Applied Natural Language Processing

Dr Jeff Mitchell, University of Sussex

Autumn 2025

Where are we?

Previously

- Documents
  - document pre-processing
  - document classification and similarity
- Words
  - semantic relationships
  - distributional semantics

Still to go

- Word sequences
  - **part-of-speech tagging**
  - Named Entity Recognition
  - Question Answering

Part-of-speech tagging

Part 1

- Parts of speech (PoS)
  - what are they?
  - what are they useful for?
- Open and closed PoS classes
- PoS Tagsets
  - The Penn Treebank Tagset
- Simple PoS Tagging
  - PoS ambiguity
  - Unigram tagging
  - Evaluation

Part 2

- Sequence Labelling
- Hidden Markov Models (HMMs)
  - Forward algorithm
  - Viterbi algorithm

Parts of speech

Parts of speech

- Words can be categorised according to how they behave grammatically
- Traditionally, linguists distinguish about 9 or 10 lexical categories, referred to as **parts of speech (PoS)**:

- adjective
- adverb
- auxiliary verb
- conjunction
- determiner
- interjection
- noun
- preposition
- pronoun
- verb

Can you give an example of each one?

Are parts of speech useful?

Identifying parts of speech can be a useful pre-processing step

- Can help to disambiguate words
  - information retrieval
  - text-to-speech
  - document classification
- Tells us what sorts of words are likely to occur nearby:
  - adjectives often followed by nouns: *happy student*
  - personal pronouns often followed by verbs: *you laugh*
- Important for identifying larger grammatical structures
  - grammatical plausible sequences / parsing
  - named entity recognition
  - information extraction

Nouns and pronouns

- used to identify people, places and things
- **Nouns** often divided into
  - Proper nouns
    - *England, Kim, Microsoft*
  - Common nouns
    - count nouns: *window, tyre, idea*
    - mass nouns: *snow, rice, courage*
- **Pronouns** stand in place of a noun
  - *she, you, I , who*

Verbs and auxiliary verbs

- Actions and processes
  - run, chase, say, believe
  - it is *believed* that he *chased* the thief
- Auxiliary verbs usually precede a main verb
  - he *should* chase the thief

Adjectives and adverbs

- Adjectives:
  - properties and qualities
  - modify nouns
  - *green, small, clever, mythical*
- Adverbs:
  - usually modify verbs or verb phrases
  - *slowly, now, unfortunately, possibly, tomorrow*

Determiners

- A modifying word that determines the kind of reference a noun or noun group has
  - I would like *a* cake; vs
  - I would like *the* cake; vs
  - I would like *every* cake; vs
  - I would like *some* cake

- Also called articles

Prepositions and particles

- Prepositions
  - specify the relative positions of two words or elements
    - I saw the boy *on* the bridge; vs
    - I saw the boy *under* the bridge
    - *on, under, over, to, with, by*
- Particles
  - sometimes distinguished from prepositions
  - generally modify a verb; sometimes referred to as phrasal verbs
    - I tidied *up* the room
    - *up, down, at, by, to*

Conjunctions and interjections

- Conjunctions
  - join words and phrases together
    - co-ordinating: *and, but, or, nor*
    - sub-ordinating: *because, if, when, as, since, until*
    - correlative: *either … or …; both … and …*
- Interjections
  - exclamations without any grammatical connection to other words
  - *hey, ouch, darn, aha, huh*

Open and closed classes

- **Open** classes: so-called because they are not fixed
  - new words may be added fairly often
  - other words may go out of the language
  - content-bearing
- **Closed** classes: these classes are fixed
  - words are functional rather than content-bearing
  - frequently occurring and often short in length
  - may be considered as **stopwords** in some applications
  - may specify how different concepts in the sentence relate to each other

Language change

New English Words 2024

- boop
- ick
- bussin'
- AGI
- cuffing season
- shrinkflation

Archaic words, no longer in use?

- ambuscade
- beldam
- camelopard
- dispraise
- sanative

Do you know what these words mean?
What parts of speech do you think they are?

Part-of-Speech tagsets

- A tagset provides a set of labels for marking PoS classes.
- Different tag sets have been derived from work on text corpora:
  - Brown corpus: 80 tags
  - Penn Treebank: 45 tags
  - Susanne corpus: 350 tags
  - British National Corpus (BNC): 60 tags

The Penn TreeBank tagset (1)

CC	Coordinating conjunction	*and, but, or*
CD	Cardinal number	*one, two*
DT	Determiner	*the, some*
EX	Existential there	*there*
FW	Foreign word	*hoc*
IN	Preposition	*of, in, by*
JJ	Adjective	*big*
JJR	Adjective, comparative	*bigger*
JJS	Adjective, superlative	*biggest*
LS	List item marker	*1, One*
MD	Modal	*can, should*

The Penn TreeBank tagset (2)

NN	Noun, singular or mass	*dog*
NNS	Noun, plural	*dogs*
NNP	Proper noun, sing.	*Edinburgh*
NNPS	Proper noun, plural	*Orkneys*
PDT	Predeterminer	*all, both*
POS	Possessive ending	*'s*
PP	Personal pronoun	*I, you, she*
PP$	Possessive pronoun	*my, theirs*
RB	Adverb	*quickly*
RBR	Adverb, comparative	*faster*
RBS	Adverb, superlative	*fastest*

the penn treebank tagset (3)

RP	Particle	*up, off*
SYM	Symbol	*+, %, &*
TO	The word "to"	*to*
UH	Interjection	*oh, oops*
VB	verb, base form	*eat*
VBD	verb, past tense	*ate*
VBG	verb, gerund	*eating*
VBN	verb, past participle	*eaten*
VBP	Verb, non-3sg, pres	*eat*
VBZ	Verb, 3sg, pres	*eats*
WDT	Wh-determiner	*which, that*
WP	Wh-pronoun	*what, who*

the Penn treebank tagset (4)

WP$	Possessive-wh	*whose*
WRB	Wh-adverb	*how, where*
$	Dollar sign	$
#	Pound sign	#
“	Left quote	‘ , “
”	Right quote	’ , ”
(	Left parenthesis	(
)	Right parenthesis	)
,	Comma	,
.	Sentence-final punctuation	. ! ?
:	Mid-sentence punctuation	: ; — ...

Part-of-speech tagging

PoS tagging is the process of assigning a single part-of-speech tag to each word (and punctuation marker) in some text.›

"/" The/DT guys/NNS that/WDT make/VBP traditional/JJ hardware/NN are/VBP really/RB being/VBG obsoleted/VBN by/IN microprocessor-based/JJ machines/NNS ,/, "/" said/VBD Mr./NNP Benton/NNP ./.

PoS Tagging

Carrying out PoS tagging

- comparatively shallow form of processing
  - one tag per word
  - no larger structures created
- non-trivial
  - must resolve ambiguities
  - the same word can have different tags in different contexts

PoS ambiguity

- In the Brown corpus:
  - **11.5% of word types and 40% of word tokens** are ambiguous with respect to POS tag i.e., could be labelled with multiple PoS tags
  - Why is the percentage of ambiguous word tokens higher than the percentage of ambiguous word types?
- Which of the words below are ambiguous with respect to PoS?

Word	PoS?	Word	PoS?
dream		desert	
the		rebel	
word		bravely	
green		over	

Local vs global ambiguity

- Which words can have multiple PoS tags in the following sentence:

Fruit flies like a banana

- This is an example of **global** ambiguity.
  - There are different plausible tag sequences which can be assigned to the sentence
  - Often rare in real text. Why?
- Most words are only **locally** ambiguous. The intended PoS can be determined from the context

Time always flies like an arrow

Evaluating taggers

- Compare output of a tagger with a human-labelled gold standard
  - assume that performance will be similar on other similar unlabelled text
- Measure accuracy: proportion of tags which are correct
  - could measure (average) precision of each class
- On well-formed text, best methods have accuracy of 96-97%
  - using the Penn Treebank tagset
  - average of one error every couple of sentences
- Inter-annotator agreement is also around 97% accurate

Information sources for PoS tagging

What information can be used to determine the most likely PoS tag for a word token?

- Word identity (the likelihood of a tag given the word)
- Adjacent PoS tags (the likelihood of a sequence of tags)

Word identities

- A word may have different possible PoS tags
- But they are not all equally likely
- **Entropy** can be used to measure the uncertainty in the tag distribution
  - 50:50 is high entropy (high uncertainty)
  - 90:10 is low entropy (low uncertainty)
  - See the lab
- Tag distributions for words are often low entropy:
  - one tag is far more likely than the other possibilities

nltk.tag.UnigramTagger

A unigram Pos tagger

- Choose the most likely tag for each word

$$tag(w) = \underset{t}{\operatorname{argmax}} P(t|w)$$

- Use labelled training data to estimate these probabilities

$$P(t|w) = \frac{\text{number of occurrences of } w \text{ tagged as } t}{\text{number of occurrences of } w}$$

- Always chooses same tag for a word regardless of context
- Usually results in a tagger with about 90% accuracy

Beyond unigram PoS tagging

- Which of these look like a possible tag sequence in well-formed English?
  - DET DET JJR NN NNS VBD
  - DET NNS VBD DET JJR NN
  - VBD JJR NNS DET NN DET
- How can we incorporate information about likely tag sequences into a tagger?
  - Hidden Markov Models (HMMs)

Part 2: Hidden Markov Models

Consider this example

"Every night I dream the same dream."

- What PoS tag would you associate with each token in the sentence above?
- Which word type(s) are ambiguous with respect to PoS tag?
- How do you know the correct tag?

The PoS tagging problem

- Input: a sequence of words w_1^n:

$$w_1^n = (w_1, w_2, \dots, w_n)$$

- Output: a sequence of tags t_1^n:

$$t_1^n = (t_1, t_2, \dots, t_n)$$

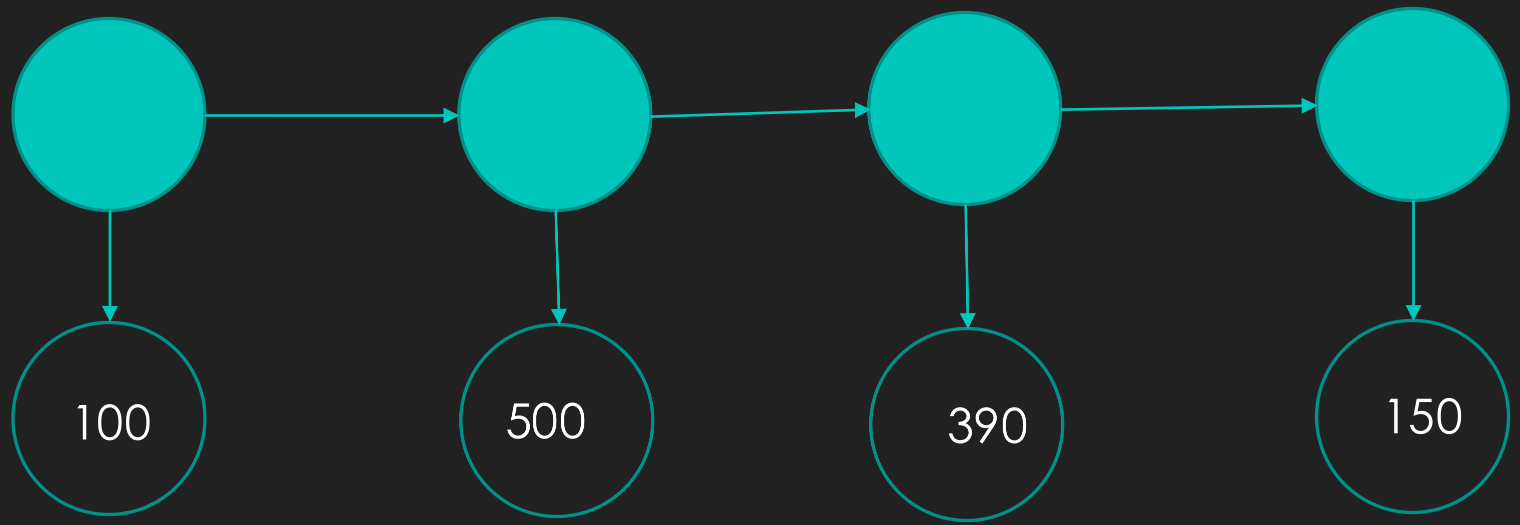
- In particular, find the most probable PoS tag sequence $\hat{t}_1^n$:

$$\hat{t}_1^n = \underset{t_1^n}{\operatorname{argmax}}\, P(t_1^n | w_1^n)$$

Hidden Markov Models (HMMs)

A sequence of **observations** is *generated* by a sequence of **hidden states.**

Can we infer the most likely sequence of hidden states from the observations



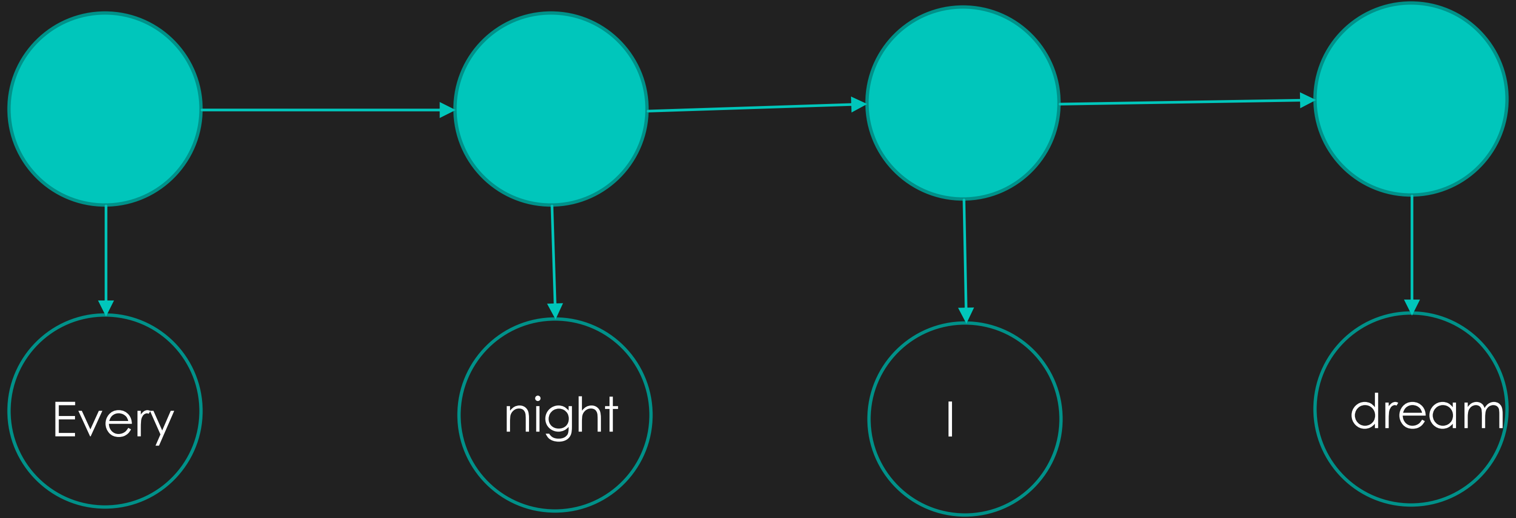
Markov Assumption: current state depends only on previous state

Output Assumption: current output depends only on current state

HMMs for PoS Tagging

POS tags are the hidden states

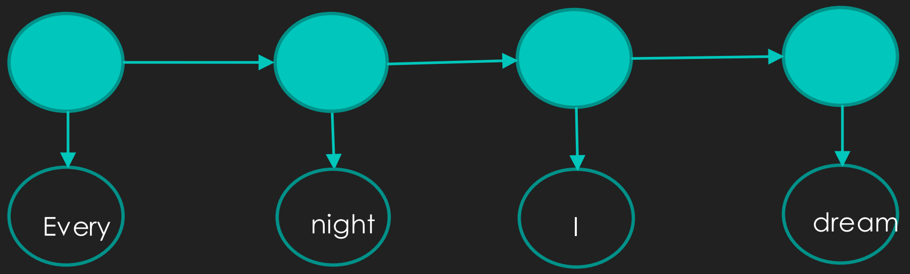
Words are the observations



Markov assumption: current POS tag depends only on single previous tag

Output assumption: word depends only on current POS tag

Encoding the assumptions probabilistically



Markov assumption: current POS tag depends only on single previous tag

$$P\left(t_i \middle| t_1^{i-1}\right) = P(t_i | t_{i-1})$$

Output assumption: word depends only on current POS tag

$$P(w_1^n | t_1^n) = \prod_i^n P(w_i | t_i)$$

Parameters for an HMM

To define a HMM tagger, we need to specify:

- **Emission** or **observation** probabilities:
  - $P(w|t)$ for each word *w* and tag *t*
- **Transition** or **bigram** probabilities:
  - $P(t_j|t_i)$ for each pair of tags t_i and t_j

These probabilities can be:

- calculated directly from POS-tagged corpora (supervised approach)
- learnt from untagged corpora (unsupervised approach) using Expectation Maximisation (EM)

Calculating emission probabilities

- For each possible tag, we need to count the number of occurrences of each word.

```
train
```

```
[('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'),
```

```python
def calculate_emissions(trainlist):
    #trainlist is a list of (word,tag) pairs
    emissions={}
    for word,tag in trainlist:
        current=emissions.get(tag,{})
        current[word]=current.get(word,0)+1
        emissions[tag]=current
    return {tag:{word:value/sum(worddist.values()) for word,value in worddist.items()}
            for tag,worddist in emissions.items()}
```

```
calculate_emissions(train)
```

```
{'NNP': {'Pierre': 6.613100552193897e-05,
  'Vinken': 2.204366850731299e-05,
  'Nov.': 0.0026231965523702454,
  'Mr.': 0.04412040251738694,
  'Elsevier': 1.1021834253656494e-05,
  'N.V.': 0.0001432838452975344,
  'Dutch': 8.817467402925195e-05,
```

Calculating transition probabilities

- For each possible tag, we need to count the number of occurrences of each previous tag.

```
train
```

```
[('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'),
```

```python
def calculate_transitions(trainlist):
    transitions={}
    previous="start"
    for _, tag in trainlist:
        current=transitions.get(previous,{})
        current[tag]=current.get(tag,0)+1
        transitions[tag]=current
        previous =tag
    return {previous:{tag:value/sum(tagdist.values()) for tag,value in tagdist.items()}
          for previous,tagdist in transitions.items()}
```

```
calculate_transitions(train)
```

```
{'NNP': {'NNP': 0.09662176841190528,
  ',': 0.05123470200593816,
  'CD': 0.03830408983058779,
  'NNS': 0.06321111977269726,
  'JJ': 0.06465412582586803,
  'MD': 0.01033533117787632l,
```

Forward Algorithm

- calculates the probability of a word sequence given a tag sequence

$$P(w_1^n|t_1^n) = \prod_{i}^{n} P(w_i|t_i)$$

remember the output assumption: the current word only depends on the current tag

w	P(w|N)	P(w|V)
flies	0.025	0.015
like	0.012	0.034
flowers	0.05	0.005

$$P(\text{flies like flowers }|N\ V\ N)$$

$$= P(\text{flies}|N).\,P(\text{like}|V).\,P(\text{flowers}|N)$$

$$= 0.025 \times 0.034 \times 0.05 = 0.0000425$$

Tagging as decoding

How do we use a HMM to find the most likely tag sequence?

$$\hat{t}_1^n = \underset{t_1^n}{\operatorname{argmax}} P(t_1^n | w_1^n)$$

Apply Bayes' Rule

$$\hat{t}_1^n = \underset{t_1^n}{\operatorname{argmax}} \frac{P(w_1^n | t_1^n) P(t_1^n)}{P(w_1^n)}$$

Drop the denominator

$$\hat{t}_1^n = \underset{t_1^n}{\operatorname{argmax}} P(w_1^n | t_1^n) P(t_1^n)$$

Simplifying assumptions

Assume output independence: current observation depends only on current state

$$\hat{t}_1^n = \underset{t_1^n}{\operatorname{argmax}} \prod_{i}^{n} P(w_i|t_i)\, P(t_1^n)$$

Make the bigram or first order Markov assumption: current state depends only on the single previous state

$$\hat{t}_1^n = \underset{t_1^n}{\operatorname{argmax}} \prod_{i}^{n} P(w_i|t_i)\, P(t_i|t_{i-1})$$

emission probabilities

transition probabilities

Decoding

- Given a tag sequence and a word sequence, we can estimate the probability that the word sequence was generated by that tag sequence

$$P(t_1^n|w_1^n) \propto \prod_i^n P(w_i|t_i)\, P(t_i|t_{i-1})$$

- So given two possible tag sequences we can choose between them

$$\hat{t}_1^n = \underset{t_1^n}{\operatorname{argmax}} \prod_i^n P(w_i|t_i)\, P(t_i|t_{i-1})$$

Decoding example

- Which is the most likely tag sequence for “flies like flowers”?
  - NVN
  - VNN

w	P(w | N)	P(w | V)
flies	0.025	0.015
like	0.012	0.034
flowers	0.05	0.005

	N	V	start
N	P(N | N) = 0.13	P(V | N)=0.43	P(start | N)=0
V	P(N | V)= 0.35	P(V | V)=0.05	P(start | V)=0
start	P(N | start)=0.29	P(V | start)=0.32	P(start | start)=0

P(N V N | flies like flowers)

w	P(w|N)	P(w|V)
flies	**0.025**	0.015
like	0.012	**0.034**
flowers	**0.05**	0.005

	N	V	start
N	P(N|N) = 0.13	**P(V|N)=0.43**	P(start|N)=0
V	**P(N|V)= 0.35**	P(V|V)=0.05	P(start|V)=0
start	**P(N|start)=0.29**	P(V|start)=0.32	P(start|start)=0

P(flies like flowers | N V N) = $P(flies|N) \times P(like|V) \times P(flowers|N)$

$= 0.025 \times 0.034 \times 0.05 = 0.0000425$

P(N V N) = $P(N|start) \times P(V|N) \times P(N|V) = 0.29 \times 0.43 \times 0.35 = 0.043645$

P(N V N | flies like flowers) = $0.0000425 \times 0.043645 = 0.0000018549125$

P(V N N | Flies like flowers)

w	P(w|N)	P(w|V)
flies	0.025	**0.015**
like	**0.012**	0.034
flowers	**0.05**	0.005

	N	V	start
N	**P(N|N) = 0.13**	P(V|N)=0.43	P(start|N)=0
V	**P(N|V)= 0.35**	P(V|V)=0.05	P(start|V)=0
start	P(N|start)=0.29	**P(V|start)=0.32**	P(start|start)=0

P(flies like flowers | V N N) $= 0.015 \times 0.012 \times 0.05 = 0.00009$

P(V N N) $= P(V|start) \times P(N|V) \times P(N|N) = 0.32 \times 0.35 \times 0.13 = 0.01456$

P(V N N | flies like flowers) $= 0.00009 \times 0.01456 = 0.0000013104$

So which is the most likely tag sequence given the word sequence?

Finding the most likely tag sequence

- Brute force?
- Number of possible tag sequences = k^n
  - where k = |tagset|, n = |word tokens|
- With a tagset of size 2 and a sentence of length 3 there are: **2^3 = 8** possible tag sequences
  - possible to try every one
- With a tagset of size 10 and a sentence of length 8 there are:
  - 10^8 = 100 000 000 possible tag sequences
  - it would take just over 1 day to check all possible tag sequence at the rate of 1000 per second
- With a tagset of size 45 and a sentence of length 15 ….

Viterbi Algorithm

- finds the best tag sequence without enumerating all possibilities
- exploits HMM assumptions
  - probability of next state only depends on current state
  - probability of current output only depends on current state
- classic example of dynamic programming
  - recursively decompose problem into smaller problems
  - keep track of (tabulate) solutions to sub-problems

Viterbi sub-problems

- For an input sequence w_1^n
- A sub-problem corresponds to a pair (i,t) where:
- ***i*** is the position in the sequence, i< n
- ***t*** is the current PoS tag

flies like flowers

i=1, store best path where
t_1=N
t_1=V

Viterbi sub-problems

- For an input sequence w_1^n
- A sub-problem corresponds to a pair (i,t) where:
- ***i*** is the position in the sequence, i< n
- ***t*** is the current PoS tag

flies like flowers

i=2, store best path where
t_2=N
t_2=V

Viterbi sub-problems

- For an input sequence w_1^n
- A sub-problem corresponds to a pair (i,t) where:
- ***i*** is the position in the sequence, i< n
- ***t*** is the current PoS tag

flies like flowers

i=3, store best path
t_3=N
t_3=V

Viterbi initialisation

flies like flowers

i=1, store best path where
t_1=N
t_1=V

Subproblem 1: t_1 = N

$$V(1,N) = P(t_1 = N) = P(flies|N) \times P(N|start)$$
$$= 0.025 \times 0.29 = 0.00725$$

Subproblem 2: t_1 = V

$$V(1,V) = P(t_1 V) = P(flies|V) \times P(V|start)$$
$$= 0.015 \times 0.32 = 0.0048$$

w	P(w|N)	P(w|V)
flies	0.025	0.015
like	0.012	0.034
flowers	0.05	0.005

	N	V	start
N	P(N|N) = 0.13	P(V|N)=0.43	P(start|N)=0
V	P(N|V)= 0.35	P(V|V)=0.05	P(start|V)=0
start	P(N|start)=0.29	P(V|start)=0.32	P(start|start)=0

Recursive Step

- For each subsequent position $i \epsilon \{2, \ldots, n\}$ and each tag, t

$$\boxed{V(i,t) = \max_{t' \epsilon T}(V(i-1,t') \times P(t|t') \times P(w_i|t))}$$

- For each possible previous tag t', what's the probability of the current tag being t?
- Which of these is highest? If the current tag is t then the previous tag must have been the one which gave the highest probability

Step 2

flies like flowers

i=2, store best path where
t_2=N
t_2=V

Subproblem 1:**V(2,N)**
The previous tag could have been N or V

- **If it was N**:

$$P(t_2 = N) = V(1, N) \times P(N|N) \times P(like|N)$$

$$P(t_2 = N) = 0.00725 \times 0.13 \times 0.012 = 0.00001131$$

- **If it was V**:

$$P(t_2 = N) = V(1, V) \times P(N|V) \times P(like|N)$$

$$P(t_2 = N) = 0.0048 \times 0.35 \times 0.012 = 0.00002016$$

- Which is higher?
- So V(2,N) = ??

w	P(w|N)	P(w|V)
flies	0.025	0.015
like	0.012	0.034
flowers	0.05	0.005

	N	V	start
N	P(N|N) = 0.13	P(V|N)=0.43	P(start|N)=0
V	P(N|V)= 0.35	P(V|V)=0.05	P(start|V)=0
start	P(N|start)=0.29	P(V|start)=0.32	P(start|start)=0

Step 2

flies like flowers

i=2, store best path where
t_2=N
t_2=V

Subproblem 2:**V(2,V)**
The previous tag could have been N or V

- **If it was N**:

$$P(t_2 = V) = V(1, N) \times P(V|N) \times P(like|V) \ = ??$$

- **If it was V**:

$$P(t_2 = V) = V(1, V) \times P(V|V) \times P(like|V) = ??$$

- Which is higher?
- So V(2,V) = ??

w	P(w|N)	P(w|V)
flies	0.025	0.015
like	0.012	0.034
flowers	0.05	0.005

	N	V	start
N	P(N|N) = 0.13	P(V|N)=0.43	P(start|N)=0
V	P(N|V)= 0.35	P(V|V)=0.05	P(start|V)=0
start	P(N|start)=0.29	P(V|start)=0.32	P(start|start)=0

Step 3

flies like flowers

i=3, store best path where
t_3=N
t_3=V

Subproblem 1:**V(3,N)**
The previous tag could have been N or V

- **If it was N:**

$$P(t_3 = N) = V(2, N) \times P(N|N) \times P(flowers|N) = ??$$

- **If it was V:**

$$P(t_3 = N) = V(2, V) \times P(N|V) \times P(flowers|N) = ??$$

- Which is higher?
- So V(3,N) = ??

w	P(w|N)	P(w|V)
flies	0.025	0.015
like	0.012	0.034
flowers	0.05	0.005

	N	V	start
N	P(N|N) = 0.13	P(V|N)=0.43	P(start|N)=0
V	P(N|V)= 0.35	P(V|V)=0.05	P(start|V)=0
start	P(N|start)=0.29	P(V|start)=0.32	P(start|start)=0

Step 3

flies like flowers

i=3, store best path where
t_3=N
t_3=V

Subproblem 2:**V(3,V)**
The previous tag could have been N or V

- **If it was N**:

$P(t_3 = V) = V(2, N) \times P(V|N) \times P(flowers|V) = ??$

- **If it was V**:

$P(t_3 = V) = V(2, V) \times P(V|V) \times P(flowers|V) = ??$

- Which is higher?
- So V(3,V) = ??

w	P(w|N)	P(w|V)
flies	0.025	0.015
like	0.012	0.034
flowers	0.05	0.005

	N	V	start
N	P(N|N) = 0.13	P(V|N)=0.43	P(start|N)=0
V	P(N|V)= 0.35	P(V|V)=0.05	P(start|V)=0
start	P(N|start)=0.29	P(V|start)=0.32	P(start|start)=0

Efficiency of Viterbi

- Number of sub-problems = $k \times n$
- In each sub-problem, we have to consider *k* previous sub-problems
- So complexity = $k^2 \times n$
- In toy example, worse than k^n
  - 12 > 8
- But with 10 tags and a sentence of length 8
  - 800 << 100000000
- With 45 tags and sentences of length 15
  - 30375 << ????

Next time

- Another sequence labelling problem
  - Named entity recognition

Making progress

- There is 1 associated notebook this week:

  - Part 1: Lab_8_1.ipynb