

Part of Speech Tagging

Informatics 2A: Lecture 16

John Longley

School of Informatics
University of Edinburgh

24 October 2013

- 1 Automatic POS tagging: the problem
- 2 Methods for tagging
 - Unigram tagging
 - Bigram tagging
 - Tagging using Hidden Markov Models: Viterbi algorithm

Reading: Jurafsky & Martin, chapters (5 and) 6.

Benefits of Part of Speech Tagging

- Essential preliminary to (anything that involves) parsing.
- Can help with **speech synthesis**. For example, try saying the sentences below out loud.
- Can help with **determining authorship**: are two given documents written by the same person? **Forensic linguistics**.

- 1 *Have you read 'The Wind in the Willows'?* (noun)
- 2 *The clock has stopped. Please wind it up.* (verb)
- 3 *The students tried to protest.* (verb)
- 4 *The students' protest was successful.* (noun)

Corpus annotation

A **corpus** (plural **corpora**) is a computer-readable collection of NL text (or speech) used as a source of information about the language: e.g. what words/constructions can occur in practice, and with what frequencies.

The usefulness of a corpus can be enhanced by *annotating* each word with a POS tag, e.g.

```
Our/PRP\$ enemies/NNS are/VBP innovative/JJ and/CC  
resourceful/JJ ,/, and/CC so/RB are/VB we/PRP ./.  
They/PRP never/RB stop/VB thinking/VBG about/IN new/JJ  
ways/NNS to/TO harm/VB our/PRP\$ country/NN and/CC  
our/PRP\$ people/NN, and/CC neither/DT do/VB we/PRP ./.
```

Typically done by an automatic tagger, then hand-corrected by a native speaker, in accordance with specified **tagging guidelines**.

POS tagging: difficult cases

Even for humans, tagging sometimes poses difficult decisions. Various tests can be applied, but they don't always yield clear answers.

E.g. Words in **-ing**: adjectives (JJ), or verbs in gerund form (VBG)?

a boring/JJ lecture

a *very* boring lecture

? a lecture that bores

the ruling/VBG classes

*the *very* ruling classes

the classes that rule

a revolving/VBG? door

*a *very* revolving door

a door that revolves

*the door seems revolving

sparkling/JJ? lemonade

? *very* sparkling lemonade

lemonade that sparkles

the lemonade seems sparkling

In view of such problems, we can't expect 100% accuracy from an automatic tagger.

Word types and tokens

- Need to distinguish **word tokens** (particular occurrences in a text) from **word types** (distinct vocabulary items).
- We'll count different inflected or derived forms (e.g. break, breaks, breaking) as distinct word types.
- A single word type (e.g. **still**) may appear with several POS.
- But most words have a clear **most frequent** POS.

Clicker question: How many tokens and types in the following?
Ignore case and punctuation.

Esau sawed wood. Esau Wood would saw wood. Oh, the wood Wood would saw!

- 1 14 tokens, 6 types
- 2 14 tokens, 7 types
- 3 14 tokens, 8 types
- 4 None of the above.

Extent of POS Ambiguity

The Brown corpus (1,000,000 word tokens) has 39,440 different word types.

- 35340 have only 1 POS tag anywhere in corpus (89.6%)
- 4100 (10.4%) have 2 to 7 POS tags

So why does just 10.4% POS-tag ambiguity by **word type** lead to difficulty?

This is thanks to *Zipfian distribution*: many high-frequency words have more than one POS tag.

In fact, more than 40% of the **word tokens** are ambiguous.

He wants **to/TO** go.

He went **to/IN** the store.

He wants **that/DT** hat.

It is obvious **that/CS** he wants a hat.

He wants a hat **that/WPS** fits.

Some tagging strategies

We'll look at several methods or strategies for automatic tagging.

- One simple strategy: just assign to each word its *most common tag*. (So **still** will *always* get tagged as an adverb — never as a noun, verb or adjective.) Call this *unigram* tagging, since we only consider one token at a time.
- Surprisingly, even this crude approach typically gives around 90% accuracy. (State-of-the-art is 96–98%).
- Can we do better? We'll look briefly at **bigram tagging**, then at **Hidden Markov Model tagging**.

Bigram tagging

We can do much better by looking at *pairs of adjacent tokens*. For each word (e.g. **still**), tabulate the frequencies of each possible POS *given the POS of the preceding word*.

Example (with made-up numbers):

still	DT	MD	JJ	...
NN	8	0	6	
JJ	23	0	14	
VB	1	12	2	
RB	6	45	3	

Given a new text, tag the words from left to right, assigning each word the most likely tag given the preceding one.

Could also consider **trigram** (or more generally **n-gram**) tagging, etc. But the frequency matrices would quickly get very large, and also (for realistic corpora) too 'sparse' to be really useful.

Problems with bigram tagging

- One incorrect tagging choice might have knock-on effects:

	The	still	smoking	remains	of	the	campfire
<i>Intended:</i>	DT	RB	VBG	NNS	IN	DT	NN
<i>Bigram:</i>	DT	JJ	NN	VBZ	...		

- No lookahead: choosing the 'most probable' tag at one stage might lead to highly improbable choice later.

	The	still	was	smashed
<i>Intended:</i>	DT	NN	VBD	VBN
<i>Bigram:</i>	DT	JJ	VBD?	

We'd prefer to find the *overall most likely* tagging sequence given the bigram frequencies. This is what the **Hidden Markov Model (HMM)** approach achieves.

Hidden Markov Models

- The idea is to model the agent that might have generated the sentence by a semi-random process that outputs a sequence of words.
- Think of the output as **visible** to us, but the internal states of the process (which contain POS information) as **hidden**.
- For some outputs, there might be several possible ways of generating them i.e. several sequences of internal states. Our aim is to compute the sequence of hidden states with the **highest probability**.
- Specifically, our processes will be '**NFAs with probabilities**'. Simple, though not a very flattering model of human language users!

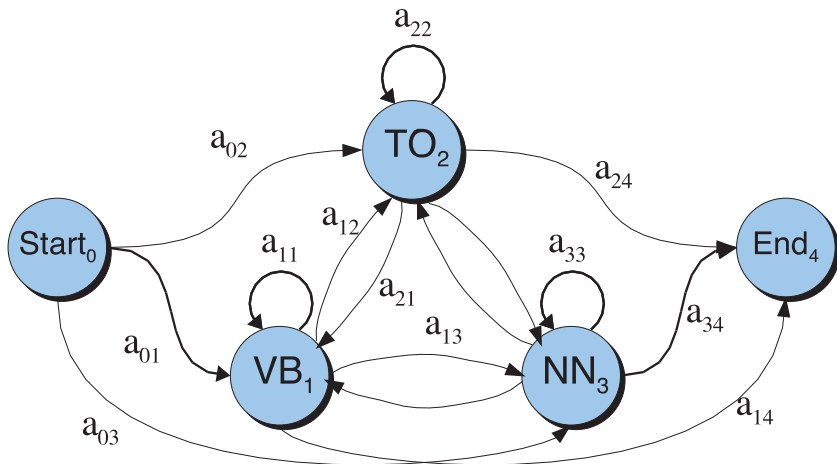
Definition of Hidden Markov Models

For our purposes, a **Hidden Markov Model (HMM)** consists of:

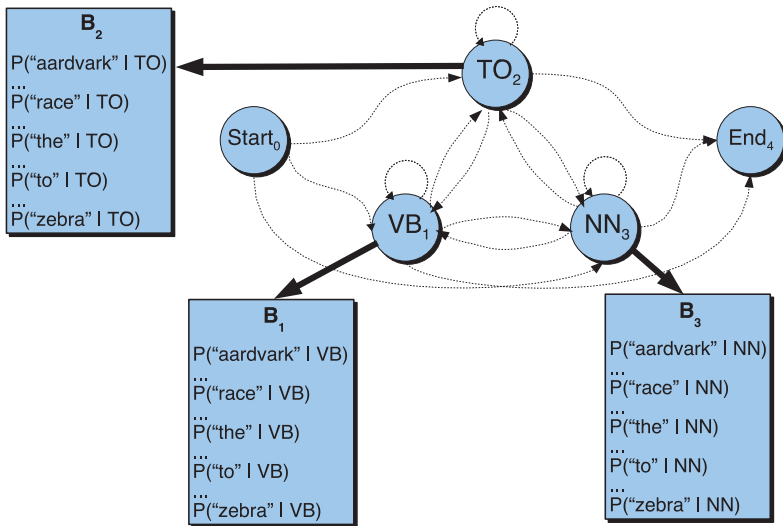
- A set $Q = \{q_0, q_1, \dots, q_n\}$ of **states**, with q_0 the start state. (Our non-start states will correspond to *parts-of-speech*).
- A **transition probability** matrix $A = (a_{ij} \mid 0 \leq i \leq n, 1 \leq j \leq n)$, where a_{ij} is the probability of jumping from q_i to q_j . For each i , we require $\sum_{j=1}^n a_{ij} = 1$.
- For each non-start state q_i and word type w , an **emission probability** $b_i(w)$ of outputting w upon entry into q_i . (Ideally, for each i , we'd have $\sum_w b_i(w) = 1$.)

We also suppose we're given an **observed sequence** w_1, w_2, \dots, w_T of word tokens generated by the HMM.

Transition Probabilities



Emission Probabilities



Transition and Emission Probabilities

	VB	TO	NN	PPPS
<s>	.019	.0043	.041	.67
VB	.0038	.035	.047	.0070
TO	.83	0	.00047	0
NN	.0040	.016	.087	.0045
PPPS	.23	.00079	.001	.00014

	I	want	to	race
VB	0	.0093	0	.00012
TO	0	0	.99	0
BB	0	.000054	0	.00057
PPSS	.37	0	0	0

How Do we Search for Best Tag Sequence?

We have defined an HMM, but how do we use it? We are given a **word sequence** and must find their corresponding **tag sequence**.

- It's easy to compute the probability of generating a word sequence $w_1 \dots w_T$ via a specific tag sequence $t_1 \dots t_T$: let t_0 denote the start state, and compute

$$\prod_{i=1}^n P(t_i | t_{i-1}) \cdot P(w_i | t_i) \quad (1)$$

using the transition and emission probabilities.

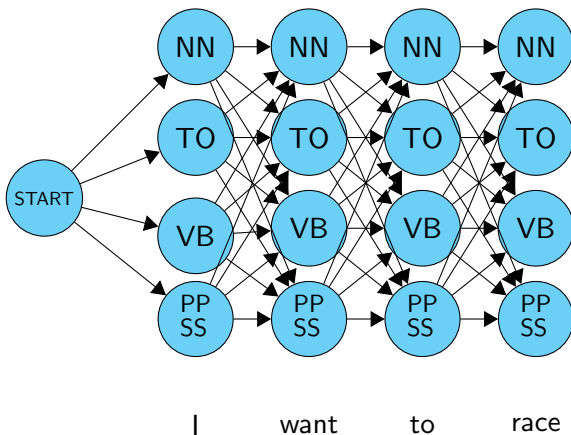
- **But how do we find the most likely tag sequence?**
- We can do this efficiently using **dynamic programming** and the **Viterbi algorithm**.

Clicker Question

Given n word tokens and on average T choices per token, how many tag sequences do we have to evaluate?

- 1 $|T|$ tag sequences
- 2 n tag sequences
- 3 $|T| \times n$ tag sequences
- 4 $|T|^n$ tag sequences

The HMM trellis



The Viterbi Algorithm

q_4	NN	0				
q_3	TO	0				
q_2	VB	0				
q_1	PPSS	0				
q_0	start	1.0				
		<s>	I	want	to	race
			w_1	w_2	w_3	w_4

- 1 Create probability matrix, with one column for each observation (i.e., word token), and one row for each non-start state (i.e., POS tag).
- 2 We proceed by filling cells, column by column.
- 3 The entry in column i , row j will be the **probability of the most probable route to state q_j that emits $w_1 \dots w_i$.**

The Viterbi Algorithm

q_4	NN	0	$1.0 \times .041 \times 0$			
q_3	TO	0	$1.0 \times .0043 \times 0$			
q_2	VB	0	$1.0 \times .19 \times 0$			
q_1	PPSS	0	$1.0 \times .67 \times .37$			
q_0	start	1.0				
	<s>		I	want	to	race
			w_1	w_2	w_3	w_4

- For each state q_j at time i , compute

$$v_i(j) = \max_{k=1}^n v_{i-1}(k) a_{kj} b_j(w_i)$$
- $v_{i-1}(k)$ is **previous Viterbi path probability**, a_{kj} is **transition probability**, and $b_j(w_i)$ is **emission probability**.
- There's also an (implicit) **backpointer** from cell (i, j) to the relevant $(i-1, k)$, where k maximizes $v_{i-1}(k) a_{kj}$.

The Viterbi Algorithm

q_4	NN	0	0	$.025 \times .0012 \times 0.000054$		
q_3	TO	0	0	$.025 \times .00079 \times 0$		
q_2	VB	0	0	$.025 \times .23 \times .0093$		
q_1	PPSS	0	.025	$.025 \times .00014 \times 0$		
q_0	start	1.0				
	<s>			want	to	race
		w_1		w_2	w_3	w_4

- For each state q_j at time i , compute

$$v_i(j) = \max_{k=1}^n v_{i-1}(k) a_{kj} b_j(w_i)$$
- $v_{i-1}(k)$ is **previous Viterbi path probability**, a_{kj} is **transition probability**, and $b_j(w_i)$ is **emission probability**.
- There's also an (implicit) **backpointer** from cell (i, j) to the relevant $(i-1, k)$, where k maximizes $v_{i-1}(k) a_{kj}$.

The Viterbi Algorithm

q_4	NN	0	0	.000000002	$.000053 \times .047 \times 0$	
q_3	TO	0	0	0	$.000053 \times .035 \times .99$	
q_2	VB	0	0	.00053	$.000053 \times .0038 \times 0$	
q_1	PPSS	0	.025	0	$.000053 \times .0070 \times 0$	
q_0	start	1.0				
	<s>		want		to	race
		w_1	w_2		w_3	w_4

- For each state q_j at time i , compute

$$v_i(j) = \max_{k=1}^n v_{i-1}(k) a_{kj} b_j(w_i)$$
- $v_{i-1}(k)$ is **previous Viterbi path probability**, a_{kj} is **transition probability**, and $b_j(w_i)$ is **emission probability**.
- There's also an (implicit) **backpointer** from cell (i, j) to the relevant $(i-1, k)$, where k maximizes $v_{i-1}(k) a_{kj}$.

The Viterbi Algorithm

q_4	NN	0	0	.0000000020	0	.0000018 × .00047 × .00057
q_3	TO	0	0	0	.0000018	.0000018 × 0 × 0
q_2	VB	0	0	.00053	0	.0000018 × .83 × .00012
q_1	PPSS	0	.025	0	0	.0000018 × 0 × 0
q_0	start	1.0				
	<s>	I	want	to		race
		w_1	w_2	w_3		w_4

- For each state q_j at time i , compute

$$v_i(j) = \max_{k=1}^n v_{i-1}(k) a_{kj} b_j(w_i)$$
- $v_{i-1}(k)$ is **previous Viterbi path probability**, a_{kj} is **transition probability**, and $b_j(w_i)$ is **emission probability**.
- There's also an (implicit) **backpointer** from cell (i, j) to the relevant $(i - 1, k)$, where k maximizes $v_{i-1}(k) a_{kj}$.

The Viterbi Algorithm

q_4	NN	0	0	.000000002	0	4.8222e-13
q_3	TO	0	0	0	.0000018	0
q_2	VB	0	0	.00053	0	1.7928e-10
q_1	PPSS	0	.025	0	0	0
q_0	start	1.0				
	<s>		want	to	race	
		w_1	w_2	w_3	w_4	

- For each state q_j at time i , compute

$$v_i(j) = \max_{k=1}^n v_{i-1}(k) a_{kj} b_j(w_i)$$
- $v_{i-1}(k)$ is **previous Viterbi path probability**, a_{kj} is **transition probability**, and $b_j(w_i)$ is **emission probability**.
- There's also an (implicit) **backpointer** from cell (i, j) to the relevant $(i - 1, k)$, where k maximizes $v_{i-1}(k) a_{kj}$.

The Viterbi algorithm: second example

Let's now tag the newspaper headline:

deal talks fail

Note that each token here could be a noun (N) or a verb (V).
We'll use a toy HMM given as follows:

	to N	to V
from start	.8	.2
from N	.4	.6
from V	.8	.2

Transitions

	deal	fail	talks
N	.2	.05	.2
V	.3	.3	.3

Emissions

The Viterbi matrix

This time we'll omit the (trivial) first column, but show more of the working, as well as the backtrace pointers.

	deal	talks	fail
N	$.8 \times .2 = .16$	$\leftarrow .16 \times .4 \times .2 = .0128$ (since $.16 \times .4 > .06 \times .8$)	$\swarrow .0288 \times .8 \times .05 = .001152$ (since $.0128 \times .4 < 0.0288 \times .8$)
V	$.2 \times .3 = .06$	$\swarrow .16 \times .6 \times .3 = .0288$ (since $.16 \times .6 > .06 \times .2$)	$\swarrow .0128 \times .6 \times .3 = .002304$ (since $.0128 \times .6 > 0.0288 \times .2$)

Looking at the highest probability entry in the final column and chasing the backpointers, we see that the tagging **N N V** wins.