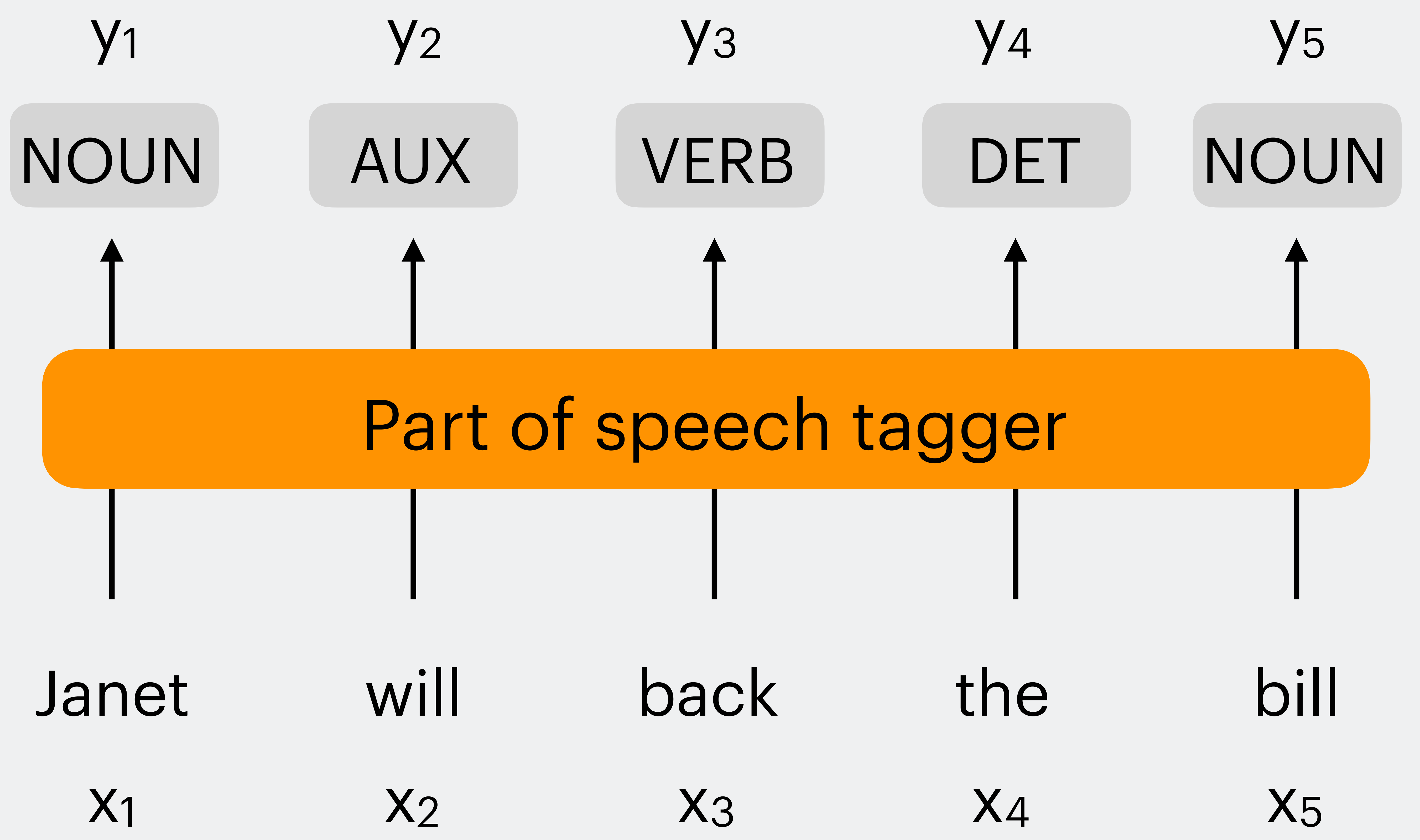
LG 467 Computers in Linguistics

[1-2021] Topic 5: POS tagging

Sakol Suethanapornkul

Previously...

POS tagging = assigning a part of speech to each word in a text



Adapted from: Jurafsky & Martin [chapter 8 PPT]

Previously...

Tag	Description	Example	Tag	Description	Example	Tag	Description	Example
CC	coord. conj.	*and, but, or*	NNP	proper noun, sing.	*IBM*	TO	"to"	*to*
CD	cardinal number	*one, two*	NNPS	proper noun, plu.	*Carolinas*	UH	interjection	*ah, oops*
DT	determiner	*a, the*	NNS	noun, plural	*llamas*	VB	verb base	*eat*
EX	existential 'there'	*there*	PDT	predeterminer	*all, both*	VBD	verb past tense	*ate*
FW	foreign word	*mea culpa*	POS	possessive ending	*'s*	VBG	verb gerund	*eating*
IN	preposition/ subordin-conj	*of, in, by*	PRP	personal pronoun	*I, you, he*	VBN	verb past partici-ple	*eaten*
JJ	adjective	*yellow*	PRP$	possess. pronoun	*your, one's*	VBP	verb non-3sg-pr	*eat*
JJR	comparative adj	*bigger*	RB	adverb	*quickly*	VBZ	verb 3sg pres	*eats*
JJS	superlative adj	*wildest*	RBR	comparative adv	*faster*	WDT	wh-determ.	*which, that*
LS	list item marker	*1, 2, One*	RBS	superlatv. adv	*fastest*	WP	wh-pronoun	*what, who*
MD	modal	*can, should*	RP	particle	*up, off*	WP$	wh-possess.	*whose*
NN	sing or mass noun	*llama*	SYM	symbol	*+,%, &*	WRB	wh-adverb	*how, where*

Source: Figure 8.2 in Jurafsky & Martin [chapter 8]

Previously...

An off-the-shelf tagger is available for English:

Code 7.1

```
from nltk import pos_tag, word_tokenize

text  = "John's big idea isn't all that bad."
token = word_tokenize(text)
pos   = pos_tag(token)

print(pos)
```

Question: What tagset is this?

POS tagging in English

Tag the following sentences with the PTB tags:

1. The/ quick/ brown/ fox/ jumps/ over/ the/ lazy/ dog/ ./

2. A/ woman/ needs/ a/ man/ like/ a/ fish/ needs/ a/ bicycle/ ./ *

NOTE: * The phrase is a famous feminist slogan coined by Irina Dunn

POS tagging in English

Question: Some things aren't right. What are they?

Code 8.1

```
from nltk import pos_tag, word_tokenize

txt1  = "The quick brown fox jumps over the lazy dog."
txt2  = "A woman needs a man like a fish needs a 
bicycle."

pos_tag(word_tokenize(txt1))
pos_tag(word_tokenize(txt2))
```

POS tagging in English

Question: Some things aren't right. What are they?

Code 8.1

```
pos_tag(word_tokenize(txt1))

[('The', 'DT'), ('quick', 'JJ'), ('brown', 'NN'), ('fox', 'NN'),
('jumps', 'VBZ'), ('over', 'IN'), ('the', 'DT'), ('lazy', 'JJ'),
('dog', 'NN'), ('.', '.')]

pos_tag(word_tokenize(txt2))
[('A', 'DT'), ('woman', 'NN'), ('needs', 'VBZ'), ('a', 'DT'),
('man', 'NN'), ('like', 'IN'), ('a', 'DT'), ('fish', 'JJ'),
('needs', 'VBZ'), ('a', 'DT'), ('bicycle', 'NN'), ('.', '.')]
```

POS tagging in English

Roughly 85% of word types aren't ambiguous

- *Janet* is always **NNP**, *hesitantly* is always **RB**

			WSJ		Brown	
Types:						
	Unambiguous	(1 tag)	44,432	(**86%**)	45,799	(**85%**)
	Ambiguous	(2+ tags)	7,025	(**14%**)	8,050	(**15%**)
Tokens:						
	Unambiguous	(1 tag)	577,421	(**45%**)	384,349	(**33%**)
	Ambiguous	(2+ tags)	711,780	(**55%**)	786,646	(**67%**)

Source: Figure 8.4 in Jurafsky & Martin [chapter 8]

POS tagging in English

But those 15% ambiguous words tend to be common words

- ~60% of word tokens are ambiguous
- For instance, take the word *back*
  - earnings growth took a **back/** seat
  - a small building in the **back/**
  - a clear majority of senators **back/** the bill
  - enable the country to buy **back/** debt
  - I was twenty-one **back/** then

Adapted from: Jurafsky & Martin [chapter 8 PPT]

Sources of information for POS tagging

Let's use a more extreme example:

Code 8.2

```
pos_tag(word_tokenize("A man needs a woman like a fish
needs water."))

#[('A', 'DT'), ('man', 'NN'), ('needs', 'VBZ'),
# (‘a', 'DT'), ('woman', 'NN'), ('like', 'IN'),
# (‘a', 'DT'), ('fish', 'JJ'), ('needs', ‘NNS'),
# ('water', 'NN'), ('.', '.')]
```

Question: Which words are mis-tagged?

Sources of information for POS tagging

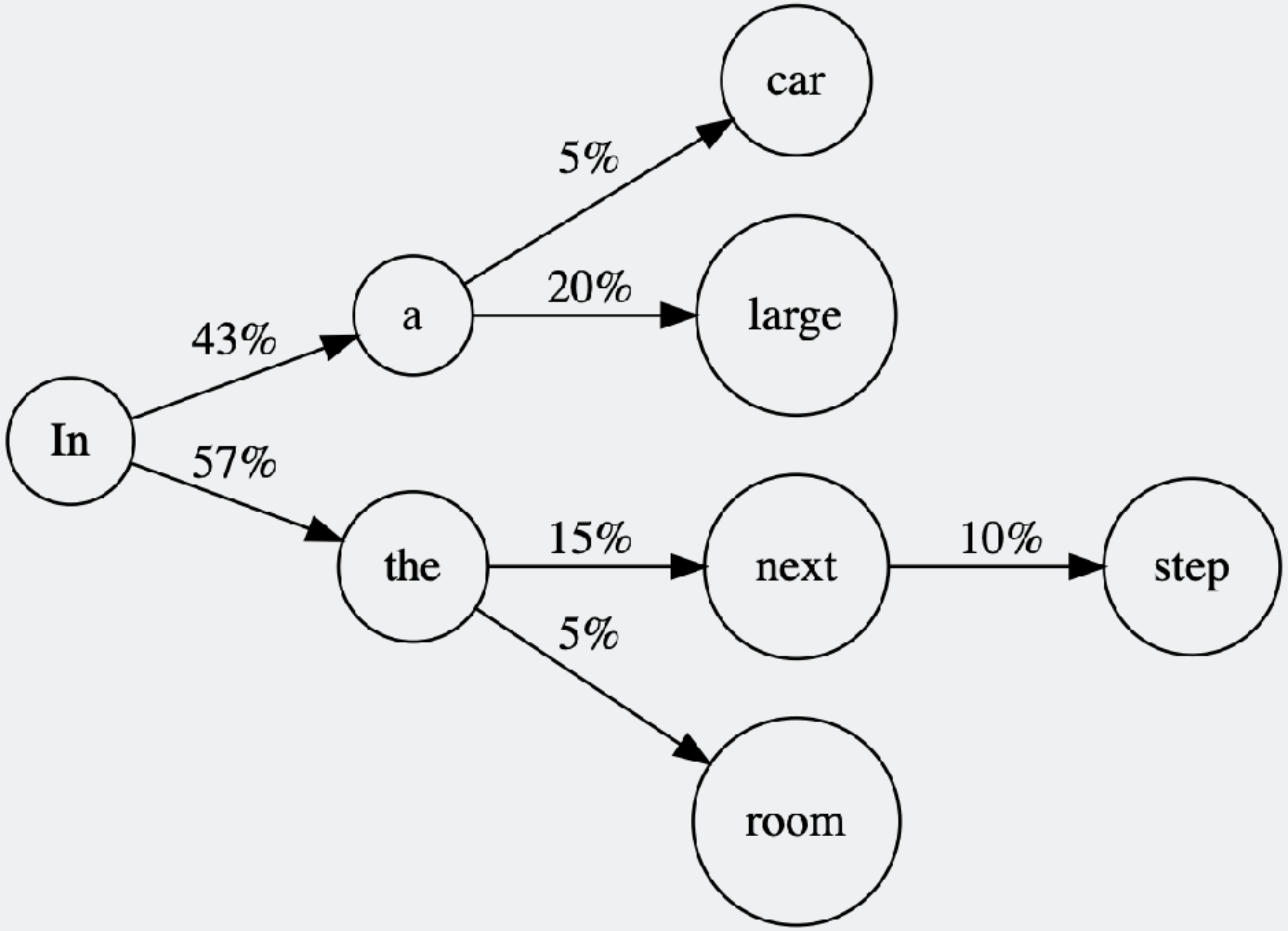
It seems like the following is probably true in NLTK's training data:

- prior probabilities of words/tags
  - *brown* is usually **NN**, i.e., $p(NN) > p(JJ)$
- conditional probabilities of sequences
  - **JJ** usually follows **IN+DT** (e.g., he’s *in*/**IN** the *next*/**JJ** room)
    - $p(JJ|IN, DT) > p(NN|IN, DT)$
- (morphology and wordshape [prefix, suffix, capitalization])

Language models as FSAs

We can model a sequence using a **weighted** bigram automaton

- Longer contexts possible as "complex" states
- Each transition depends on previous state



Hidden Markov Models (HMMs)

But this weighted bigram automaton is for words. How about hidden categories like POS?

Suppose we want to predict p(NN|JJ)

- Markov assumption probability of NN at this point depends on previous word being JJ
  - But typically, we have: *the large brown fox*….
  - We don't actually know for sure if '*brown*' is **JJ**

Hidden Markov Models (HMMs)

We need to:

- estimate likelihood of chain: ***DT JJ NN NN….***?
- Do so for every conceivable chain
- Find most likely one….without running out of memory!

HMM is in fact a **weighted FSA**

Hidden Markov Models (HMMs)

The HMM definition comprises:

- $V = v_1 \ldots . v_V$ # input vocabulary items
- $Q = q_1, \ldots q_N$ (q_0, q_F) # states
- $A = a_{11}, a_{12}, \ldots a_{n1} \ldots a_{nn}$ # transition prob. matrix
- $O = \langle o_1, \ldots . o_T \rangle$ # ordered observations of V
- $B = b_i(o_t)$ # prob. of o_t given q_i

Hidden Markov Models (HMMs)

The POS tagging task maps directly to the HMM definition:

- V: words of the English language
- Q: the parts of speech (state: DT, state: NN, etc.)
- A: the probability of NN given DT
- O: the text to be tagged $<w_1, \ldots w_n>$
- B: the probability of *the* given *DT*, i.e., p(the|DT)

Hidden Markov Models (HMMs)

Transition probabilities (A):

	NNP	MD	VB	JJ	NN	RB	DT
<*s*>	0.2767	0.0006	0.0031	0.0453	0.0449	0.0510	0.2026
NNP	0.3777	0.0110	0.0009	0.0084	0.0584	0.0090	0.0025
MD	0.0008	0.0002	$\boxed{0.7968}$	0.0005	0.0008	0.1698	0.0041
VB	0.0322	0.0005	0.0050	0.0837	0.0615	0.0514	0.2231
JJ	0.0366	0.0004	0.0001	0.0733	0.4509	0.0036	0.0036
NN	0.0096	0.0176	0.0014	0.0086	0.1216	0.0177	0.0068
RB	0.0068	0.0102	0.1011	0.1012	0.0120	0.0728	0.0479
DT	0.1147	0.0021	0.0002	0.2157	0.4744	0.0102	0.0017

p(VB|MD) = 0.7968 (rows give the condition)

Source: Figure 8.12 in Jurafsky & Martin [chapter 8]

Hidden Markov Models (HMMs)

Emission probabilities (B):

	Janet	will	back	the	bill
NNP	0.000032	0	0	0.000048	0
MD	0	0.308431	0	0	0
VB	0	0.000028	0.000672	0	0.000028
JJ	0	0	0.000340	0	0
NN	0	0.000200	0.000223	0	0.002337
RB	0	0	0.010446	0	0
DT	0	0	0	0.506099	0

p(will|MD) = 0.31 (assumming this is MD, chance to get 'will')

Source: Figure 8.12 in Jurafsky & Martin [chapter 8]

Standard algorithms for POS tagging

- Supervised Machine Learning Algorithms:
  - Hidden Markov Models
  - Conditional Random Fields (CRF)/ Maximum Entropy Markov Models (MEMM)
  - Neural sequence models (RNNs or Transformers)
  - Large Language Models (like BERT)
- All required a hand-labeled training set, equal performance (97% on English)
- All make use of information sources we discussed

Adapted from: Jurafsky & Martin [chapter 8 PPT]

SpaCy

SpaCy: Introduction

NLTK is extremely good for teaching and research

- Lots of different algorithms for different purposes

SpaCy is designed for application and production

- Text is fed through an NLP pipeline
- What comes out is different components of NLP processes

SpaCy: Installation

In Terminal (Mac):

```
[NAME]@[NAME] ~ % conda install -c conda-forge spacy
[NAME]@[NAME] ~ % python -m spacy download en_core_web_sm
```

In Anaconda Prompt (Windows):

```
c:\Users\[NAME] conda install -c conda-forge spacy
c:\Users\[NAME] python -m spacy download en_core_web_sm
```

First steps in SpaCy

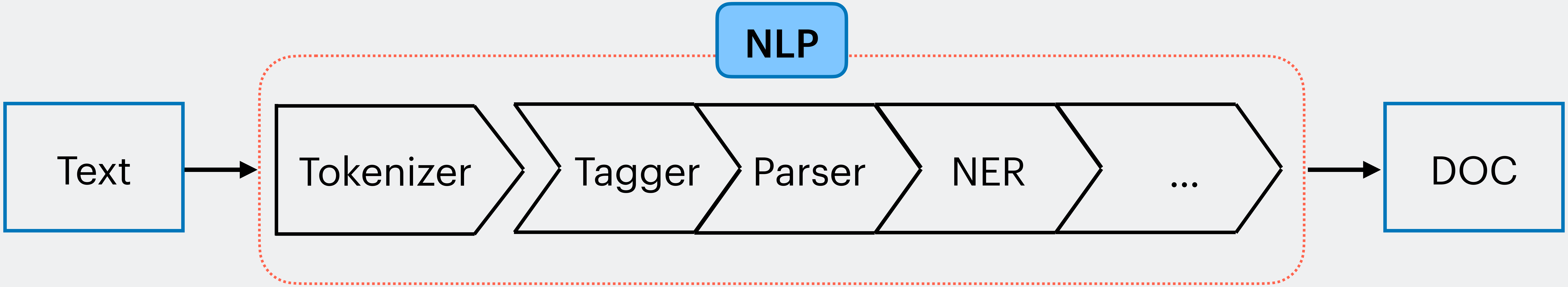
In English, there are four pre-trained pipeline models

- `en_core_web_sm` [small model, 13 MB]
- `en_core_web_md` [medium sized model, 44 MB]
- `en_core_web_lg` [large model, 742 MB]
- `en_core_web_trf` [Transformer based model, 438 MB]

NOTE: SpaCy provides data sources each model was trained on on its website

SpaCy: Introduction

A text is first tokenized before being processed through a pipeline



Adapted from: Spacy's website

First steps in SpaCy

These are the first few steps you must do:

Code 8.3

```
# #1 Import SpaCy
import spacy

# #2 Load the English model into nlp object
nlp = spacy.load(“en_core_web_sm”)

# #3 Process a text
doc = nlp("This is an example sentence.”)

# Swap #3 with text file
with open('ABC.txt') as f:
    txt = f.read()

doc = nlp(txt)
```

First steps in SpaCy

Now that we have a Document (Doc) object, what's next?

Name	Description	Creates
tagger	Part-of-speech tagger	Token.tag, Token.pos
parser	Dependency parser	Token.dep, Token.head, Doc.sents, Doc.noun_chunks
ner	Named entity recognizer	Doc.ents, Token.ent_iob, Token.ent_type

First steps in SpaCy

Now that we have a Document (Doc) object, what's next?

Code 8.4

```
# Print indices, tokens, and tags
[tok.i for tok in doc]
[tok.text for tok in doc]
[tok.lemma_ for tok in doc]
[tok.pos_ for tok in doc]
[tok.tag_ for tok in doc]

for tok in doc:
    print(tok.i, tok.text, tok.pos_, tok_tag_)

# If you need help
spacy.explain("DET")
spacy.explain("JJ")
```

Writing your own `FreqDist`

Previously, we relied on NLTK's `FreqDist()` to get frequency counts. It's time for our own version!

Code 8.5

```
from collections import defaultdict

# Create a dict; use default value for unknown key
pos_ct = defaultdict(int)

# Let's check:
print(pos_ct["DET"])
```

Writing your own `FreqDist`

Previously, we relied on NLTK's `FreqDist()` to get frequency counts. It's time for our own version!

```
for pos in [tok.pos_ for tok in doc]:
    pos_ct[pos] += 1

# To select tags and counts
[(t, c) for (t, c) in pos_ct.items()]

for t, c in pos_ct.items():
    print(t, "\t", c)

# You can use .items(), .keys(), .values()
```

Code 8.5
[Continue]

Our plan next week...

- Parsing, Context-Free Grammar (CFG), and Treebank
- Readings
  - J & M 3rd edition, Chapter 12
  - NLTK 7.4.2 Tree