Hidden Markov Model and Viterbi Algorithm

Instructor: Wei Xu

Many slides adapted from Michael Collins
Overview

- The Tagging Problem
- Generative models, and the noisy-channel model, for supervised learning
- Hidden Markov Model (HMM) taggers
  - Basic definitions
  - Parameter estimation
  - The Viterbi algorithm
Hidden Markov Models

- We have an input sentence \( x = x_1, x_2, \ldots, x_n \)
  \((x_i \text{ is the } i\text{'th word in the sentence})\)

- We have a tag sequence \( y = y_1, y_2, \ldots, y_n \)
  \((y_i \text{ is the } i\text{'th tag in the sentence})\)

- We’ll use an HMM to define
  \[
p(x_1, x_2, \ldots, x_n, y_1, y_2, \ldots, y_n)
  \]
  for any sentence \( x_1 \ldots x_n \) and tag sequence \( y_1 \ldots y_n \) of the same length.

- Then the most likely tag sequence for \( x \) is
  \[
  \arg \max_{y_1 \ldots y_n} p(x_1 \ldots x_n, y_1, y_2, \ldots, y_n)
  \]
Trigram Hidden Markov Models (Trigram HMMs)

For any sentence \( x_1 \ldots x_n \) where \( x_i \in \mathcal{V} \) for \( i = 1 \ldots n \), and any tag sequence \( y_1 \ldots y_{n+1} \) where \( y_i \in \mathcal{S} \) for \( i = 1 \ldots n \), and \( y_{n+1} = \text{STOP} \), the joint probability of the sentence and tag sequence is

\[
p(x_1 \ldots x_n, y_1 \ldots y_{n+1}) = \prod_{i=1}^{n+1} q(y_i | y_{i-2}, y_{i-1}) \prod_{i=1}^{n} e(x_i | y_i)
\]

where we have assumed that \( x_0 = x_{-1} = * \).

Parameters of the model:

- \( q(s | u, v) \) for any \( s \in \mathcal{S} \cup \{ \text{STOP} \} \), \( u, v \in \mathcal{S} \cup \{ * \} \)
- \( e(x | s) \) for any \( s \in \mathcal{S} \), \( x \in \mathcal{V} \)
An Example

If we have $n = 3$, $x_1 \ldots x_3$ equal to the sentence the dog laughs, and $y_1 \ldots y_4$ equal to the tag sequence D N V STOP, then

$$p(x_1 \ldots x_n, y_1 \ldots y_{n+1}) = q(D|*,*) \times q(N|*,D) \times q(V|D,N) \times q(STOP|N,V) \times e(the|D) \times e(dog|N) \times e(laughs|V)$$

- STOP is a special tag that terminates the sequence
- We take $y_0 = y_{-1} = *$, where * is a special "padding" symbol
Why the Name?

\[
p(x_1 \ldots x_n, y_1 \ldots y_n) = q(\text{STOP}|y_{n-1}, y_n) \prod_{j=1}^{n} q(y_j | y_{j-2}, y_{j-1})
\]

\[\text{Markov Chain}\]

\[\times \prod_{j=1}^{n} e(x_j | y_j)\]

\[x_j's\ are\ observed\]
Overview

- The Tagging Problem

- Generative models, and the noisy-channel model, for supervised learning

- Hidden Markov Model (HMM) taggers
  - Basic definitions
  - Parameter estimation
  - The Viterbi algorithm
Smoothed Estimation

\[ q(Vt \mid DT, JJ) = \lambda_1 \times \frac{\text{Count}(Dt, JJ, Vt)}{\text{Count}(Dt, JJ)} + \lambda_2 \times \frac{\text{Count}(JJ, Vt)}{\text{Count}(JJ)} + \lambda_3 \times \frac{\text{Count}(Vt)}{\text{Count}()} \]

\[ \lambda_1 + \lambda_2 + \lambda_3 = 1, \quad \text{and for all } i, \lambda_i \geq 0 \]

\[ e(\text{base} \mid Vt) = \frac{\text{Count}(Vt, \text{base})}{\text{Count}(Vt)} \]
Profits soared at Boeing Co., easily topping forecasts on Wall Street, as their CEO Alan Mulally announced first quarter results.
Dealing with Low-Frequency Words

A common method is as follows:

- **Step 1:** Split vocabulary into two sets
  
  - **Frequent words** = words occurring $\geq 5$ times in training
  - **Low frequency words** = all other words

- **Step 2:** Map low frequency words into a small, finite set, depending on prefixes, suffixes etc.
Dealing with Low-Frequency Words: An Example

[Bikel et. al 1999] (named-entity recognition)

<table>
<thead>
<tr>
<th>Word class</th>
<th>Example</th>
<th>Intuition</th>
</tr>
</thead>
<tbody>
<tr>
<td>twoDigitNum</td>
<td>90</td>
<td>Two digit year</td>
</tr>
<tr>
<td>fourDigitNum</td>
<td>1990</td>
<td>Four digit year</td>
</tr>
<tr>
<td>containsDigitAndAlpha</td>
<td>A8956-67</td>
<td>Product code</td>
</tr>
<tr>
<td>containsDigitAndDash</td>
<td>09-96</td>
<td>Date</td>
</tr>
<tr>
<td>containsDigitAndSlash</td>
<td>11/9/89</td>
<td>Date</td>
</tr>
<tr>
<td>containsDigitAndComma</td>
<td>23,000.00</td>
<td>Monetary amount</td>
</tr>
<tr>
<td>containsDigitAndPeriod</td>
<td>1.00</td>
<td>Monetary amount, percentage</td>
</tr>
<tr>
<td>othernum</td>
<td>456789</td>
<td>Other number</td>
</tr>
<tr>
<td>allCaps</td>
<td>BBN</td>
<td>Organization</td>
</tr>
<tr>
<td>capPeriod</td>
<td>M.</td>
<td>Person name initial</td>
</tr>
<tr>
<td>firstWord</td>
<td>first word of sentence</td>
<td>no useful capitalization information</td>
</tr>
<tr>
<td>initCap</td>
<td>Sally</td>
<td>Capitalized word</td>
</tr>
<tr>
<td>lowercase</td>
<td>can</td>
<td>Uncapitalized word</td>
</tr>
<tr>
<td>other</td>
<td>,</td>
<td>Punctuation marks, all other words</td>
</tr>
</tbody>
</table>
Dealing with Low-Frequency Words: An Example

Profits/NA soared/NA at/NA Boeing/SC Co./CC ,/NA easily/NA
topping/NA forecasts/NA on/NA Wall/SL Street/CL ,/NA as/NA their/NA
CEO/NA Alan/SP Mulally/CP announced/NA first/NA quarter/NA
results/NA ./NA

↓

firstword/NA soared/NA at/NA initCap/SC Co./CC ,/NA easily/NA
lowercase/NA forecasts/NA on/NA initCap/SL Street/CL ,/NA as/NA
their/NA CEO/NA Alan/SP initCap/CP announced/NA first/NA
quarter/NA results/NA ./NA

NA = No entity
SC = Start Company
CC = Continue Company
SL = Start Location
CL = Continue Location
Overview

- The Tagging Problem
- Generative models, and the noisy-channel model, for supervised learning
- Hidden Markov Model (HMM) taggers
  - Basic definitions
  - Parameter estimation
  - The Viterbi algorithm
The Viterbi Algorithm

Problem: for an input \(x_1 \ldots x_n\), find

\[
\operatorname{arg\ max}_{y_1 \ldots y_{n+1}} p(x_1 \ldots x_n, y_1 \ldots y_{n+1})
\]

where the \(\operatorname{arg\ max}\) is taken over all sequences \(y_1 \ldots y_{n+1}\) such that \(y_i \in S\) for \(i = 1 \ldots n\), and \(y_{n+1} = \text{STOP}\).

We assume that \(p\) again takes the form

\[
p(x_1 \ldots x_n, y_1 \ldots y_{n+1}) = \prod_{i=1}^{n+1} q(y_i|y_{i-2}, y_{i-1}) \prod_{i=1}^{n} e(x_i|y_i)
\]

Recall that we have assumed in this definition that \(y_0 = y_{-1} = \ast\), and \(y_{n+1} = \text{STOP}\).
Brute Force Search is Hopelessly Inefficient

Problem: for an input $x_1 \ldots x_n$, find

$$\arg \max_{y_1 \ldots y_{n+1}} p(x_1 \ldots x_n, y_1 \ldots y_{n+1})$$

where the $\arg \max$ is taken over all sequences $y_1 \ldots y_{n+1}$ such that $y_i \in S$ for $i = 1 \ldots n$, and $y_{n+1} = \text{STOP}$. 
The Viterbi Algorithm

- Define $n$ to be the length of the sentence
- Define $S_k$ for $k = -1 \ldots n$ to be the set of possible tags at position $k$:
  
  $S_{-1} = S_0 = \{*, \text{end}\}$
  
  $S_k = S$ for $k \in \{1 \ldots n\}$

- Define

  $$r(y_{-1}, y_0, y_1, \ldots, y_k) = \prod_{i=1}^{k} q(y_i | y_{i-2}, y_{i-1}) \prod_{i=1}^{k} e(x_i | y_i)$$

- Define a dynamic programming table

  $$\pi(k, u, v) = \text{maximum probability of a tag sequence ending in tags } u, v \text{ at position } k$$

  that is,

  $$\pi(k, u, v) = \max_{(y_{-1}, y_0, y_1, \ldots, y_k): y_{k-1} = u, y_k = v} r(y_{-1}, y_0, y_1 \ldots y_k)$$
An Example

\[ \pi(k, u, v) = \text{maximum probability of a tag sequence ending in tags } u, v \text{ at position } k \]

The man saw the dog with the telescope
A Recursive Definition

Base case:

$$\pi(0, *, *) = 1$$

Recursive definition:
For any $k \in \{1 \ldots n\}$, for any $u \in S_{k-1}$ and $v \in S_k$:

$$\pi(k, u, v) = \max_{w \in S_{k-2}} (\pi(k-1, w, u) \times q(v|w, u) \times e(x_k|v))$$
Justification for the Recursive Definition

For any $k \in \{1 \ldots n\}$, for any $u \in S_{k-1}$ and $v \in S_k$:

$$\pi(k, u, v) = \max_{w \in S_{k-2}} (\pi(k - 1, w, u) \times q(v|w, u) \times e(x_k|v))$$

The man saw the dog with the telescope
The Viterbi Algorithm

Input: a sentence \( x_1 \ldots x_n \), parameters \( q(s|u,v) \) and \( e(x|s) \).

Initialization: Set \( \pi(0, *, *) = 1 \)

Definition: \( S_{-1} = S_0 = \{\ast\} \), \( S_k = S \) for \( k \in \{1 \ldots n\} \)

Algorithm:

- For \( k = 1 \ldots n \),
  - For \( u \in S_{k-1}, v \in S_k \),
    \[ \pi(k, u, v) = \max_{w \in S_{k-2}} (\pi(k - 1, w, u) \times q(v|w, u) \times e(x_k|v)) \]
  - Return \( \max_{u \in S_{n-1}, v \in S_n} (\pi(n, u, v) \times q(\text{STOP}|u, v)) \)
The Viterbi Algorithm with Backpointers

Input: a sentence $x_1 \ldots x_n$, parameters $q(s|u, v)$ and $e(x|s)$.

Initialization: Set $\pi(0, *, *) = 1$

Definition: $S_{-1} = S_0 = \{*\}$, $S_k = S$ for $k \in \{1 \ldots n\}$

Algorithm:

- For $k = 1 \ldots n$,
  - For $u \in S_{k-1}$, $v \in S_k$,
    $$\pi(k, u, v) = \max_{w \in S_{k-2}} (\pi(k-1, w, u) \times q(v|w, u) \times e(x_k|v))$$
    $$bp(k, u, v) = \arg \max_{w \in S_{k-2}} (\pi(k-1, w, u) \times q(v|w, u) \times e(x_k|v))$$
  - Set $(y_{n-1}, y_n) = \arg \max_{(u,v)} (\pi(n, u, v) \times q(\text{STOP}|u, v))$
  - For $k = (n-2) \ldots 1$, $y_k = bp(k + 2, y_{k+1}, y_{k+2})$
- Return the tag sequence $y_1 \ldots y_n$
The Viterbi Algorithm: Running Time

- $O(n|S|^3)$ time to calculate $q(s|u,v) \times e(x_k|s)$ for all $k, s, u, v$.
- $n|S|^2$ entries in $\pi$ to be filled in.
- $O(|S|)$ time to fill in one entry
- $\Rightarrow O(n|S|^3)$ time in total
Pros and Cons

- Hidden Markov model taggers are very simple to train (just need to compile counts from the training corpus)
- Perform relatively well (over 90% performance on named entity recognition)
- Main difficulty is modeling

\[ e(\text{word} \mid \text{tag}) \]

can be very difficult if “words” are complex