Overview

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

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 ./.

N = Noun
V = Verb
P = Preposition
Adv = Adverb
Adj = Adjective
...
Named Entity Extraction as Tagging

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 easily topping forecasts on Wall Street, as their CEO Alan Mulally announced first quarter results.

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

Our Goal

Training set:
1 Pierre Vinken, 61 years old, will join the board as a nonexecutive director.
2 Mr. Vinken is chairman of Elsevier N.V., the Dutch publishing group.
3 Rudolph Agnew, 55 years old and chairman of Consolidated Gold Fields PLC, was named a nonexecutive director.

ANaive 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.

From the training set, induce a function/algorithm that maps new sentences to their tag sequences.

Two Types of Constraints

Influential members introduced legislation that would restrict how the new savings-and-loan bailout agency can raise capital.

- “Local”: e.g., can is more likely to be a modal verb rather than a noun
- “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
Overview

- The Tagging Problem
- Hidden Markov Model (HMM) taggers
  - Basic definitions
  - Parameter estimation
  - The Viterbi Algorithm
- Log-linear taggers
- Log-linear models for parsing and other problems

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:

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

Chain rule

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

Independence assumptions

- END is a special tag that terminates the sequence
- We take $t_0 = t_{-1} = *$, where * is a special “padding” symbol

Independence Assumptions in the Trigram HMM Tagger

- 1st independence assumption: each tag only depends on previous two tags
  $$P(t_j \mid w_1 \ldots w_{j-1}, t_1 \ldots t_{j-1}) = P(t_j \mid t_{j-2}, t_{j-1})$$
- 2nd independence assumption: each word only depends on underlying tag
  $$P(w_j \mid w_1 \ldots w_{j-1}, t_1 \ldots t_j) = P(w_j \mid t_j)$$
**An Example**

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

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

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

Hispaniola/NNP quickly/RB became/VB an/DT important/JJ base/Vt

Probability of generating base/Vt:

$$P(Vt \mid DT, JJ) \times P(\text{base} \mid Vt)$$

**Why the Name?**

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

**Overview**

- The Tagging Problem
- Hidden Markov Model (HMM) taggers
  - Basic definitions
  - Parameter estimation
  - The Viterbi Algorithm
- Log-linear taggers
- Log-linear models for parsing and other problems
Smoothed Estimation

\[ P(V_t | D_t, J) = \lambda_1 \times \frac{\text{Count}(D_t, J, V_t)}{\text{Count}(D_t, J)} + \lambda_2 \times \frac{\text{Count}(J, V_t)}{\text{Count}(J)} + \lambda_3 \times \frac{\text{Count}(V_t)}{\text{Count}()} \]

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

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

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, percentage</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>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

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

- Our goal is to calculate
  \[ \max_{u, v \in T} \pi[n, u, v] \]

The Viterbi Algorithm: Recursive Definitions

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

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

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

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

Where \( \text{Score}(S, i, t, u, v) = \log P(v \mid t, u) + \log P(w_i \mid v) \)

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

The Viterbi Algorithm: Running Time

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

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

- \( O(|T|) \) time to fill in one entry

- \( \Rightarrow O(n|T|^3) \) time
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 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
  \[ P(t_1, t_2, \ldots, t_n \mid w_1, w_2, \ldots, w_n) \]
  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
  \[ T^* = \arg \max_T P(T \mid S) \]

Overview

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

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

A Trigram Log-Linear Tagger:

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

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

- We take \( t_0 = t_{-1} = * \)
- Independence assumption: each tag only depends on previous two tags
  \[
P(t_j \mid w_1, \ldots, w_n, t_1, \ldots, t_{j-1}) = P(t_j \mid w_1, \ldots, w_n, t_{j-2}, t_{j-1})
\]
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 ??
  \( \mathcal{Y} = \{ \text{NN, NNS, Vt, Vi, IN, DT, \ldots} \} \)
- 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 \( P(\text{tag}|\text{history}) \)

**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 \( f_k \) for \( k = 1 \ldots m \)
  \( \Rightarrow \) A feature vector \( f(x, y) \in \mathbb{R}^m \) for any \( x \in \mathcal{X} \) and \( y \in \mathcal{Y} \).

**Representation: Histories**

- A history is a 4-tuple \( \langle t_{-2}, t_{-1}, w_{[1:n]}, i \rangle \)
- \( t_{-2}, t_{-1} \) 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.

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

An Example (continued)

- \( \mathcal{X} \) is the set of all possible histories of form \( \langle t_{-2}, t_{-1}, w_{[1:n]}, i \rangle \)
- \( \mathcal{Y} = \{ \text{NN, NNS, Vt, Vi, IN, DT, \ldots} \} \)
- We have \( m \) features \( f_k : \mathcal{X} \times \mathcal{Y} \to \mathbb{R} \) for \( k = 1 \ldots m \)

For example:

\[
\begin{align*}
f_1(h, t) &= \begin{cases} 
1 & \text{if current word } w_i \text{ is base and } t = \text{Vt} \\
0 & \text{otherwise}
\end{cases} \\
f_2(h, t) &= \begin{cases} 
1 & \text{if current word } w_i \text{ ends in } \text{ing and } t = \text{VBG} \\
0 & \text{otherwise}
\end{cases}
\end{align*}
\]

\[
\begin{align*}
f_1(\langle \text{JJ, DT, \langle \text{Hispaniola, \ldots} \rangle, 6 \rangle, \text{Vt} \rangle) &= 1 \\
f_2(\langle \text{JJ, DT, \langle \text{Hispaniola, \ldots} \rangle, 6 \rangle, \text{Vt} \rangle) &= 0 \\
\ldots
\end{align*}
\]
The Full Set of Features in [(Ratnaparkhi, 96)]

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

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

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

\[ f_{101}(h, t) = \begin{cases} 1 & \text{if current word } w_i \text{ ends in } \text{ing and } t = \text{VBG} \\ 0 & \text{otherwise} \end{cases} \]

\[ f_{102}(h, t) = \begin{cases} 1 & \text{if current word } w_i \text{ starts with } \text{pre} \text{ and } t = \text{NN} \\ 0 & \text{otherwise} \end{cases} \]

Log-Linear Models

- We have some input domain \( \mathcal{X} \), and a finite label set \( \mathcal{Y} \). Aim is to provide a conditional probability \( P(y \mid x) \) 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 \( f_k \) for \( k = 1 \ldots m \)

  \[ \Rightarrow \text{A feature vector } f(x, y) \in \mathbb{R}^m \text{ for any } x \in \mathcal{X} \text{ and } y \in \mathcal{Y}. \]

- We also have a parameter vector \( v \in \mathbb{R}^m \)

  We define

\[
P(y \mid x, v) = \frac{e^{v \cdot f(x, y)}}{\sum_{y' \in \mathcal{Y}} e^{v \cdot f(x, y')}}
\]

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

- Contextual Features, e.g.,

\[ f_{103}(h, t) = \begin{cases} 1 & \text{if } \langle t_{-2}, t_{-1}, t \rangle = \langle \text{DT, JJ, Vt} \rangle \\ 0 & \text{otherwise} \end{cases} \]

\[ f_{104}(h, t) = \begin{cases} 1 & \text{if } \langle t_{-1}, t \rangle = \langle \text{JJ, Vt} \rangle \\ 0 & \text{otherwise} \end{cases} \]

\[ f_{105}(h, t) = \begin{cases} 1 & \text{if } \langle t \rangle = \langle \text{Vt} \rangle \\ 0 & \text{otherwise} \end{cases} \]

\[ f_{106}(h, t) = \begin{cases} 1 & \text{if previous word } w_{i-1} = \text{the and } t = \text{Vt} \\ 0 & \text{otherwise} \end{cases} \]

\[ f_{107}(h, t) = \begin{cases} 1 & \text{if next word } w_{i+1} = \text{the and } t = \text{Vt} \\ 0 & \text{otherwise} \end{cases} \]

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

\[
v^* = \arg\max_v \left( \sum_i \log P(y_i \mid x_i, v) - \frac{1}{2\sigma^2} \sum_k v_k^2 \right)
\]

(see last lecture on log-linear models)

- Training set is simply all history/tag pairs seen in the training data
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
  \[ \pi[i, u, v] = \text{maximum log probability of a tag sequence ending in tags } u, v \text{ at position } i \]

- Our goal is to calculate \( \max_{u, v \in T} \pi[n, u, v] \)

The Viterbi Algorithm: Recursive Definitions

- **Base case:**
  \[ \pi[0, *, *,] = \log 1 = 0 \]
  \[ \pi[0, u, v] = \log 0 = -\infty \text{ for all other } u, v \]
  here * is a special tag padding the beginning of the sentence.

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

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

Where \( \text{Score}(S, i, t, u, v) = \log P(v | t, u, w_1, \ldots, w_n, i) \)

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

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: McCallum et. al

- 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 to avoid the well known serial port chip bugs. The
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 $P(word \mid tag)$:

\[
P(2.6) \text{ What configuration of serial cable should I use}\mid \text{ question}) = \\
P(2.6 \mid \text{ question}) \times \\
P(\text{What} \mid \text{ question}) \times \\
P(\text{configuration} \mid \text{ question}) \times \\
P(\text{of} \mid \text{ question}) \times \\
P(\text{serial} \mid \text{ question}) \times \\... \\
\]

- i.e. have a language model for each $tag$

FAQ Segmentation: McCallum et. al

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

⇒

⇒

FAQ Segmentation: McCallum et. al

- Use a language model again:

\[
P(2.6) \text{ What configuration of serial cable should I use}\mid \text{ question}) = \\
P(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(prev-is-blank \mid \text{ question}) \times \\... \\]
### FAQ Segmentation: Results

<table>
<thead>
<tr>
<th>Method</th>
<th>Precision</th>
<th>Recall</th>
</tr>
</thead>
<tbody>
<tr>
<td>ME-Stateless</td>
<td>0.038</td>
<td>0.362</td>
</tr>
<tr>
<td>TokenHMM</td>
<td>0.276</td>
<td>0.140</td>
</tr>
<tr>
<td>FeatureHMM</td>
<td>0.413</td>
<td>0.529</td>
</tr>
<tr>
<td>MEMM</td>
<td>0.867</td>
<td>0.681</td>
</tr>
</tbody>
</table>

- Precision and recall results are for recovering segments
- ME-stateless is a log-linear model that treats every sentence separately (no dependence between adjacent tags)
- TokenHMM is an HMM with first solution we’ve just seen
- FeatureHMM is an HMM with second solution we’ve just seen
- MEMM is a log-linear trigram tagger (MEMM stands for ‘Maximum-Entropy Markov Model’)

### Log-Linear Taggers: Summary

- The input sentence is $S = w_1 \ldots w_n$
- Each tag sequence $T$ has a conditional probability
  
  $$P(T \mid S) = \prod_{j=1}^{n} P(t_j \mid w_1 \ldots w_n, t_1 \ldots t_{j-1})$$  
  
  - **Chain rule**
  
  $$= \prod_{j=1}^{n} P(t_j \mid w_1 \ldots w_n, t_{j-2}, t_{j-1})$$  
  
  - **Independence assumptions**

- Estimate $P(t_j \mid w_1 \ldots w_n, t_{j-2}, t_{j-1})$ using log-linear models
- Use the Viterbi algorithm to compute
  
  $$\arg\max_{T \in \mathcal{T}} \log P(T \mid S)$$

### Overview

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

### 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)?
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)

An Example Tree

Ratnaparkhi’s Parser: Three Layers of Structure

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

Layer 1: Part-of-Speech Tags

- 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

\[
\langle d_1 \ldots d_n \rangle = \langle \text{DT}, \text{NN}, \text{Vt}, \text{DT}, \text{NN}, \text{IN}, \text{DT}, \text{NN} \rangle
\]
Layer 2: Chunks

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

(Other common chunks are ADJP, QP)

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

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

$\langle d_1 \ldots d_{2n} \rangle = \langle \ DT, \ NN, \ Vt, \ DT, \ NN, \ IN, \ DT, \ NN, \ Start(NP), \ Join(NP), \ Other, \ Start(NP), \ Join(NP), \ Other, \ Start(NP), \ Join(NP) \rangle$
The lawyer questioned the witness about the revolver.

Check=NO
the lawyer questioned the witness about the revolver.

Check=YES
The Final Sequence of decisions

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

A General Approach: (Conditional) History-Based Models

- Step 1: represent a tree as a sequence of \textbf{decisions} \( d_1 \ldots d_m \)
  \[
  T = \langle d_1, d_2, \ldots d_m \rangle
  \]
  \( m \) is \textbf{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)
Applying a Log-Linear Model

- Step 3: Use a log-linear model to estimate

\[ P(d_i | d_1 \ldots d_{i-1}, S) \]

- A reminder:

\[ P(d_i | d_1 \ldots d_{i-1}, S) = \frac{e^{f((d_1 \ldots d_{i-1}, S), d_i) \cdot v}}{\sum_{d \in A} e^{f((d_1 \ldots d_{i-1}, S), d) \cdot v}} \]

where:
- \( <d_1 \ldots d_{i-1}, S> \) is the history
- \( d_i \) is the outcome
- \( f \) maps a history/outcome pair to a feature vector
- \( v \) is a parameter vector
- \( A \) is set of possible actions

The big question: how do we define \( f \)?

Ratnaparkhi’s method defines \( f \) 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 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

\[ P(t_j | w_1 \ldots w_n, j, t_1 \ldots t_{j-1}) = P(t_j | w_1 \ldots w_n, j, t_{j-2} \ldots t_{j-1}) \]

- 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