THE UNIVERSITY OF MELBOURNE

Comp90042 Natural Language Processing

Workshop (Week4 | 2025s1)

Dr Jean Lee

Agenda

1. Discussion (~35 mins)
   - **Part of Speech (POS) Tag**
   - **Hidden Markov Model (HMM) and Viterbi algorithm**

2. Programming (~25 mins)

25 Mar	4	workshop-04.pdf	05-pos-tagging.ipynb 06-hmm.ipynb english-bidirectional-distsim.tagger stanford-postagger-4.2.0.jar	

Discussion : Part of Speech (POS) Tag

1. What is a **POS tag**?

   (a) POS tag (by hand) the following sentence: `Pierre Vinken, 61 years old, will join the board as a nonexecutive director Nov. 29.` according to the Penn Treebank tags. (Note that some of the tags are somewhat obscure.)

   (b) What are some common approaches to POS tagging? What aspects of the data might allow us to predict POS tags systematically?

Part of Speech (POS) Tag

What is POS Tag?

- A pos tag, aka word classes, lexical categories, or morphological classes
- a **label assigned to word token** in a sentence which indicates some **grammatical properties** (primarily syntactic) of its function in the sentence.
- Token-level classification, Sequence labeling

Part of Speech (POS) Tagsets

Major English tagsets

- Brown
- **Penn Treebank**
- CLAWS/BNC
- “Universal”

Number	Tag	Description
1.	CC	Coordinating conjunction
2.	CD	Cardinal number
3.	DT	Determiner
4.	EX	Existential *there*
5.	FW	Foreign word
6.	IN	Preposition or subordinating conjunction
7.	JJ	Adjective
8.	JJR	Adjective, comparative
9.	JJS	Adjective, superlative
10.	LS	List item marker
11.	MD	Modal
12.	NN	Noun, singular or mass
13.	NNS	Noun, plural
14.	NNP	Proper noun, singular
15.	NNPS	Proper noun, plural
16.	PDT	Predeterminer
17.	POS	Possessive ending
18.	PRP	Personal pronoun
19.	PRP$	Possessive pronoun
20.	RB	Adverb
21.	RBR	Adverb, comparative
22.	RBS	Adverb, superlative
23.	RP	Particle
24.	SYM	Symbol
25.	TO	*to*
26.	UH	Interjection
27.	VB	Verb, base form
28.	VBD	Verb, past tense
29.	VBG	Verb, gerund or present participle
30.	VBN	Verb, past participle
31.	VBP	Verb, non-3rd person singular present
32.	VBZ	Verb, 3rd person singular present
33.	WDT	Wh-determiner
34.	WP	Wh-pronoun
35.	WP$	Possessive wh-pronoun
36.	WRB	Wh-adverb

https://www.ling.upenn.edu/courses/Fall_2003/ling001/penn_treebank_pos.html

POS Tag (by hand)

```
NNP   Pierre
NNP   Vinken
,     ,
CD    61
NNS   years
JJ    old
,     ,
MD    will
VB    join
DT    the
NN    board
IN    as
DT    a
JJ    nonexecutive
NN    director
NNP   Nov.
CD    29
.     .
```

Number	Tag	Description
1.	CC	Coordinating conjunction
2.	CD	Cardinal number
3.	DT	Determiner
4.	EX	Existential *there*
5.	FW	Foreign word
6.	IN	Preposition or subordinating conjunction
7.	JJ	Adjective
8.	JJR	Adjective, comparative
9.	JJS	Adjective, superlative
10.	LS	List item marker
11.	MD	Modal
12.	NN	Noun, singular or mass
13.	NNS	Noun, plural
14.	NNP	Proper noun, singular
15.	NNPS	Proper noun, plural
16.	PDT	Predeterminer
17.	POS	Possessive ending
18.	PRP	Personal pronoun
19.	PRP$	Possessive pronoun
20.	RB	Adverb
21.	RBR	Adverb, comparative
22.	RBS	Adverb, superlative
23.	RP	Particle
24.	SYM	Symbol
25.	TO	*to*
26.	UH	Interjection
27.	VB	Verb, base form
28.	VBD	Verb, past tense
29.	VBG	Verb, gerund or present participle
30.	VBN	Verb, past participle
31.	VBP	Verb, non-3rd person singular present
32.	VBZ	Verb, 3rd person singular present
33.	WDT	Wh-determiner
34.	WP	Wh-pronoun
35.	WP$	Possessive wh-pronoun
36.	WRB	Wh-adverb

Note that some of the tags are somewhat obscure!

POS Tag - 05-pos-tagging.ipynb

NNP	Pierre
NNP	Vinken
,	,
CD	61
NNS	years
JJ	old
,	,
MD	will
VB	join
DT	the
NN	board
IN	as
DT	a
JJ	nonexecutive
NN	director
NNP	Nov.
CD	29
.	.

```
1 treebank.tagged_sents()[0]
```

```
[('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'),
 ('.', '.')]
```

```
1 treebank.tagged_sents(tagset="universal")[0]
```

```
[('Pierre', 'NOUN'),
 ('Vinken', 'NOUN'),
 (',', '.'),
 ('61', 'NUM'),
 ('years', 'NOUN'),
 ('old', 'ADJ'),
 (',', '.'),
 ('will', 'VERB'),
 ('join', 'VERB'),
 ('the', 'DET'),
 ('board', 'NOUN'),
 ('as', 'ADP'),
 ('a', 'DET'),
 ('nonexecutive', 'ADJ'),
 ('director', 'NOUN'),
 ('Nov.', 'NOUN'),
 ('29', 'NUM'),
 ('.', '.')]
```

POS tagging

What are some common approaches to POS tagging?

- ***Unigram :*** Assign **most common POS tag** on each **token**

```
form
{'NN': 17, 'VB': 1}
```

- ***N-gram:*** Assign **most common POS tag** In the same sequence of ***n* tokens**
- ***Rule-based:*** Write rules that **disambiguate** unigram tags.
- ***Sequential:*** Learn a **Hidden Markov Model** in a tagged corpus
- ***Classifier-based:*** Treat as a **supervised** machine learning problem

POS tagging

What aspects of the data might allow us to predict POS tags systematically?

- ***Frequency :*** The most common tags of words can be identified from their frequency in the training corpus (unigram approach).
- ***Context:*** The context in which a word appears, including the tags of surrounding words (n-gram approach), helps in disambiguating POS tags.
- ***Morphology:*** Word forms, such as suffixes (e.g., "-ed" for past tense verbs), can inform rule-based and classifier-based tagging.
- ***Syntax and Semantics:*** The syntactic structure and the semantic relationships between words can be leveraged in rule-based and sequential tagging models to improve accuracy.

POS Tag - 05-pos-tagging.ipynb

```
1 unigram_tagger.ta
```

```
[('You', 'PRP'),
 ('better', 'JJR'),
 ('start', 'VB'),
 ('swimming', None),
 ('or', 'CC'),
 ('sink', 'VB'),
 ('like', 'IN'),
 ('a', 'DT'),
 ('stone', 'NN'),
 (',', ','),
 ('cause', 'NN'),
 ('the', 'DT'),
 ('times', 'NNS'),
 ('they', 'PRP'),
 ('are', 'VBP'),
 ('a', 'DT'),
 ('-', ':'),
 ('changing', 'VBG'),
 ('.', '.')]
```

```
1 bigram_tagger.tag(exa
```

```
[('You', 'PRP'),
 ('better', None),
 ('start', None),
 ('swimming', None),
 ('or', None),
 ('sink', None),
 ('like', None),
 ('a', None),
 ('stone', None),
 (',', None),
 ('cause', None),
 ('the', None),
 ('times', None),
 ('they', None),
 ('are', None),
 ('a', None),
 ('-', None),
 ('changing', None),
 ('.', None)]
```

```
1 bigram_tagger.tag(ex
```

```
[('You', 'PRP'),
 ('better', 'JJR'),
 ('start', 'VB'),
 ('swimming', 'NN'),
 ('or', 'CC'),
 ('sink', 'VB'),
 ('like', 'IN'),
 ('a', 'DT'),
 ('stone', 'NN'),
 (',', ','),
 ('cause', 'VB'),
 ('the', 'DT'),
 ('times', 'NNS'),
 ('they', 'PRP'),
 ('are', 'VBP'),
 ('a', 'DT'),
 ('-', ':'),
 ('changing', 'VBG'),
 ('.', '.')]
```

```
1 stanford_tagger.tag(example_sentence)
```

```
[('You', 'PRP'),
 ('better', 'RBR'),
 ('start', 'VB'),
 ('swimming', 'NN'),
 ('or', 'CC'),
 ('sink', 'NN'),
 ('like', 'IN'),
 ('a', 'DT'),
 ('stone', 'NN'),
 (',', ','),
 ('cause', 'VB'),
 ('the', 'DT'),
 ('times', 'NNS'),
 ('they', 'PRP'),
 ('are', 'VBP'),
 ('a', 'DT'),
 ('-', 'HYPH'),
 ('changing', 'NN'),
 ('.', '.')]
```

Hidden Markov Model (HMM)

2. What are the assumptions that go into a **Hidden Markov Model**? What is the time complexity of the **Viterbi algorithm**? Is this practical?

   (a) How can an HMM be used for POS tagging a text? For the purposes of POS tagging:

      i. How can the initial state probabilities π be estimated?

      ii. How can the transition probabilities A be estimated?

      iii. How can the emission probabilities B be estimated?

   (b) Estimate π, A and B for POS tagging, based on the following corpus:

```
1. silver-JJ wheels-NNS turn-VBP
2. wheels-NNS turn-VBP right-JJ
3. right-JJ wheels-NNS turn-VBP
```

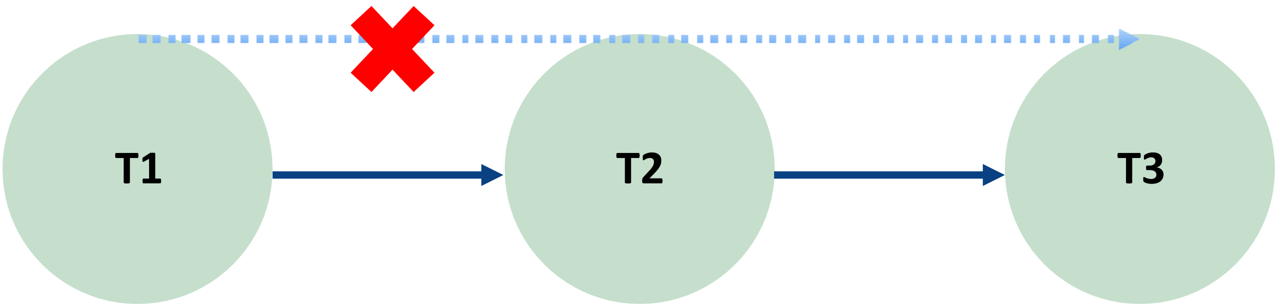
https://web.stanford.edu/~jurafsky/slp3/17.pdf

Hidden Markov Model (HMM)

What are the assumptions in HMM?

- **Markov assumption:** *the likelihood of transitioning into a given (next) state* depends only on the current state, and not the previous state(s) (or output(s))

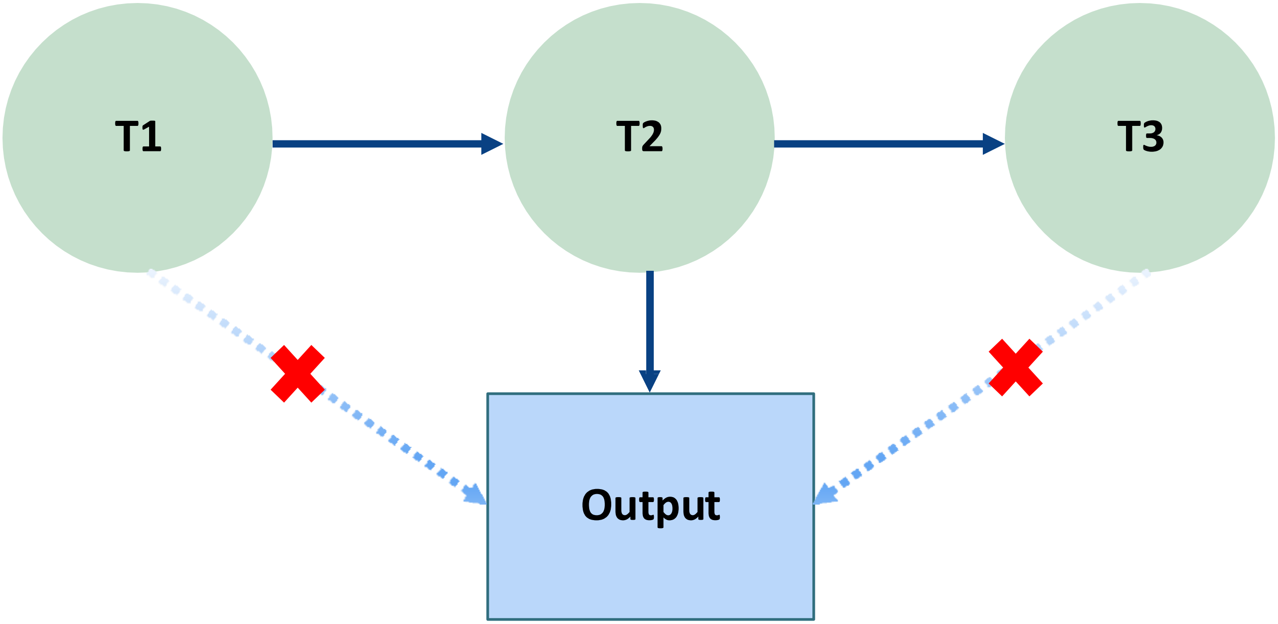
State 3 only depends on state 2, not sate 1



Hidden Markov Model (HMM)

What are the assumptions in HMM?

- **Output independence assumption:** *the likelihood of a state producing a certain word (as output)* does not depend on the preceding (or following) state(s) (or output(s)).



Hidden Markov Model (HMM)

How can an HMM be used for POS tagging a text? For the purposes of POS tagging (Sequence labeling)?

$$\hat{t} = argmax_t \, P(\boldsymbol{w} \mid \boldsymbol{t}) \, P(\boldsymbol{t})$$

- **Markov assumption:** (POS tags -> states of the HMM) ***Transition probabilities A (tag-tag pairs)***
  - The current state (tag) depends only on previous state (e.g) $P\ (NN \mid DT)$

$$P(\boldsymbol{t}) = \prod_{i=1}^{n} P(t_i \mid t_{i-1})$$ [Prob. of a tag depends only on the previous tag]

- **Output independence assumption:** (Tokens -> outputs) ***Emission probabilities B (word-tag pairs)***
  - An observed event (word) depends only on the hidden state (tag)

$$P(\boldsymbol{w} \mid \boldsymbol{t}) = \prod_{i=1}^{n} P(w_i \mid t_i)$$ [Prob. of a word depends only on the tag]

(e.g.) $P(like \mid VB) = \dfrac{count(VB,\ \ like)}{count(VB)}$

Hidden Markov Model (HMM)

What are the parameters do we need to learn when training HMM?

- Initial state probabilities π
  - record the distribution of tags for the **first token** of each sentence
- Transition probabilities A (tag-tag pairs)
  - record the distribution of tags of **the immediately following token**
- Emission probabilities B (word-tag pairs)
  - record the distribution of **corresponding tokens**

Hidden Markov Model (HMM)

Estimate π, A and B for POS tagging, based on the following corpus

```
1: silver-JJ  | wheels-NNS  | turn-VBP
2: wheels-NNS | turn-VBP    | right-JJ
3: right-JJ   | wheels-NNS  | turn-VBP
```

- Initial state probabilities π
  - record the distribution of tags for the **first token** of each sentence
  - In this case, 2 begin with **JJ** and 1 with **NNS**

$$\pi[JJ, NNS, VBP] = [\frac{2}{3}, \frac{1}{3}, 0]$$

Hidden Markov Model (HMM)

Estimate π, A and B for POS tagging, based on the following corpus

```
1: silver-JJ  | wheels-NNS | turn-VBP
2: wheels-NNS | turn-VBP   | right-JJ
3: right-JJ   | wheels-NNS | turn-VBP
```

- <u>Transition probabilities *A* (tag-tag pairs)</u>
  - record the distribution of tags of **the immediately following token**

A	JJ	NNS	VBP
(from) JJ	0	2/2	0
NNS	0	0	3/3
VBP	1/1	0	0

Hidden Markov Model (HMM)

Estimate π, A and B for POS tagging, based on the following corpus

```
1: silver-JJ  | wheels-NNS | turn-VBP
2: wheels-NNS | turn-VBP   | right-JJ
3: right-JJ   | wheels-NNS | turn-VBP
```

- **Emission probabilities B (word-tag pairs)**
  - record the distribution of **corresponding tokens**

B	right	silver	turn	wheels
(from) JJ	2/3	1/3	0	0
NNS	0	0	0	3/3
VBP	0	0	3/3	0

HMM and Viterbi algorithm

3. Consider using the following Hidden Markov Model to tag the sentence `silver wheels turn`:

   $\pi[\text{JJ}, \text{NNS}, \text{VBP}] = [0.3, 0.4, 0.3]$ Initial state probabilities π

Transition probabilities A

A	JJ	NNS	VBP
JJ	0.4	0.5	0.1
NNS	0.1	0.4	0.5
VBP	0.4	0.5	0.1

Emission probabilities B

B	silver	wheels	turn
JJ	0.8	0.1	0.1
NNS	0.3	0.4	0.3
VBP	0.1	0.3	0.6

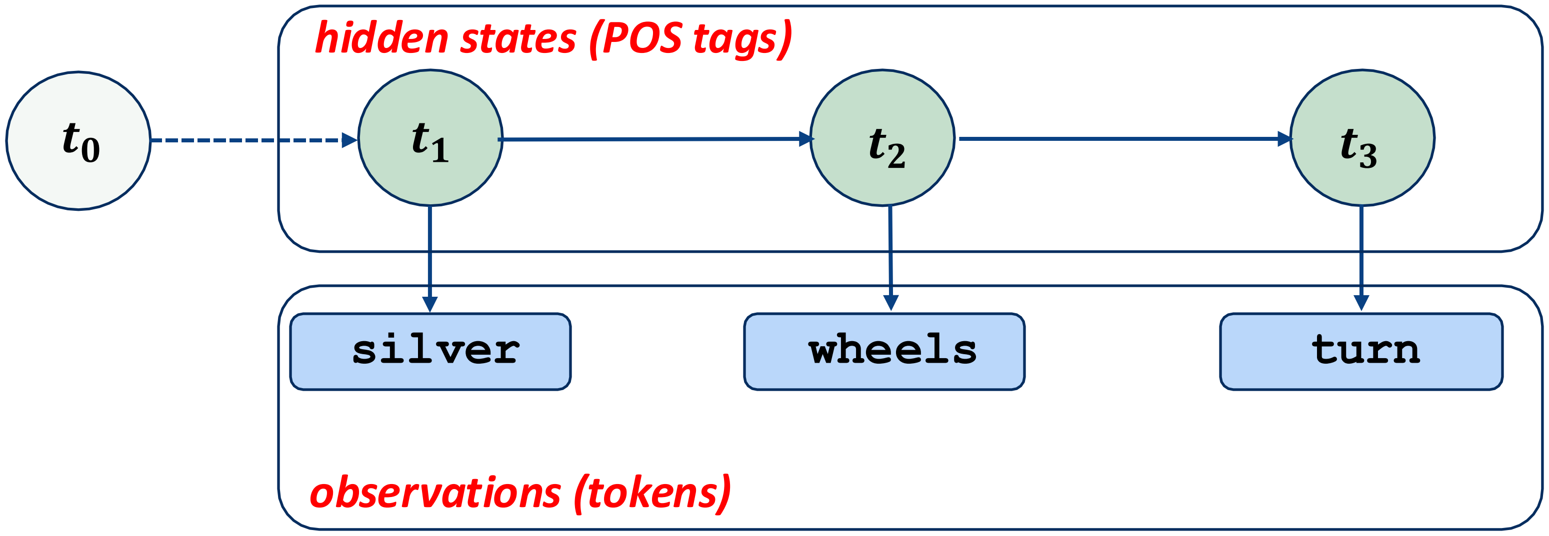
(a) Visualise the HMM as a graph.

(b) Use the **Viterbi algorithm** to find the most likely tag for this sequence.

HMM and Viterbi algorithm

Visualise the HMM as a graph

- Goal: ***predicting pos tags when given a sequence of word token***
- POS tags are the hidden states
- Word tokens are the corresponding outputs



HMM and Viterbi algorithm

What is Viterbi algorithm?

- Another **dynamic programming** algorithm
- Most **common decoding** algorithms for HMMs
- *Decoding : the task of determining which sequence of variables (tag) is the underlying source of some sequence of observations (token)*
- Goal : ***Compute the joint probability of the observation sequence together (token) with the best state (tag) sequence***

HMM and Viterbi algorithm

What is the time complexity of the Viterbi algorithm? Is this practical?

$O(T^2W)$, *where for an HMM with T possible states and a sentence of length W*

- In POS tagging,
  - Possible tags: 100
  - Tokens (in a typical sentence): 10-20
  - Time complexity: ~100,000 - 200,000
  - Yes, ***it's practical***

E18: Eisenstein, Jacob; Natural Language Processing, Chapter 7.3

HMM and Viterbi algorithm

Use the Viterbi algorithm to find the most likely tag for this sequence

$\pi[\text{JJ}, \text{NNS}, \text{VBP}] = [\boxed{0.3,} 0.4, 0.3]$

A	JJ	NNS	VBP	B	silver	wheels	turn
JJ	0.4	0.5	0.1	JJ	0.8	0.1	0.1
NNS	0.1	0.4	0.5	NNS	0.3	0.4	0.3
VBP	0.4	0.5	0.1	VBP	0.1	0.3	0.6

Initial state probabilities π * Emission probabilities B

α		1:*silver*	2:*wheels*	3:*turn*
JJ:	JJ	$\pi[\text{JJ}]B[\text{JJ}, \text{silver}]$ $\boxed{0.3 \times 0.8 = 0.24}$		
NNS:	NNS	$\pi[\text{NNS}]B[\text{NNS}, \text{silver}]$ $0.4 \times 0.3 = 0.12$		
VBP:	VBP	$\pi[\text{VBP}]B[\text{VBP}, \text{silver}]$ $0.3 \times 0.1 = 0.03$		

HMM and Viterbi algorithm

Use the Viterbi algorithm to find the most likely tag for this sequence

$\pi[\text{JJ}, \text{NNS}, \text{VBP}] = [0.3, 0.4, 0.3]$

A	JJ	NNS	VBP	B	silver	wheels	turn
JJ	0.4	0.5	0.1	JJ	0.8	0.1	0.1
NNS	0.1	0.4	0.5	NNS	0.3	0.4	0.3
VBP	0.4	0.5	0.1	VBP	0.1	0.3	0.6

Recursive step:

compute the probabilities of each tag for the token based on

Transition probabilities A

* Emission probabilities B

Then, choose tag which produce ***highest probability***

α	*1:silver*		*2:wheels*	*3:turn*
JJ:	0.24	JJ → JJ 0.24 NNS → JJ 0.12 VBP → JJ 0.03	$A[\text{JJ,JJ}]B[\text{JJ, wheels}]$ $\times 0.4 \times 0.1 = \mathbf{0.0096}$ $A[\text{NNS,JJ}]B[\text{JJ, wheels}]$ $\times 0.1 \times 0.1 = 0.0012$ $A[\text{VBP,JJ}]B[\text{JJ, wheels}]$ $\times 0.4 \times 0.1 = 0.0012$	
NNS:	0.12	JJ → NNS 0.24 NNS → NNS 0.12 VBP → NNS 0.03	$A[\text{JJ,NNS}]B[\text{NNS, wheels}]$ $\times 0.5 \times 0.4 = \mathbf{0.048}$ $A[\text{NNS,NNS}]B[\text{NNS, wheels}]$ $\times 0.4 \times 0.4 = 0.0192$ $A[\text{VBP,NNS}]B[\text{NNS, wheels}]$ $\times 0.5 \times 0.4 = 0.006$	
VBP:	0.03	JJ → VBP 0.24 NNS → VBP 0.12 VBP → VBP 0.03	$A[\text{JJ,VBP}]B[\text{VBP, wheels}]$ $\times 0.1 \times 0.3 = 0.0072$ $A[\text{NNS,VBP}]B[\text{VBP, wheels}]$ $\times 0.5 \times 0.3 = \mathbf{0.018}$ $A[\text{VBP,VBP}]B[\text{VBP, wheels}]$ $\times 0.1 \times 0.3 = 0.0009$	

HMM and Viterbi algorithm

Use the Viterbi algorithm to find the most likely tag for this sequence

$\pi[\text{JJ}, \text{NNS}, \text{VBP}] = [0.3, 0.4, 0.3]$

A	JJ	NNS	VBP	B	silver	wheels	turn
JJ	0.4	0.5	0.1	JJ	0.8	0.1	0.1
NNS	0.1	0.4	0.5	NNS	0.3	0.4	0.3
VBP	0.4	0.5	0.1	VBP	0.1	0.3	0.6

Recursive step:

compute the probabilities of each tag for the token based on

<u>Transition probabilities *A*</u>

<u>* Emission probabilities *B*</u>

*Then, choose tag which produce **highest probability***

α	*1:silver*	*2:wheels*		*3:turn*
JJ:	0.24	0.0096	JJ $\rightarrow$ JJ	$A[\text{JJ,JJ}]B[\text{JJ, turn}]$
		JJ $\rightarrow$ JJ	0.0096	$\times 0.4 \times 0.1 = 0.000384$
			NNS $\rightarrow$ JJ	$A[\text{NNS,JJ}]B[\text{JJ, turn}]$
			0.048	$\times 0.1 \times 0.1 = 0.00048$
			VBP $\rightarrow$ JJ	$A[\text{VBP,JJ}]B[\text{JJ, turn}]$
			0.018	$\times 0.4 \times 0.1 = \mathbf{0.00072}$
NNS:	0.12	0.048	JJ $\rightarrow$ NNS	$A[\text{JJ,NNS}]B[\text{NNS, turn}]$
		JJ $\rightarrow$ NNS	0.0096	$\times 0.5 \times 0.3 = 0.00144$
			NNS $\rightarrow$ NNS	$A[\text{NNS,NNS}]B[\text{NNS, turn}]$
			0.048	$\times 0.4 \times 0.3 = \mathbf{0.00576}$
			VBP $\rightarrow$ NNS	$A[\text{VBP,NNS}]B[\text{NNS, turn}]$
			0.018	$\times 0.5 \times 0.3 = 0.0027$
VBP:	0.03	0.018	JJ $\rightarrow$ VBP	$A[\text{JJ,VBP}]B[\text{VBP, turn}]$
		NNS $\rightarrow$ VBP	0.0096	$\times 0.1 \times 0.6 = 0.000576$
			NNS $\rightarrow$ VBP	$A[\text{NNS,VBP}]B[\text{VBP, turn}]$
			0.048	$\times 0.5 \times 0.6 = \mathbf{0.0144}$
			VBP $\rightarrow$ VBP	$A[\text{VBP,VBP}]B[\text{VBP, turn}]$
			0.018	$\times 0.1 \times 0.6 = 0.00108$

Programming!

Programming

1. In the iPython notebook `05-pos-tagging`:
   - Why does the bigram tagger — when used without "backoff" — perform worse than the unigram tagger? Find some examples of tokens which are tagged differently by the two models; give evidence from the training corpus as to why they are tagged differently.

2. In the iPython notebook `06-hmm`:
   - The Viterbi algorithm is implemented with loops. Try to implement Viterbi using recursion instead.
   - Can you see the difference between the speed of the Viterbi algorithm and the exhaustive search over the lattice? How much faster is Viterbi than exhaustive search on an example problem? (hint: *time* or *clock* functions from the *time* package can be useful)