Sequence Models

- Hidden Markov Models (HMM)

- MaxEnt Markov Models (MEMM)

Many slides from Michael Collins

Overview and HMMs

- 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

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

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

Supervised Learning Problems

- We have training examples $x^{(i)}, y^{(i)}$ for $i = 1 \ldots m$. Each $x^{(i)}$ is an input, each $y^{(i)}$ is a label.
- Task is to learn a function f mapping inputs x to labels $f(x)$

Supervised Learning Problems

- We have training examples $x^{(i)}, y^{(i)}$ for $i = 1 \ldots m$. Each $x^{(i)}$ is an input, each $y^{(i)}$ is a label.
- Task is to learn a function f mapping inputs x to labels $f(x)$
- Conditional models:
  - Learn a distribution $p(y|x)$ from training examples
  - For any test input x, define $f(x) = \arg\max_y p(y|x)$

Generative Models

- We have training examples $x^{(i)}, y^{(i)}$ for $i = 1 \ldots m$. Task is to learn a function f mapping inputs x to labels $f(x)$.

Generative Models

- We have training examples $x^{(i)}, y^{(i)}$ for $i = 1 \dots m$. Task is to learn a function f mapping inputs x to labels $f(x)$.

- Generative models:
  - Learn a distribution $p(x, y)$ from training examples
  - Often we have $p(x, y) = p(y)p(x|y)$

Generative Models

- We have training examples $x^{(i)}, y^{(i)}$ for $i = 1 \ldots m$. Task is to learn a function f mapping inputs x to labels $f(x)$.

- Generative models:
  - Learn a distribution $p(x, y)$ from training examples
  - Often we have $p(x, y) = p(y)p(x|y)$

- Note: we then have

$$p(y|x) = \frac{p(y)p(x|y)}{p(x)}$$

where $p(x) = \sum_y p(y)p(x|y)$

Decoding with Generative Models

- We have training examples $x^{(i)}, y^{(i)}$ for $i = 1 \ldots m$. Task is to learn a function f mapping inputs x to labels $f(x)$.
- Generative models:
  - Learn a distribution $p(x, y)$ from training examples
  - Often we have $p(x, y) = p(y)p(x|y)$
- Output from the model:

$$
\begin{aligned}
f(x) &= \arg\max_y p(y|x) \\
&= \arg\max_y \frac{p(y)p(x|y)}{p(x)} \\
&= \arg\max_y p(y)p(x|y)
\end{aligned}
$$

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 is the i'th word in the sentence)
- We have a tag sequence $y = y_1, y_2, \ldots, y_n$ (y_i is the i'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 $y_{n+1} =$ 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 y_0 = y_-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 $\texttt{D N V STOP}$, then

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

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

Why the Name?

$$p(x_1 \ldots x_n, y_1 \ldots y_n) = \underbrace{q(\text{STOP}|y_{n-1}, y_n) \prod_{j=1}^{n} q(y_j \mid y_{j-2}, y_{j-1})}_{\text{Markov Chain}}$$

$$\times \underbrace{\prod_{j=1}^{n} e(x_j \mid y_j)}_{x_j\text{'s are observed}}$$

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

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

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

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

Dealing with Low-Frequency Words: An Example

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 ≥ 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
...

- Inference and the Viterbi Algorithm

The Viterbi Algorithm

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 $y_i \in \mathcal{S}$ for $i = 1 \ldots n$, and $y_{n+1} =$ 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 $y_0 = y_{-1} =$ *, and $y_{n+1} =$ STOP.

Brute Force Search is Hopelessly Inefficient

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

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

where the $\arg\max$ is taken over all sequences $y_1 \dots y_{n+1}$ such that $y_i \in \mathcal{S}$ for $i = 1 \dots n$, 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:
$$S_{-1} = S_0 = \{*\}$$
$$S_k = S \quad \text{for } k \in \{1 \ldots n\}$$
- 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
$$\pi(k, u, v) \quad = \quad \text{maximum probability of a tag sequence ending in tags } u, v \text{ at position } k$$

that is,
$\pi(k, u, v) = \max_{\langle y_{-1}, y_0, y_1, \ldots, y_k \rangle : y_{k-1} = u, y_k = v} r(y_{-1}, y_0, y_1 \ldots y_k)$

A Recursive Definition

Base case:

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

Recursive definition:

For any $k \in \{1 \dots n\}$, for any $u \in \mathcal{S}_{k-1}$ and $v \in \mathcal{S}_k$:

$$\pi(k, u, v) = \max_{w \in \mathcal{S}_{k-2}} \left(\pi(k-1, w, u) \times q(v|w, u) \times e(x_k|v) \right)$$

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: $\mathcal{S}_{-1} = \mathcal{S}_0 = \{*\}$, $\mathcal{S}_k = \mathcal{S}$ for $k \in \{1 \ldots n\}$

Algorithm:

- For $k = 1 \ldots n$,
  - For $u \in \mathcal{S}_{k-1}$, $v \in \mathcal{S}_k$,

$$\pi(k, u, v) = \max_{w \in \mathcal{S}_{k-2}} \left(\pi(k-1, w, u) \times q(v|w, u) \times e(x_k|v)\right)$$

- **Return** $\max_{u \in \mathcal{S}_{n-1}, v \in \mathcal{S}_n} \left(\pi(n, u, v) \times q(\text{STOP}|u, v)\right)$

The Viterbi Algorithm with Backpointers

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

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

Definition: $\mathcal{S}_{-1} = \mathcal{S}_0 = \{*\}$, $\mathcal{S}_k = \mathcal{S}$ for $k \in \{1 \dots n\}$
Algorithm:

- For $k = 1 \dots n$,
  - For $u \in \mathcal{S}_{k-1}$, $v \in \mathcal{S}_k$,

$$\begin{aligned}
\pi(k,u,v) &= \max_{w \in \mathcal{S}_{k-2}} \left(\pi(k-1,w,u) \times q(v|w,u) \times e(x_k|v)\right) \\
bp(k,u,v) &= \arg\max_{w \in \mathcal{S}_{k-2}} \left(\pi(k-1,w,u) \times q(v|w,u) \times e(x_k|v)\right)
\end{aligned}$$

- Set $(y_{n-1}, y_n) = \arg\max_{(u,v)} (\pi(n,u,v) \times q(\mathsf{STOP}|u,v))$
- For $k = (n-2) \dots 1$, $y_k = bp(k+2, y_{k+1}, y_{k+2})$
- **Return** the tag sequence $y_1 \dots y_n$

The Viterbi Algorithm: Running Time

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

A Simple Bi-gram Example:

(X, Y): P(X/Y),   POS tags for “bears fish” ?

- noun * .80
- Verb * .10
- STOP noun .50
- STOP verb .50
- noun verb .77
- verb noun .65
- noun noun .0001
- nerb verb .0001

bears noun .02

bears verb .02

fish verb .07

fish noun .08

Answer

- bears: noun
- fish: verb

The Forward Algorithm

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

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

Definition: $\mathcal{S}_{-1} = \mathcal{S}_0 = \{*\}$, $\mathcal{S}_k = \mathcal{S}$ for $k \in \{1 \ldots n\}$

Algorithm:

- For $k = 1 \ldots n$,
  - For $u \in \mathcal{S}_{k-1}$, $v \in \mathcal{S}_k$,

$$\pi(k,u,v) = \mathop{\text{Sum}}_{w \in \mathcal{S}_{k-2}} \left(\pi(k-1,w,u) \times q(v|w,u) \times e(x_k|v)\right)$$

- **Return** $\mathop{\text{Sum}}_{u \in \mathcal{S}_{n-1}, v \in \mathcal{S}_n} \left(\pi(n,u,v) \times q(\text{STOP}|u,v)\right)$

Pros and Cons

- Hidden markov model taggers are very simple to train (just need to compile counts from the training corpus) If you already have a labeled training set.

  Use forward-backward algorithms in the unsupervised setting.
- Perform relatively well (over 90% performance on named entity recognition)
- Main difficulty is modeling

$$e(word \mid tag)$$

can be very difficult if "words" are complex

- MaxEnt Markov Models (MEMMs)

Log-Linear Models for Tagging

- We have an input sentence $w_{[1:n]} = w_1, w_2, \ldots, w_n$ (w_i is the i'th word in the sentence)
- We have a tag sequence $t_{[1:n]} = t_1, t_2, \ldots, t_n$ (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 $w_{[1:n]}$ and tag sequence $t_{[1:n]}$ of the same length.
(Note: contrast with HMM that defines $p(t_1 \ldots t_n, w_1 \ldots w_n)$)

- Then the most likely tag sequence for $w_{[1:n]}$ is

$$t^*_{[1:n]} = \text{argmax}_{t_{[1:n]}} p(t_{[1:n]} | w_{[1:n]})$$

How to model $p(t_{[1:n]}|w_{[1:n]})$?

A Trigram Log-Linear Tagger:

$$p(t_{[1:n]}|w_{[1:n]}) = \prod_{j=1}^{n} p(t_j \mid w_1 \ldots w_n, t_1 \ldots t_{j-1})$$ 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, $\ldots\}$

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

Recap: Feature Vector Representations in 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 $f : \mathcal{X} \times \mathcal{Y} \rightarrow \mathbb{R}$
  (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** $f(x, y) \in \mathbb{R}^m$ 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:

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

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

$$\ldots$$

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

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

$$v^* = \text{argmax}_v \left(\underbrace{\sum_i \log p(y_i | x_i; v)}_{\color{red}{Log-Likelihood}} - \underbrace{\frac{\lambda}{2} \sum_k v_k^2}_{\color{red}{Regularizer}} \right)$$

(see last lecture on log-linear models)

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

The Viterbi Algorithm

Problem: for an input $w_1 \dots w_n$, find

$$\arg\max_{t_1 \dots t_n} p(t_1 \dots t_n \mid w_1 \dots w_n)$$

We assume that p takes the form

$$p(t_1 \dots t_n \mid w_1 \dots w_n) = \prod_{i=1}^{n} q(t_i | t_{i-2}, t_{i-1}, w_{[1:n]}, i)$$

(In our case $q(t_i | t_{i-2}, t_{i-1}, w_{[1:n]}, i)$ is the estimate from a log-linear model.)

The Viterbi Algorithm

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

$$r(t_1 \dots t_k) = \prod_{i=1}^{k} q(t_i | t_{i-2}, t_{i-1}, w_{[1:n]}, i)$$

- Define a dynamic programming table

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

that is,

$$\pi(k, u, v) = \max_{\langle t_1, \dots, t_{k-2} \rangle} r(t_1 \dots t_{k-2}, u, v)$$

A Recursive Definition

Base case:

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

Recursive definition:

For any $k \in \{1 \ldots n\}$, for any $u \in \mathcal{S}_{k-1}$ and $v \in \mathcal{S}_k$:

$$\pi(k, u, v) = \max_{t \in \mathcal{S}_{k-2}} \left(\pi(k-1, t, u) \times q(v|t, u, w_{[1:n]}, k) \right)$$

where $\mathcal{S}_k$ is the set of possible tags at position k

The Viterbi Algorithm with Backpointers

Input: a sentence $w_1 \dots w_n$, log-linear model that provides $q(v|t,u,w_{[1:n]},i)$ for any tag-trigram t,u,v, for any $i \in \{1 \dots n\}$

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

Algorithm:

- For $k = 1 \dots n$,
  - For $u \in \mathcal{S}_{k-1}$, $v \in \mathcal{S}_k$,

$$\begin{aligned} \pi(k,u,v) &= \max_{t \in \mathcal{S}_{k-2}} \left(\pi(k-1,t,u) \times q(v|t,u,w_{[1:n]},k)\right) \\ bp(k,u,v) &= \arg\max_{t \in \mathcal{S}_{k-2}} \left(\pi(k-1,t,u) \times q(v|t,u,w_{[1:n]},k)\right) \end{aligned}$$

- Set $(t_{n-1}, t_n) = \arg\max_{(u,v)} \pi(n,u,v)$
- For $k = (n-2) \dots 1$, $t_k = bp(k+2, t_{k+1}, t_{k+2})$
- **Return** the tag sequence $t_1 \dots t_n$

Summary

- Key ideas in log-linear taggers:
  - Decompose
$$p(t_1 \dots t_n | w_1 \dots w_n) = \prod_{i=1}^{n} p(t_i | t_{i-2}, t_{i-1}, w_1 \dots w_n)$$
  - Estimate
$$p(t_i | t_{i-2}, t_{i-1}, w_1 \dots w_n)$$
    using a log-linear model
  - For a test sentence $w_1 \dots w_n$, use the Viterbi algorithm to find
$$\arg\max_{t_1 \dots t_n} \left(\prod_{i=1}^{n} p(t_i | t_{i-2}, t_{i-1}, w_1 \dots w_n) \right)$$
- Key advantage over HMM taggers: flexibility in the features they can use