum edit distance)
Viterbi Algorithm for HMMs

For input of length $T$: $o_1, \ldots, o_T$, we want to find the sequence of states $s_1, \ldots, s_T$

- Each $s_t$ in this sequence is one of the states in the HMM.
- For each state $q$ we initialize our table: $V[q, 1] = \pi_q \cdot b_q(o_1)$
- Then compute recursively for $t = 1 \ldots T - 1$ for each state $q$:
  \[
  V[q, t + 1] = \max_{q'} \{ V[q', t] \cdot a_{q', q} \cdot b_q(o_{t+1}) \}
  \]

- After the loop terminates, the best score is $\max_q V[q, T]$

Outline

Algorithms for Hidden Markov Models
  Main HMM Algorithms
  HMM as Parser
  Viterbi Algorithm for HMMs
  HMM as Language Model
HMM as Language Model

Consider the input *killer crazy clown problem*

So the task is to find the sum over all sequences of states:

\[
\sum_{s_1,s_2,s_3,s_4} P(\text{killer crazy clown problem}, s_1, s_2, s_3, s_4)
\]

A sub-problem is to find the most likely sequence of states for *killer crazy clown*:

\[
\sum_{s_1,s_2,s_3} P(\text{killer crazy clown}, s_1, s_2, s_3)
\]
HMM as Language Model

- In our example there are two possible values for \( s_4 \):

\[
\sum_{s_1, \ldots, s_4} P(\text{killer crazy clown problem}, s_1, s_2, s_3, s_4) = \\
\sum_{s_1, s_2, s_3} P(\text{killer crazy clown problem}, s_1, s_2, s_3, N) + \\
\sum_{s_1, s_2, s_3} P(\text{killer crazy clown problem}, s_1, s_2, s_3, A)
\]

- Very similar to the Viterbi algorithm. Sum instead of max, and that’s the only difference!

HMM as Language Model

- Provide an index for each input symbol:

  1:killer 2:crazy 3:clown 4:problem

\[
V[N, 3] = \sum_{s_1, s_2} P(\text{killer crazy clown}, s_1, s_2, N)
\]

\[
V[N, 4] = \sum_{s_1, s_2, s_3} P(\text{killer crazy clown problem}, s_1, s_2, s_3, N)
\]

- Putting them together:

\[
V[N, 4] = V[N, 3] \cdot a_{N,N} \cdot b_N(\text{problem}) + \\
V[A, 3] \cdot a_{A,N} \cdot b_N(\text{problem})
\]

\[
V[A, 4] = V[N, 3] \cdot a_{N,A} \cdot b_A(\text{problem}) + \\
V[A, 3] \cdot a_{A,A} \cdot b_A(\text{problem})
\]

- The best score for the input is given by: \( V[N, 4] + V[A, 4] \)
HMM as Language Model

- For input of length $T$: $o_1, \ldots, o_T$, we want to find
  \[ P(o_1, \ldots, o_T) = \sum_{y_1, \ldots, y_T} P(y_1, \ldots, y_T, o_1, \ldots, o_T) \]
- Each $y_t$ in this sequence is one of the states in the HMM.
- For each state $q$ we initialize our table: $V[q, 1] = \pi_q \cdot b_q(o_1)$
- Then compute recursively for $t = 1 \ldots T - 1$ for each state $q$:
  \[ V[q, t + 1] = \sum_{q'} \{ V[q', t] \cdot a_{q', q} \cdot b_q(o_{t+1}) \} \]
- After the loop terminates, the best score is $\sum_q V[q, T]$
- So: Viterbi with sum instead of max gives us an algorithm for HMM as a language model.
- This algorithm is sometimes called the forward algorithm.