6.864 (Fall 2006): Lecture 7
Tagging and History-Based Models

Overview

• The Tagging Problem

• Hidden Markov Model (HMM) taggers

• Log-linear taggers

• Log-linear models for parsing and other problems

Tagging Problems

• Mapping strings to Tagged Sequences

\[ a \ b \ e \ e \ a \ f \ h \ j \Rightarrow a/C \ b/D \ e/C \ e/C \ a/D \ f/C \ h/D \ j/C \]

Part-of-Speech Tagging

INPUT:
Profits soared at Boeing Co., easily topping forecasts on Wall Street, as their CEO Alan Mulally announced first quarter results.

OUTPUT:
Profits/N soared/V at/P Boeing/N Co./N ,/. easily/ADV topping/V forecasts/N on/P Wall/N Street/N ,/. as/P their/POSS CEO/N Alan/N Mulally/N announced/V first/ADJ quarter/N results/N ./.
Information Extraction

Named Entity Recognition

**INPUT:** Profits soared at Boeing Co., easily topping forecasts on Wall Street, as their CEO Alan Mulally announced first quarter results.

**OUTPUT:** Profits soared at Boeing Co., easily topping forecasts on Wall Street, as their CEO Alan Mulally announced first quarter results.

Extracting Glossary Entries from the Web

**Input:**

**Output:** St. Elmo's Fire: A luminous, and often audible, electric discharge that is sporadic in nature. It occurs from objects, especially pointed ones, when the electrical field strength near their surfaces attains a value near 100 volts per centimeter...

Our Goal

**Training set:**
1 Pierre/NNP Vinken/NNP , 61/CD years/NNS old/JJ , will/MD join/VB the/DT board/NN as/IN a/DT nonexecutive/JJ director/NN Nov./NNP 29/CD .
2 Mr./NNP Vinken/NNP is/VBZ chairman/NN of/IN Elsevier/NNP N.V./NNP , the/DT Dutch/NNP publishing/VBG group/NNP .
3 Rudolph/NNP Agnew/NNP , 55/CD years/NNS old/JJ and/CC chairman/NN of/IN Consolidated/NNP Gold/NPP Fields/NNP PLC/NNP , was/VBD named/VBN a/DT nonexecutive/JJ director/NN of/IN this/DT British/JJ industrial/JJ conglomerate/NN .

- From the training set, induce a function/algorithm that maps new sentences to their tag sequences.
Our Goal (continued)

- A test data sentence:
  Influential members of the House Ways and Means Committee introduced legislation that would restrict how the new savings-and-loan bailout agency can raise capital, creating another potential obstacle to the government’s sale of sick thrifts.

- Should be mapped to underlying tags:
  Influential/JJ members/NNS of/IN the/DT House/NNP Ways/NNP and/CC Means/NNP Committee/NNP introduced/VBD legislation/NN that/WDT would/MD restrict/VB how/WRB the/DT new/JJ savings-and-loan/NN bailout/NN agency/NN can/MD raise/VB capital/NN, creating/VBG another/DT potential/JJ obstacle/NN to/TO the/DT government/NN’s/POS sale/NN of/IN sick/JJ thrifts/NNS.

- Our goal is to minimize the number of tagging errors on sentences not seen in the training set

Two Types of Constraints

- “Local”: e.g., can is more likely to be a modal verb MD rather than a noun NN
- “Contextual”: e.g., a noun is much more likely than a verb to follow a determiner
- Sometimes these preferences are in conflict:
  The trash can is in the garage

A Naive Approach

- Use a machine learning method to build a “classifier” that maps each word individually to its tag
- A problem: does not take contextual constraints into account

Hidden Markov Models

- We have an input sentence $S = w_1, w_2, \ldots, w_n$ ($w_i$ is the $i$’th word in the sentence)
- We have a tag sequence $T = t_1, t_2, \ldots, t_n$ ($t_i$ is the $i$’th tag in the sentence)
- We’ll use an HMM to define
  \[ P(t_1, t_2, \ldots, t_n, w_1, w_2, \ldots, w_n) \]
  for any sentence $S$ and tag sequence $T$ of the same length.
- Then the most likely tag sequence for $S$ is
  \[ T^* = \arg\max_T P(T, S) \]
How to model $P(T, S)$?

A Trigram HMM Tagger:

$$P(T, S) = P(\text{END} | t_1 \ldots t_n, w_1 \ldots w_n) \times \prod_{j=1}^{n} \left[ \frac{P(t_j | w_1 \ldots w_{j-1}, t_1 \ldots t_{j-1}) \times P(w_j | w_1 \ldots w_{j-1}, t_1 \ldots t_j)}{P(t_j | t_{j-2}, t_{j-1}) \times P(w_j | t_j)} \right]$$

Chain rule

$$= P(\text{END} | t_{n-1}, t_n) \times \prod_{j=1}^{n} \left[ P(t_j | t_{j-2}, t_{j-1}) \times P(w_j | t_j) \right]$$

Independence assumptions

- END is a special tag that terminates the sequence
- We take $t_0 = t_{-1} = \text{START}$
- 1st assumption: each tag only depends on previous two tags $P(t_j | t_{j-1}, t_{j-2})$
- 2nd assumption: each word only depends on underlying tag $P(w_j | t_j)$

Why the Name?

$$P(T, S) = P(\text{END} | t_{n-1}, t_n) \prod_{j=1}^{n} P(t_j | t_{j-2}, t_{j-1}) \times \prod_{j=1}^{n} P(w_j | t_j)$$

Hidden Markov Chain

$w_j$'s are observed

An Example

- $S =$ the boy laughed
- $T =$ DT NN VBD

$$P(T, S) = P(\text{END} | \text{NN, VBD}) \times P(\text{DT} | \text{START, START}) \times P(\text{NN} | \text{START, DT}) \times P(\text{VBD} | \text{DT, NN}) \times P(\text{the} | \text{DT}) \times P(\text{boy} | \text{NN}) \times P(\text{laughed} | \text{VBD})$$

How to model $P(T, S)$?

Hispaniola/NNP quickly/RB became/VB an/DT important/JJ base/Vt from which Spain expanded its empire into the rest of the Western Hemisphere.

“Score” for tag Vt:

$$P(\text{Vt} | \text{DT, JJ}) \times P(\text{base} | \text{Vt})$$
**Smoothed Estimation**

\[
P(V_t \mid DT, JJ) = \lambda_1 \times \frac{\text{Count}(DT, JJ, V_t)}{\text{Count}(DT, JJ)} + \lambda_2 \times \frac{\text{Count}(JJ, V_t)}{\text{Count}(JJ)} + \lambda_3 \times \frac{\text{Count}(V_t)}{\text{Count}()} + \lambda_4 \times \frac{\text{Count}(\text{base}, V_t)}{\text{Count}(\text{base})}
\]

\[
P(\text{base} \mid V_t) = \frac{\text{Count}(V_t, \text{base})}{\text{Count}(V_t)}
\]

---

**Dealing with Low-Frequency Words**

- **Step 1**: Split vocabulary into two sets
  - **Frequent words** = words occurring ≥ 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] An Algorithm that Learns What’s in a Name

<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>othernum</td>
<td>1.00</td>
<td>Monetary amount, percentage</td>
</tr>
<tr>
<td>allCaps</td>
<td>BBN</td>
<td>Other number</td>
</tr>
<tr>
<td>capPeriod</td>
<td>M.</td>
<td>Organization</td>
</tr>
<tr>
<td>fi rstWord</td>
<td>fi rst 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**

Profi ts/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 fi rst/NA quarter/NA results/NA /NA

↓

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

| NA      | = No entity |
| SC      | = Start Company |
| CC      | = Continue Company |
| SL      | = Start Location |
| CL      | = Continue Location |
| ...     |              |
The Viterbi Algorithm

- Question: how do we calculate the following?
  \[ T^* = \arg\max_T \log P(T, S) \]

- Define \( n \) to be the length of the sentence

- Define a dynamic programming table
  \[ \pi[i, t-2, t-1] = \text{maximum log probability of a tag sequence ending in tags } t-2, t-1 \text{ at position } i \]

- Our goal is to calculate \( \max_{t-2, t-1} \pi[n, t-2, t-1] \)

The Viterbi Algorithm: Recursive Definitions

- Base case:
  \[ \pi[0, *, *] = \log 1 = 0 \]
  \[ \pi[0, t-2, t-1] = \log 0 = -\infty \text{ for all other } t-2, t-1 \]

  here \( * \) is a special tag padding the beginning of the sentence.

- Recursive case: for \( i = 1 \ldots n \), for all \( t-2, t-1 \),
  \[ \pi[i, t-2, t-1] = \max_{t \in T \cup \{*\}} \{ \pi[i, t-1, t-2] + \text{Score}(S, i, t, t-2, t-1) \} \]

Backpointers allow us to recover the max probability sequence:
\[ \text{BP}[i, t-2, t-1] = \arg\max_{t \in T \cup \{*\}} \{ \pi[i, t-1, t-2] + \text{Score}(S, i, t, t-2, t-1) \} \]

Where \( \text{Score}(S, i, t, t-2, t-1) = \log P(t-1 \mid t, t-2) + \log P(w_i \mid t-1) \)

Complexity is \( O(nk^3) \), where \( n = \text{length of sentence}, k \) is number of possible tags

The Viterbi Algorithm: Running Time

- \( O(n|T|^3) \) time to calculate \( \text{Score}(S, i, t, t-2, t-1) \) for all \( i, t, t-2, t-1 \).

- \( n|T|^2 \) entries in \( \pi \) to be filled in.

- \( O(T) \) time to fill in one entry (assuming \( O(1) \) time to look up \( \text{Score}(S, i, t, t-2, t-1) \))

\( \Rightarrow O(n|T|^3) \) time

Pros and Cons

- Hidden markov model taggers are very simple to train (compile counts from the training corpus)

- Perform relatively well (over 90% performance on named entities)

- Main difficulty is modeling \( P(\text{word} \mid \text{tag}) \) can be very difficult if “words” are complex
Log-Linear Models

- We have an input sentence \( S = w_1, w_2, \ldots, w_n \) (\( w_i \) is the \( i \)'th word in the sentence)
- We have a tag sequence \( T = t_1, t_2, \ldots, t_n \) (\( t_i \) is the \( i \)'th tag in the sentence)
- We’ll use an log-linear model to define
  \[
P(t_1, t_2, \ldots, t_n | w_1, w_2, \ldots, w_n)
  \]
  for any sentence \( S \) and tag sequence \( T \) of the same length.
  (Note: contrast with HMM that defines
  \[
P(t_1, t_2, \ldots, t_n, w_1, w_2, \ldots, w_n)
  \]
- Then the most likely tag sequence for \( S \) is
  \[
  T^* = \arg\max_T P(T|S)
  \]

How to model \( P(T|S) \)?

A Trigram Log-Linear Tagger:

\[
P(T|S) = \prod_{j=1}^{n} P(t_j | w_1 \ldots w_n, t_1 \ldots t_{j-1}) \quad \text{Chain rule}
\]

\[
= \prod_{j=1}^{n} P(t_j | t_{j-2}, t_{j-1}, w_1, \ldots, w_n) \quad \text{Independence assumptions}
\]

- We take \( t_0 = t_{-1} = \text{START} \)
- Assumption: each tag only depends on previous two tags
  \[
P(t_j | t_{j-1}, t_{j-2}, w_1, \ldots, w_n)
  \]

An Example

Hispaniola/NNP quickly/RB became/VB an/DT important/JJ base/?? from which Spain expanded its empire into the rest of the Western Hemisphere.

- There are many possible tags in the position ??
  \( Y = \{\text{NN, NNS, Vt, Vi, IN, DT, \ldots}\} \)
- The input domain \( \mathcal{X} \) is the set of all possible histories (or contexts)
- Need to learn a function from (history, tag) pairs to a probability \( P(\text{tag}|\text{history}) \)

Representation: Histories

- A history is a 4-tuple \( \langle t_1, t_2, w_{[1:n]}, i \rangle \)
- \( t_{-1}, t_{-2} \) are the previous two tags.
- \( w_{[1:n]} \) are the \( n \) words in the input sentence.
- \( i \) is the index of the word being tagged
- \( \mathcal{X} \) is the set of all possible histories

Hispaniola/NNP quickly/RB became/VB an/DT important/JJ base/?? from which Spain expanded its empire into the rest of the Western Hemisphere.

- \( t_{-1}, t_{-2} = \text{DT, JJ} \)
- \( w_{[1:n]} = \langle \text{Hispaniola, quickly, became, \ldots, Hemisphere, \ldots} \rangle \)
- \( i = 6 \)
Feature Vector Representations

- We have some input domain $\mathcal{X}$, and a finite label set $\mathcal{Y}$. Aim is to provide a conditional probability $P(y \mid x)$ for any $x \in \mathcal{X}$ and $y \in \mathcal{Y}$.

- A feature is a function $f : \mathcal{X} \times \mathcal{Y} \to \mathbb{R}$ (Often binary features or indicator functions $f : \mathcal{X} \times \mathcal{Y} \to \{0, 1\}$).

- Say we have $m$ features $\phi_k$ for $k = 1 \ldots m$
  \[ \Rightarrow \text{A feature vector } \phi(x, y) \in \mathbb{R}^m \text{ for any } x \in \mathcal{X} \text{ and } y \in \mathcal{Y}. \]

An Example (continued)

- $\mathcal{X}$ is the set of all possible histories of form $\{t_{-1}, t_{-2}, w_{[1:n]}, i\}$

- $\mathcal{Y} = \{\text{NN, NNS, Vt, Vi, IN, DT, ...}\}$

- We have $m$ features $f_k : \mathcal{X} \times \mathcal{Y} \to \mathbb{R}$ for $k = 1 \ldots m$

For example:

\[
\begin{align*}
\phi_1(h, t) &= \begin{cases} 
    1 & \text{if current word } w_i \text{ is base and } t = \text{Vt} \\
    0 & \text{otherwise}
\end{cases} \\
\phi_2(h, t) &= \begin{cases} 
    1 & \text{if current word } w_i \text{ ends in } \text{Ing and } t = \text{VBG} \\
    0 & \text{otherwise}
\end{cases} \\
&\vdots \\
\phi_1(\langle \text{JJ, DT, Hispaniola, ...}, 6 \rangle, \text{Vt}) &= 1 \\
\phi_2(\langle \text{JJ, DT, Hispaniola, ...}, 6 \rangle, \text{Vt}) &= 0 \\
&\vdots
\end{align*}
\]

The Full Set of Features in [(Ratnaparkhi, 96)]

- Word/tag features for all word/tag pairs, e.g.,

\[
\phi_{100}(h, t) = \begin{cases} 
    1 & \text{if current word } w_i \text{ is base and } t = \text{Vt} \\
    0 & \text{otherwise}
\end{cases}
\]

- Spelling features for all prefixes/suffixes of length $\leq 4$, e.g.,

\[
\begin{align*}
\phi_{101}(h, t) &= \begin{cases} 
    1 & \text{if current word } w_i \text{ ends in } \text{ing and } t = \text{VBG} \\
    0 & \text{otherwise}
\end{cases} \\
\phi_{102}(h, t) &= \begin{cases} 
    1 & \text{if current word } w_i \text{ starts with } \text{pre and } t = \text{NN} \\
    0 & \text{otherwise}
\end{cases}
\end{align*}
\]

- Contextual Features, e.g.,

\[
\begin{align*}
\phi_{103}(h, t) &= \begin{cases} 
    1 & \text{if } \langle t_{-2}, t_{-1}, t \rangle = \langle \text{DT, JJ, Vt} \rangle \\
    0 & \text{otherwise}
\end{cases} \\
\phi_{104}(h, t) &= \begin{cases} 
    1 & \text{if } \langle t_{-1}, t \rangle = \langle \text{JJ, Vt} \rangle \\
    0 & \text{otherwise}
\end{cases} \\
\phi_{105}(h, t) &= \begin{cases} 
    1 & \text{if } \langle t \rangle = \langle \text{Vt} \rangle \\
    0 & \text{otherwise}
\end{cases} \\
\phi_{106}(h, t) &= \begin{cases} 
    1 & \text{if previous word } w_{i-1} = \text{the and } t = \text{Vt} \\
    0 & \text{otherwise}
\end{cases} \\
\phi_{107}(h, t) &= \begin{cases} 
    1 & \text{if next word } w_{i+1} = \text{the and } t = \text{Vt} \\
    0 & \text{otherwise}
\end{cases}
\end{align*}
\]
The Final Result

- We can come up with practically any questions (features) regarding history/tag pairs.

- For a given history $x \in \mathcal{X}$, each label in $\mathcal{Y}$ is mapped to a different feature vector

\[
\begin{align*}
\phi((\text{JJ, DT, (Hispaniola, ...), 6}), \text{Vt}) &= 1001011001001100110 \\
\phi((\text{JJ, DT, (Hispaniola, ...), 6}), \text{JJ}) &= 01100101011110010 \\
\phi((\text{JJ, DT, (Hispaniola, ...), 6}), \text{NN}) &= 000111101001100100 \\
\phi((\text{JJ, DT, (Hispaniola, ...), 6}), \text{IN}) &= 000101101100000010 \\
\cdots
\end{align*}
\]

Training the Log-Linear Model

- To train a log-linear model, we need a training set $(x_i, y_i)$ for $i = 1 \ldots n$. Then search for

\[
W^* = \arg\max_W \left( \sum_i \log P(y_i|x_i, W) - C \sum_k W_k^2 \right)
\]

(see last lecture on log-linear models)

- Training set is simply all history/tag pairs seen in the training data

Log-Linear Models

- We have some input domain $\mathcal{X}$, and a finite label set $\mathcal{Y}$. Aim is to provide a conditional probability $P(y | x)$ for any $x \in \mathcal{X}$ and $y \in \mathcal{Y}$.

- A feature is a function $f : \mathcal{X} \times \mathcal{Y} \rightarrow \mathbb{R}$ (Often binary features or indicator functions $f : \mathcal{X} \times \mathcal{Y} \rightarrow \{0, 1\}$).

- Say we have $m$ features $\phi_k$ for $k = 1 \ldots m$  
$\Rightarrow$ A feature vector $\phi(x, y) \in \mathbb{R}^m$ for any $x \in \mathcal{X}$ and $y \in \mathcal{Y}$.

- We also have a parameter vector $W \in \mathbb{R}^m$

- We define

\[
P(y | x, W) = \frac{e^{W \cdot \phi(x, y)}}{\sum_{y' \in \mathcal{Y}} e^{W \cdot \phi(x, y')}}
\]

The Viterbi Algorithm for Log-Linear Models

- Question: how do we calculate the following?:

\[
T^* = \arg\max_T \log P(T|S)
\]

- Define $n$ to be the length of the sentence

- Define a dynamic programming table

\[
\pi[i, t_{-2}, t_{-1}] = \text{maximum log probability of a tag sequence ending in tags } t_{-2}, t_{-1} \text{ at position } i
\]

- Our goal is to calculate $\max_{t_{-2}, t_{-1} \in T} \pi[n, t_{-2}, t_{-1}]$
The Viterbi Algorithm: Recursive Definitions

- **Base case:**
  \[ \pi[0, *, *] = \log 1 = 0 \]
  \[ \pi[0, t-2, t-1] = \log 0 = -\infty \text{ for all other } t-2, t-1 \]

  Here * is a special tag padding the beginning of the sentence.

- **Recursive case:** for \( i = 1 \ldots n \), for all \( t-2, t-1 \),
  \[ \pi[i, t-2, t-1] = \max_{t \in \mathcal{T} \cup \{*\}} \{ \pi[i-1, t, t-2] + \text{Score}(S, i, t, t-2, t-1) \} \]

  Backpointers allow us to recover the max probability sequence:
  \[ \text{BP}[i, t-2, t-1] = \arg\max_{t \in \mathcal{T} \cup \{*\}} \{ \pi[i-1, t, t-2] + \text{Score}(S, i, t, t-2, t-1) \} \]

  Where \( \text{Score}(S, i, t, t-2, t-1) = \log P(t-1 \mid t, t-2, w_1, \ldots, w_n, i) \)

  Identical to Viterbi for HMMs, except for the definition of \( \text{Score}(S, i, t, t-2, t-1) \)

FAQ Segmentation: McCallum et. al

- McCallum et. al compared HMM and log-linear taggers on a *FAQ Segmentation* task

- Main point: in an HMM, modeling
  \[ P(\text{word} | \text{tag}) \]

  is difficult in this domain

FAQ Segmentation: Line Features

- begins-with-number
- begins-with-ordinal
- begins-with-punctuation
- begins-with-question-word
- begins-with-subject
- blank
- contains-alphanumeric
- contains-bracketed-number
- contains-http
- contains-non-space
- contains-number
- contains-pipe
- contains-question-mark
- ends-with-question-mark
- first-alpha-is-capitalized
- indented-1-to-4
- indented-5-to-10
- more-than-one-third-space
- only-punctuation
- prev-is-blank
- prev-begins-with-ordinal
- shorter-than-30
FAQ Segmentation: The Log-Linear Tagger

Here follows a diagram of the necessary connections programs to work properly. They are as far as I know agreed upon by commercial comms software developers for

Pins 1, 4, and 8 must be connected together inside is to avoid the well known serial port chip bugs. The

⇒ “tag=question;prev=head;begins-with-number”
“tag=question;prev=head;contains-alphanum”
“tag=question;prev=head;contains-nonspace”
“tag=question;prev=head;contains-number”
“tag=question;prev=head;prev-is-blank”

FAQ Segmentation: An HMM Tagger

First solution for $P(word \mid tag)$:

$P(\text{"2.6) What configuration of serial cable should I use"} | \text{question}) =$
\begin{align*}
\times & P(\text{What} \mid \text{question}) \\
\times & P(\text{configuration} \mid \text{question}) \\
\times & P(\text{of} \mid \text{question}) \\
\times & P(\text{serial} \mid \text{question}) \\
\end{align*}

⇒ i.e. have a language model for each tag

FAQ Segmentation: McCallum et. al

Second solution: first map each sentence to string of features:

⇒

Use a language model again:

$P(\text{"2.6) What configuration of serial cable should I use"} | \text{question}) =$
\begin{align*}
\times & P(\text{begins-with-number} \mid \text{question}) \\
\times & P(\text{contains-alphanum} \mid \text{question}) \\
\times & P(\text{contains-nonspace} \mid \text{question}) \\
\times & P(\text{contains-number} \mid \text{question}) \\
\times & P(\text{prev-is-blank} \mid \text{question}) \\
\end{align*}

FAQ Segmentation: Results

<table>
<thead>
<tr>
<th>Method</th>
<th>COAP</th>
<th>SegPrec</th>
<th>SegRec</th>
</tr>
</thead>
<tbody>
<tr>
<td>ME-Stateless</td>
<td>0.520</td>
<td>0.038</td>
<td>0.362</td>
</tr>
<tr>
<td>TokenHMM</td>
<td>0.865</td>
<td>0.276</td>
<td>0.140</td>
</tr>
<tr>
<td>FeatureHMM</td>
<td>0.941</td>
<td>0.413</td>
<td>0.529</td>
</tr>
<tr>
<td>MEMM</td>
<td>0.965</td>
<td>0.867</td>
<td>0.681</td>
</tr>
</tbody>
</table>
Overview

- The Tagging Problem
- Hidden Markov Model (HMM) taggers
- Log-linear taggers
- Log-linear models for parsing and other problems

Log-Linear Taggers: Summary

- The input sentence is $S = w_1 \ldots w_n$
- Each tag sequence $T$ has a conditional probability
  \[ P(T \mid S) = \prod_{j=1}^{n} P(t_j \mid w_1 \ldots w_n, j, t_1 \ldots t_{j-1}) \]
  \[ = \prod_{j=1}^{n} P(t_j \mid w_1 \ldots w_n, j, t_{j-2}, t_{j-1}) \]
  Chain rule
  Independence assumptions
- Estimate $P(t_j \mid w_1 \ldots w_n, j, t_{j-2}, t_{j-1})$ using log-linear models
- Use the Viterbi algorithm to compute
  \[ \arg \max_{T \in \mathcal{T}} \log P(T \mid S) \]

A General Approach: (Conditional) History-Based Models

- We’ve shown how to define $P(T \mid S)$ where $T$ is a tag sequence
- How do we define $P(T \mid S)$ if $T$ is a parse tree (or another structure)?

Step 1: represent a tree as a sequence of decisions $d_1 \ldots d_m$
\[ T = \langle d_1, d_2, \ldots d_m \rangle \]
$m$ is not necessarily the length of the sentence

Step 2: the probability of a tree is
\[ P(T \mid S) = \prod_{i=1}^{m} P(d_i \mid d_1 \ldots d_{i-1}, S) \]

Step 3: Use a log-linear model to estimate
\[ P(d_i \mid d_1 \ldots d_{i-1}, S) \]

Step 4: Search?? (answer we’ll get to later: beam or heuristic search)
**An Example Tree**

```
S(queried)
  NP(lawyer)
    DT the
    NN lawyer
  VP(queried)
    Vt questioned
  PP(about)
    IN about
    NP(witness)
      DT the
      NN witness
  PP(about)
    IN about
    NP(revolver)
      DT the
      NN revolver
```

**Layer 1: Part-of-Speech Tags**

```
DT  NN  Vt  DT  NN  IN  DT  NN
the  lawyer  questioned  the  witness  about  the  revolver
```

- Step 1: represent a tree as a sequence of decisions $d_1 \ldots d_m$

$$T = \langle d_1, d_2, \ldots d_m \rangle$$

- First $n$ decisions are tagging decisions

$$\langle d_1 \ldots d_n \rangle = \langle DT, NN, Vt, DT, NN, IN, DT, NN \rangle$$

**Ratnaparkhi’s Parser: Three Layers of Structure**

1. Part-of-speech tags
2. Chunks
3. Remaining structure

**Layer 2: Chunks**

```
NP
  Vt questioned
  NP
    IN
      NP
        DT the
        NN revolver
```

Chunks are defined as any phrase where all children are part-of-speech tags

(Other common chunks are ADJP, QP)
Layer 2: Chunks

- Step 1: represent a tree as a sequence of decisions $d_1 \ldots d_n$
  \[ T = \langle d_1, d_2, \ldots d_n \rangle \]

- First $n$ decisions are tagging decisions
  Next $n$ decisions are chunk tagging decisions
  \[ \langle d_1 \ldots d_{2n} \rangle = \langle \text{DT, NN, Vt, DT, NN, IN, DT, NN, Start(NP), Join(NP), Other, Start(NP), Join(NP), Other, Start(NP), Join(NP)} \rangle \]

Layer 3: Remaining Structure

Alternate Between Two Classes of Actions:

- Join(X) or Start(X), where X is a label (NP, S, VP etc.)

- Check=YES or Check=NO

Meaning of these actions:

- Start(X) starts a new constituent with label X
  (always acts on leftmost constituent with no start or join label above it)
- Join(X) continues a constituent with label X
  (always acts on leftmost constituent with no start or join label above it)
- Check=NO does nothing
- Check=YES takes previous Join or Start action, and converts it into a completed constituent
The lawyer questioned the witness about the revolver.
The lawyer questioned the witness about the revolver.
Start(S)  Start(VP)  Join(VP)  PP
NP        Vt        NP        IN  NP
| DT   | NN   | questioned | DT   | NN
the   lawyer                the   witness

Check=YES

Start(S)  VP
NP        Vt  NP        PP
| DT   | NN   | questioned  | DT   | NN  | IN  NP
| the   | witness | about | the | revolver

Check=YES

Start(S)  VP
NP        Vt  NP        PP
| DT   | NN   | questioned  | DT   | NN  | IN  NP
| the   | witness | about | the | revolver

Start(S)  Join(S)
VP        NP        PP  NP
| Vt       | DT   | NN   | IN | NP
| questioned | the | witness | about | the | revolver

Check=YES
A General Approach: (Conditional) History-Based Models

- Step 1: represent a tree as a sequence of decisions \( d_1 \ldots d_m \)
  \[ T = \langle d_1, d_2, \ldots d_m \rangle \]
  \( m \) is not necessarily the length of the sentence

- Step 2: the probability of a tree is
  \[ P(T \mid S) = \prod_{i=1}^{m} P(d_i \mid d_1 \ldots d_{i-1}, S) \]

- Step 3: Use a log-linear model to estimate
  \[ P(d_i \mid d_1 \ldots d_{i-1}, S) \]

- Step 4: Search?? (answer we’ll get to later: beam or heuristic search)

The Final Sequence of decisions

\[ \langle d_1 \ldots d_m \rangle = \langle \text{DT, NN, Vt, DT, NN, IN, DT, NN,} \]
\[ \text{Start(NP), Join(NP), Other, Start(NP), Join(NP),} \]
\[ \text{Other, Start(NP), Join(NP),} \]
\[ \text{Start(S), Check=NO, Start(VP), Check=NO,} \]
\[ \text{Join(VP), Check=NO, Start(PP), Check=NO,} \]
\[ \text{Join(PP), Check=YES, Join(VP), Check=YES,} \]
\[ \text{Join(S), Check=YES} \rangle \]

Applying a Log-Linear Model

- Step 3: Use a log-linear model to estimate
  \[ P(d_i \mid d_1 \ldots d_{i-1}, S) \]

- A reminder:
  \[ P(d_i \mid d_1 \ldots d_{i-1}, S) = \frac{e^{\phi(d_1 \ldots d_{i-1}, S), d_i} \cdot W}{\sum_{d \in \mathcal{A}} e^{\phi(d_1 \ldots d_{i-1}, S), d} \cdot W} \]

where:
\[ \langle d_1 \ldots d_{i-1}, S \rangle \] is the history
\[ d_i \] is the outcome
\[ \phi \] maps a history/outcome pair to a feature vector
\[ W \] is a parameter vector
\[ \mathcal{A} \] is set of possible actions
(may be context dependent)
Applying a Log-Linear Model

- Step 3: Use a log-linear model to estimate
  \[ P(d_i \mid d_1 \ldots d_{i-1}, S) = \frac{e^{\phi((d_1 \ldots d_{i-1}, S), d_i)} w}{\sum_{d \in \mathcal{A}} e^{\phi((d_1 \ldots d_{i-1}, S), d)} w} \]

- The big question: how do we define \( \phi \)?
  - Ratnaparkhi's method defines \( \phi \) differently depending on whether next decision is:
    - A tagging decision
      (same features as before for POS tagging!)
    - A chunking decision
    - A start/join decision after chunking
    - A check=no/check=yes decision

Layer 2: Chunks

<table>
<thead>
<tr>
<th>Start(NP)</th>
<th>Join(NP)</th>
<th>Other</th>
<th>Start(NP)</th>
<th>Join(NP)</th>
<th>IN</th>
<th>DT</th>
<th>NN</th>
</tr>
</thead>
<tbody>
<tr>
<td>DT</td>
<td>NN</td>
<td>Vt</td>
<td>DT</td>
<td>NN</td>
<td>about</td>
<td>the</td>
<td>revolver</td>
</tr>
<tr>
<td>the</td>
<td>lawyer</td>
<td>questioned</td>
<td>the</td>
<td>witness</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

⇒ “TAG=Join(NP);Word0=witness;POS0=NN”
“TAG=Join(NP);POS0=NN”
“TAG=Join(NP);Word+1=about;POS+1=IN”
“TAG=Join(NP);POS+1=IN”
“TAG=Join(NP);Word+2=the;POS+2=DT”
“TAG=Join(NP);POS+2=IN”
“TAG=Join(NP);Word-1=the;POS-1=DT;TAG-1=Start(NP)”
“TAG=Join(NP);POS-1=DT;TAG-1=Start(NP)”
“TAG=Join(NP);TAG-1=Start(NP)”
...

Layer 3: Join or Start

- Looks at head word, constituent (or POS) label, and start/join annotation of \( n \)'th tree relative to the decision, where \( n = -2, -1 \)
- Looks at head word, constituent (or POS) label of \( n \)'th tree relative to the decision, where \( n = 0, 1, 2 \)
- Looks at bigram features of the above for (-1,0) and (0,1)
- Looks at trigram features of the above for (-2,-1,0), (-1,0,1) and (0, 1, 2)
- The above features with all combinations of head words excluded
- Various punctuation features

Layer 3: Check=NO or Check=YES

- A variety of questions concerning the proposed constituent
The Search Problem

- In POS tagging, we could use the Viterbi algorithm because
  \[ P(t_j \mid w_1 \ldots w_n, j, t_1 \ldots t_{j-1}) = P(t_j \mid w_1 \ldots w_n, j, t_{j-2} \ldots t_{j-1}) \]

- Now: Decision \( d_i \) could depend on arbitrary decisions in the “past” \( \Rightarrow \) no chance for dynamic programming

- Instead, Ratnaparkhi uses a beam search method