6.864 (Fall 2007): Lecture 7
Tagging

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 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 [Company Boeing Co.], easily topping forecasts on [Location Wall Street], as their CEO [Person Alan Mulally] announced first quarter results.

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/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

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

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/NN ./.
3 Rudolph/NNP Agnew/NNP ,/, 55/CD years/NNS old/JJ and/CC chairman/NN of/IN Consolidated/NNP Gold/NNP Fields/NNP PLC/NNP ,/, was/VBD named/VBN a/DT nonexecutive/JJ director/NN of/IN this/DT British/JJ industrial/JJ conglomerate/NN ./.
. . .
38,219 It/PRP is/VBZ also/RB pulling/VBG 20/CD people/NNS out/IN of/IN Puerto/NNP Rico/NNP ,/, who/WP were/VBD helping/VBG Huricane/NNP Hugo/NNP victims/NNS ,/, and/CC sending/VBG them/PRP to/TO San/NNP Francisco/NNP instead/RB ./.

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

Two Types of Constraints

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 ./.

- “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

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

$$T^* = \text{argmax}_T P(T, S)$$

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

A Trigram HMM Tagger:

$$\begin{aligned} P(T,S) = & P(\text{END} \mid w_1 \ldots w_n, t_1 \ldots t_n) \times \\ & \prod_{j=1}^{n} [\ P(t_j \mid w_1 \ldots w_{j-1}, t_1 \ldots t_{j-1}) \times \\ & \quad P(w_j \mid w_1 \ldots w_{j-1}, t_1 \ldots t_j)] \end{aligned}$$ Chain rule

$$\begin{aligned} = & P(\text{END} | t_{n-1}, t_n) \times \\ & \prod_{j=1}^{n} [P(t_j \mid t_{j-2}, t_{j-1}) \times P(w_j \mid t_j)] \end{aligned}$$ 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

$$P(t_j | w_1 \ldots w_{j-1}, t_1 \ldots t_{j-1}) = P(t_j | t_{j-2}, t_{j-1})$$

- 2nd independence assumption: each word only depends on underlying tag

$$P(w_j | w_1 \ldots w_{j-1}, t_1 \ldots t_j) = P(w_j | t_j)$$

An Example

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

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

Why the Name?

$$
P(T, S) = \underbrace{P(\text{END}|t_{n-1}, t_n) \prod_{j=1}^{n} P(t_j \mid t_{j-2}, t_{j-1})}_{\text{Hidden Markov Chain}} \times \underbrace{\prod_{j=1}^{n} P(w_j \mid t_j)}_{w_j\text{'s are observed}}
$$

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

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

Probability of generating base/Vt:

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

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

$$
\begin{aligned}
P(\text{Vt} \mid \text{DT, JJ}) &= \lambda_1 \times \frac{Count(\text{Dt, JJ, Vt})}{Count(\text{Dt, JJ})} \\
&+ \lambda_2 \times \frac{Count(\text{JJ, Vt})}{Count(\text{JJ})} \\
&+ \lambda_3 \times \frac{Count(\text{Vt})}{Count()}
\end{aligned}
$$

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

$$P(\text{base} \mid \text{Vt}) = \frac{Count(\text{Vt, base})}{Count(\text{Vt})}$$

Dealing with Low-Frequency Words

A common method is as follows:

- **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] **(named-entity recognition)**

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

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

$\Downarrow$

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

The Viterbi Algorithm

- Question: how do we calculate the following?:

$$T^* = \text{argmax}_T \log P(T, S)$$

- Define n to be the length of the sentence
- Define a dynamic programming table

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

- Our goal is to calculate

$$\max_{u,v \in \mathcal{T}} \pi[n, u, v]$$

The Viterbi Algorithm: Recursive Definitions

- **Base case:**

$$\begin{aligned} \pi[0, *, *] &= \log 1 = 0 \\ \pi[0, u, v] &= \log 0 = -\infty \text{ for all other } u, v \end{aligned}$$

here $*$ is a special tag padding the beginning of the sentence.

- **Recursive case:** for $i = 1 \ldots n$, for all u, v,

$$\pi[i, u, v] = \max_{t \in \mathcal{T} \cup \{*\}} \{\pi[i-1, t, u] + Score(S, i, t, u, v)\}$$

Backpointers allow us to recover the max probability sequence:

$$\text{BP}[i, u, v] = \text{argmax}_{t \in \mathcal{T} \cup \{*\}} \{\pi[i-1, t, u] + Score(S, i, t, u, v)\}$$

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

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

The Viterbi Algorithm: Running Time

- $O(n|\mathcal{T}|^3)$ time to calculate $Score(S, i, t, u, v)$ for all i, t, u, v.
- $n|\mathcal{T}|^2$ entries in π to be filled in.
- $O(\mathcal{T})$ time to fill in one entry
- $\Rightarrow O(n|\mathcal{T}|^3)$ 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

$$P(word \mid tag)$$

can be very difficult if "words" are complex

Overview

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

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

$$T^* = \text{argmax}_T P(T|S)$$

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

A Trigram Log-Linear Tagger:

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

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

- We take $t_0 = t_{-1} = *$
- Independence assumption: each tag only depends on previous two tags

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

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 ??
$\mathcal{Y} = \{$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 $P(tag|history)$

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 .

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

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 $f : \mathcal{X} \times \mathcal{Y} \rightarrow \mathbb{R}$
(Often **binary features** or **indicator functions** $f : \mathcal{X} \times \mathcal{Y} \rightarrow \{0, 1\}$).

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

An Example (continued)

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

For example:

$$f_1(h, t) = \begin{cases} 1 & \text{if current word } w_i \text{ is } \texttt{base} \text{ and } t = \texttt{Vt} \\ 0 & \text{otherwise} \end{cases}$$

$$f_2(h, t) = \begin{cases} 1 & \text{if current word } w_i \text{ ends in } \texttt{ing} \text{ and } t = \texttt{VBG} \\ 0 & \text{otherwise} \end{cases}$$

$\ldots$

$f_1(\langle$JJ, DT, $\langle$ Hispaniola, $\ldots\rangle$, 6$\rangle$, Vt$) = 1$
$f_2(\langle$JJ, DT, $\langle$ Hispaniola, $\ldots\rangle$, 6$\rangle$, Vt$) = 0$
$\ldots$

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

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

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

- Spelling features for all prefixes/suffixes of length ≤ 4, e.g.,

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

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

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

- Contextual Features, e.g.,

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

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

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

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

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

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 $k = 1 \ldots m$
$\Rightarrow$ A feature vector $\mathbf{f}(x, y) \in \mathbb{R}^m$ for any $x \in \mathcal{X}$ and $y \in \mathcal{Y}$.
- We also have a **parameter vector** $\mathbf{v} \in \mathbb{R}^m$
- We define

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

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

$$\mathbf{v}^* = \text{argmax}_{\mathbf{v}} \left(\underbrace{\sum_i \log P(y_i | x_i, \mathbf{v})}_{Log-Likelihood} - \underbrace{\frac{1}{2\sigma^2} \sum_k v_k^2}_{Gaussian\ Prior} \right)$$

(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?:

$$T^* = \text{argmax}_T \log P(T|S)$$

- 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 $\max_{u,v \in \mathcal{T}} \pi[n,u,v]$

The Viterbi Algorithm: Recursive Definitions

- **Base case:**

$$\begin{aligned} \pi[0,*,*] &= \log 1 = 0 \\ \pi[0,u,v] &= \log 0 = -\infty \text{ for all other } u,v \end{aligned}$$

here $*$ is a special tag padding the beginning of the sentence.

- **Recursive case:** for $i = 1 \ldots n$, for all u, v,

$$\pi[i,u,v] = \max_{t \in \mathcal{T} \cup \{*\}} \{\pi[i-1,t,u] + Score(S,i,t,u,v)\}$$

Backpointers allow us to recover the max probability sequence:

$$\text{BP}[i,u,v] = \text{argmax}_{t \in \mathcal{T} \cup \{*\}} \{\pi[i-1,t,u] + Score(S,i,t,u,v)\}$$

Where $Score(S,i,t,u,v) = \log P(v \mid t,u,w_1,\ldots,w_n,i)$

Identical to Viterbi for HMMs, except for the definition of $Score(S,i,t,u,v)$

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

$$P(word|tag)$$

is difficult in this domain

FAQ Segmentation: McCallum et. al

```
    <head>X-NNTP-POSTER: NewsHound v1.33
    <head>
    <head>Archive name: acorn/faq/part2
    <head>Frequency: monthly
    <head>
<question>2.6) What configuration of serial cable should I use
  <answer>
  <answer>   Here follows a diagram of the necessary connections
  <answer>programs to work properly. They are as far as I know t
  <answer>agreed upon by commercial comms software developers fo
  <answer>
  <answer>   Pins 1, 4, and 8 must be connected together inside
  <answer>is to avoid the well known serial port chip bugs. The
```

FAQ Segmentation: Line Features

```
begins-with-number
begins-with-ordinal
begins-with-punctuation
begins-with-question-word
begins-with-subject
blank
contains-alphanum
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

```
    <head>X-NNTP-POSTER: NewsHound v1.33
    <head>
    <head>Archive name: acorn/faq/part2
    <head>Frequency: monthly
    <head>
<question>2.6) What configuration of serial cable should I use

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

              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

```
<question>2.6) What configuration of serial cable should I use
```

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

$P($"2.6) What configuration of serial cable should I use" | question$) =$
$P($ 2.6$)$ | question$)\times$
$P(What$ | question$)\times$
$P(configuration$ | question$)\times$
$P(of$ | question$)\times$
$P(serial$ | question$)\times$
$\ldots$

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

FAQ Segmentation: McCallum et. al

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

```
<question>2.6) What configuration of serial cable should I use
```

⇒

```
<question>begins-with-number contains-alphanum contains-nonspac
```

- Use a language model again:

$P($"2.6) What configuration of serial cable should I use" | question$) =$
$P($begins-with-number | question$)\times$
$P($contains-alphanum | question$)\times$
$P($contains-nonspace | question$)\times$
$P($contains-number | question$)\times$
$P($prev-is-blank | question$)\times$

FAQ Segmentation: Results

Method	Precision	Recall
ME-Stateless	0.038	0.362
TokenHMM	0.276	0.140
FeatureHMM	0.413	0.529
MEMM	0.867	0.681

- Precision and recall results are for recovering segments
- ME-stateless is a log-linear model that treats every sentence seperately (no dependence between adjacent tags)
- TokenHMM is an HMM with fi rst 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")

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 $S = w_1 \dots w_n$
- Each tag sequence T has a conditional probability

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

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

- Estimate $P(t_j \mid w_1 \dots w_n, t_{j-2}, t_{j-1})$ using log-linear models
- Use the Viterbi algorithm to compute

$$\text{argmax}_{T \in \mathcal{T}^n} \log P(T \mid S)$$

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)?

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

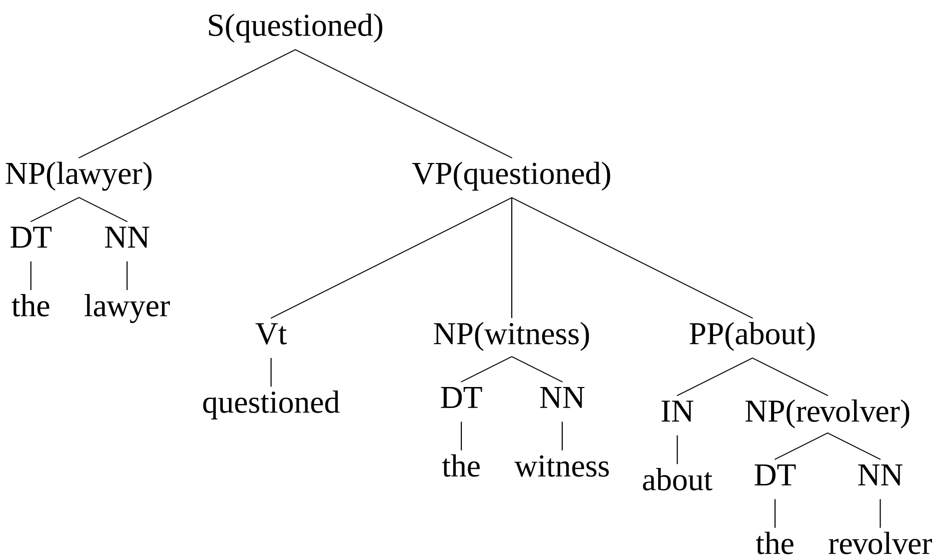
$$P(T \mid S) = \prod_{i=1}^{m} P(d_i \mid d_1 \ldots d_{i-1}, S)$$

- 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

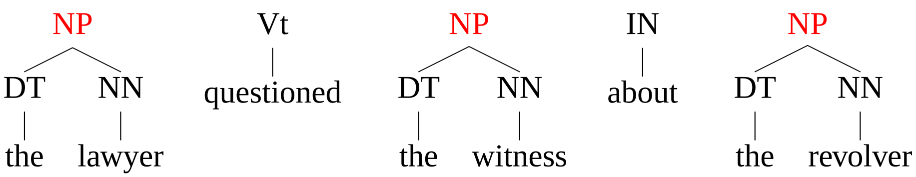
DT	NN	Vt	DT	NN	IN	DT	NN
the	lawyer	questioned	the	witness	about	the	revolver

- 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 $\rangle$

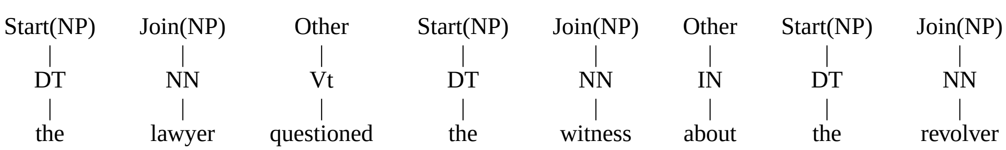
Layer 2: Chunks



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

$\langle d_1 \ldots d_{2n} \rangle = \langle$ DT, NN, Vt, DT, NN, IN, DT, NN,
Start(NP), Join(NP), Other, Start(NP), Join(NP),
Other, Start(NP), Join(NP)$\rangle$

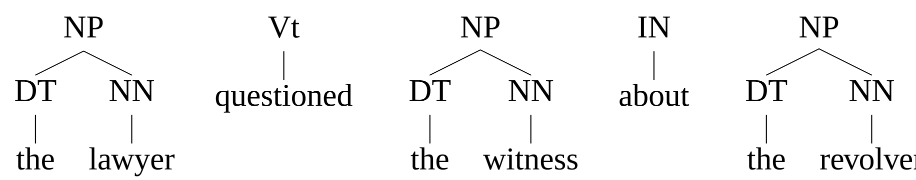
Layer 3: Remaining Structure

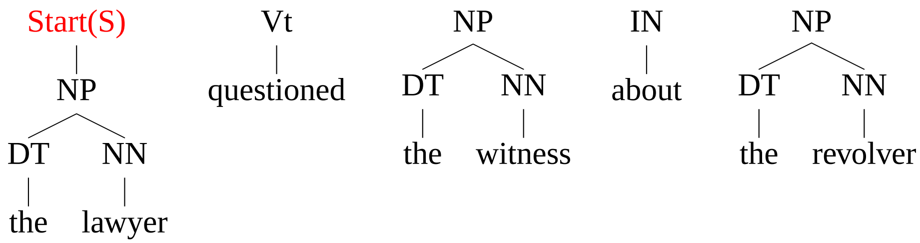
Alternate Between Two Classes of Actions:

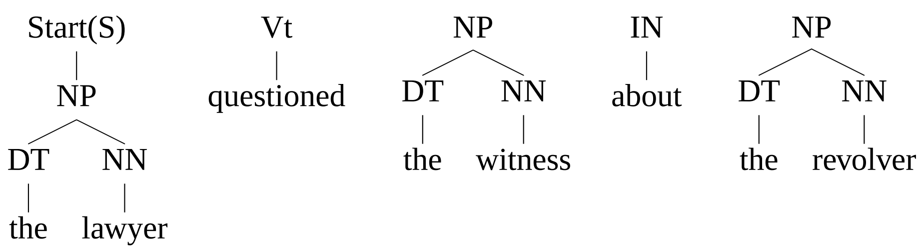
- Join(X) or Start(X), where X is a label (NP, S, VP etc.)
- Check=YES or Check=NO

Meaning of these actions:

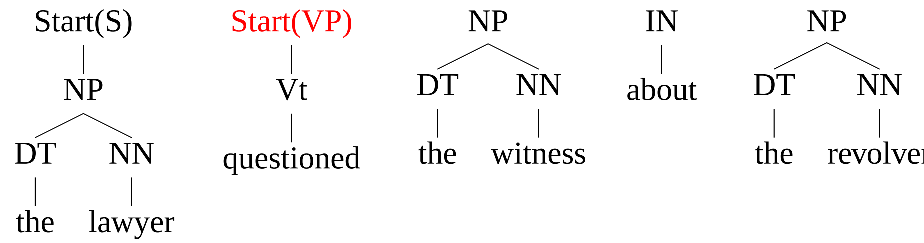
- Start(X) starts a new constituent with label X
  (always acts on leftmost constituent with no start or join label above it)
- Join(X) continues a constituent with label X
  (always acts on leftmost constituent with no start or join label above it)
- Check=NO does nothing
- Check=YES takes previous Join or Start action, and converts it into a completed constituent

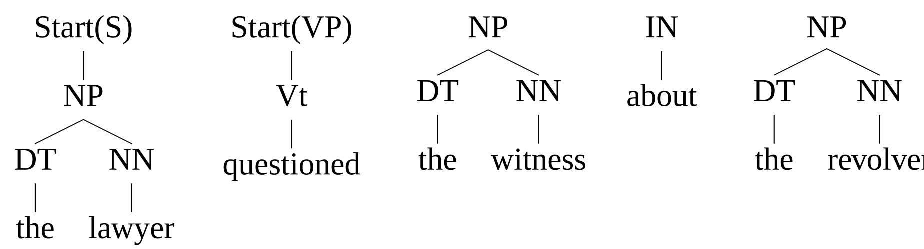




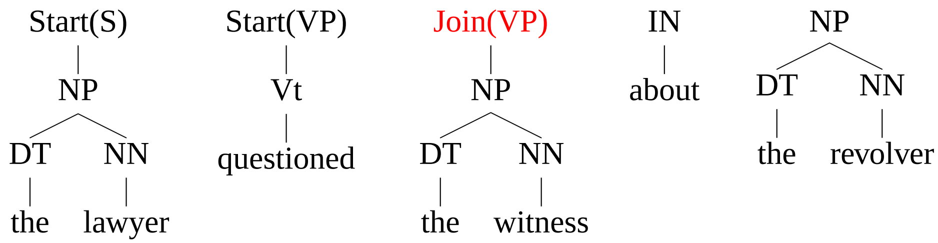


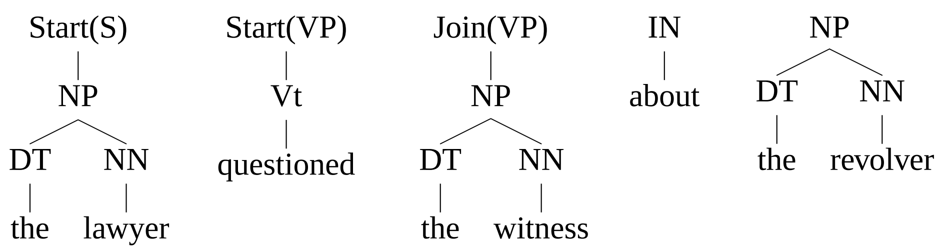
Check=NO



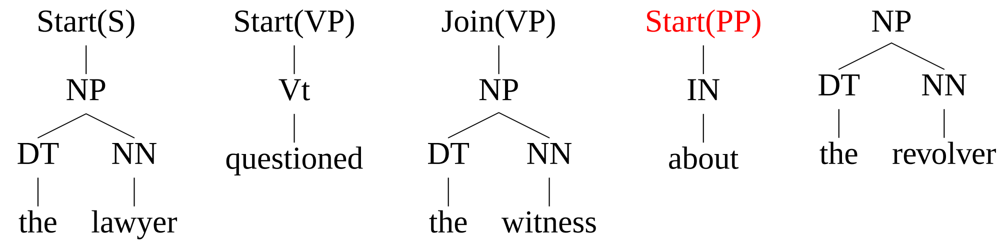


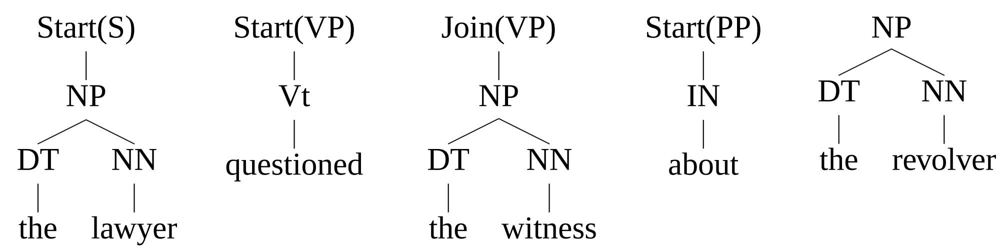
Check=NO



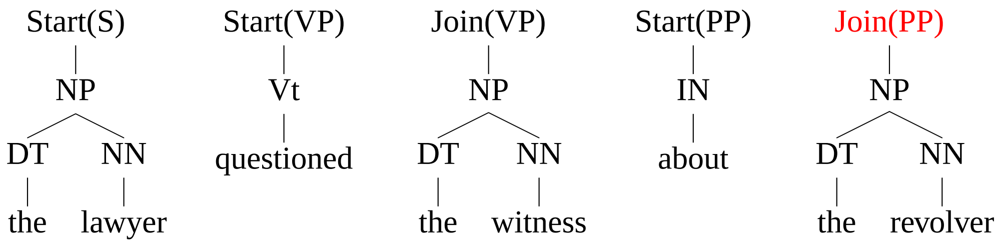


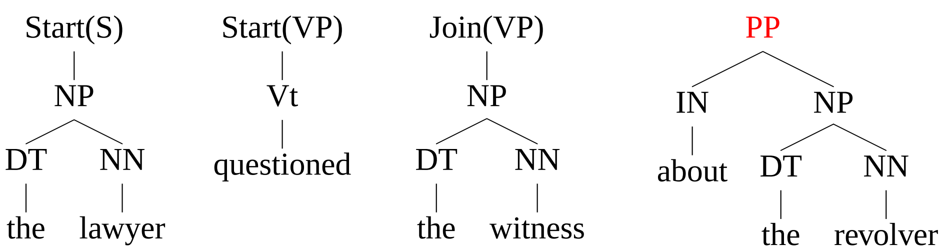
Check=NO



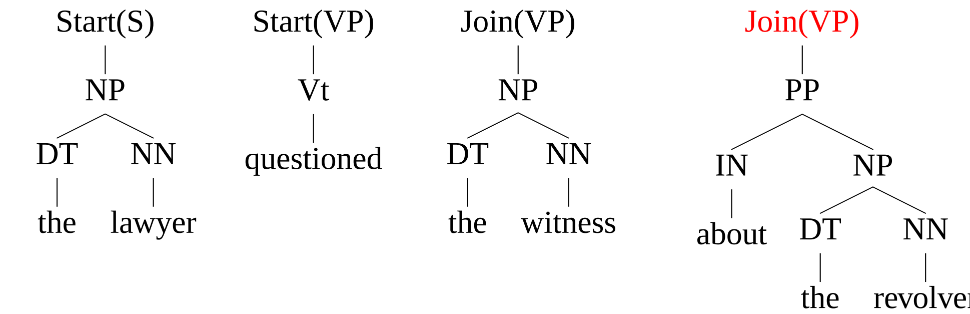


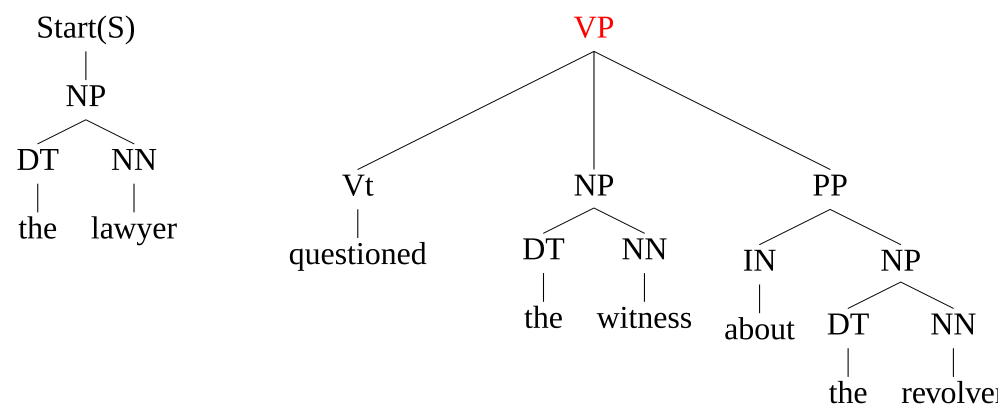
Check=NO



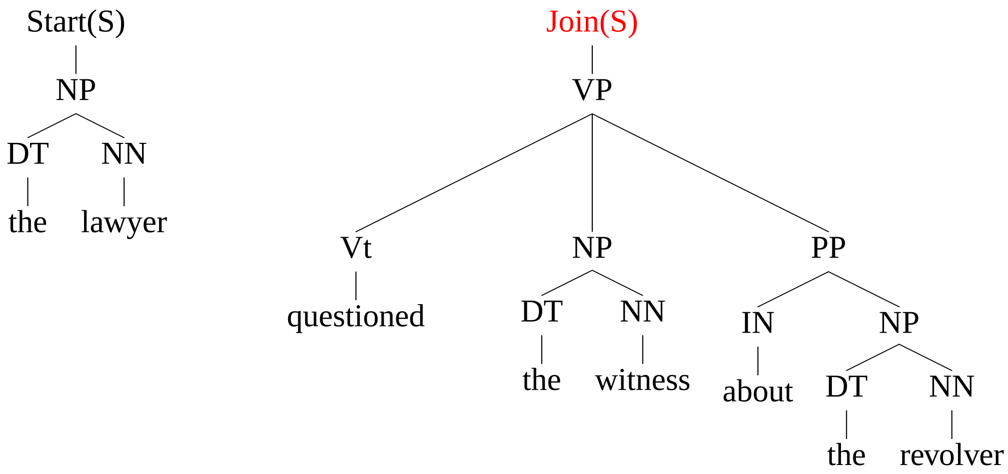


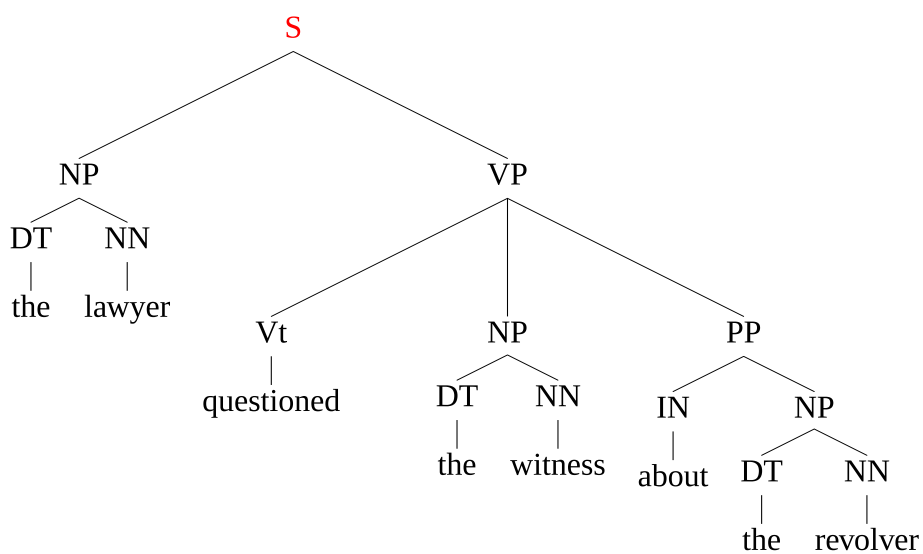
Check=YES





Check=YES





Check=YES

The Final Sequence of decisions

$\langle d_1 \ldots d_m \rangle = \langle$ DT, NN, Vt, DT, NN, IN, DT, NN,
Start(NP), Join(NP), Other, 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 **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)

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:

$$P(d_i \mid d_1 \ldots d_{i-1}, S) = \frac{e^{\mathbf{f}(\langle d_1 \ldots d_{i-1}, S \rangle, d_i) \cdot \mathbf{v}}}{\sum_{d \in \mathcal{A}} e^{\mathbf{f}(\langle d_1 \ldots d_{i-1}, S \rangle, d) \cdot \mathbf{v}}}$$

where:

$\langle d_1 \ldots d_{i-1}, S \rangle$	is the history
d_i	is the outcome
$\mathbf{f}$	maps a history/outcome pair to a feature vector
$\mathbf{v}$	is a parameter vector
$\mathcal{A}$	is set of possible actions

Applying a Log-Linear Model

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

$$P(d_i \mid d_1 \ldots d_{i-1}, S) = \frac{e^{\mathbf{f}(\langle d_1 \ldots d_{i-1}, S \rangle, d_i) \cdot \mathbf{v}}}{\sum_{d \in \mathcal{A}} e^{\mathbf{f}(\langle d_1 \ldots d_{i-1}, S \rangle, d) \cdot \mathbf{v}}}$$

- 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

$$P(t_j \mid w_1 \ldots w_n, j, t_1 \ldots t_{j-1}) = P(t_j \mid w_1 \ldots w_n, j, t_{j-2} \ldots t_{j-1})$$

- 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