Part of Speech Tagging

CL1: Jordan Boyd-Graber

University of Maryland

September 30, 2013

COLLEGE OF INFORMATION STUDIES

Adapted from slides by Ray Mooney

Roadmap

- The part of speech task
- Hidden Markov Models (high level)
- Hidden Markov Model (rigorous definition)
- Estimating HMM
- Tagging with HMM
- Examples with NLTK

Outline

1. What is POS Tagging and why do we care?
2. HMM Intuition
3. HMM Recapitulation
4. HMM Estimation
5. NLTK Taggers

POS Tagging: Task Definition

- Annotate each word in a sentence with a part-of-speech marker.
- Lowest level of syntactic analysis.

John	saw	the	saw	and	decided	to	take	it	to	the	table
NNP	VBD	DT	NN	CC	VBD	TO	VB	PRP	IN	DT	NN

- Useful for subsequent syntactic parsing and word sense disambiguation.

What are POS Tags?

- Original Brown corpus used a large set of 87 POS tags.
- Most common in NLP today is the Penn Treebank set of 45 tags. Tagset used in these slides for "real" examples. Reduced from the Brown set for use in the context of a parsed corpus (i.e. treebank).
- The C5 tagset used for the British National Corpus (BNC) has 61 tags.

Tag Examples

- Noun (person, place or thing)
  - Singular (NN): dog, fork
  - Plural (NNS): dogs, forks
  - Proper (NNP, NNPS): John, Springfields
- Personal pronoun (PRP): I, you, he, she, it
- Wh-pronoun (WP): who, what
- Verb (actions and processes)
  - Base, infinitive (VB): eat
  - Past tense (VBD): ate
  - Gerund (VBG): eating
  - Past participle (VBN): eaten
  - Non 3rd person singular present tense (VBP): eat
  - 3rd person singular present tense: (VBZ): eats
  - Modal (MD): should, can
  - To (TO): to (to eat)

Tag Examples (cont.)

- Adjective (modify nouns)
  - Basic (JJ): red, tall
  - Comparative (JJR): redder, taller
  - Superlative (JJS): reddest, tallest
- Adverb (modify verbs)
  - Basic (RB): quickly
  - Comparative (RBR): quicker
  - Superlative (RBS): quickest
- Preposition (IN): on, in, by, to, with
- Determiner:
  - Basic (DT) a, an, the
  - WH-determiner (WDT): which, that
- Coordinating Conjunction (CC): and, but, or,
- Particle (RP): off (took off), up (put up)

Open vs. Closed Class

- Closed class categories are composed of a small, fixed set of grammatical function words for a given language.
  - Pronouns, Prepositions, Modals, Determiners, Particles, Conjunctions
- Open class categories have large number of words and new ones are easily invented.
  - Nouns (Googler, textlish), Verbs (Google), Adjectives (geeky), Abverb (chompingly)

Ambiguity

"Like" can be a verb or a preposition

- I like/VBP candy.
- Time flies like/IN an arrow.

Around can be a preposition, particle, or adverb

- I bought it at the shop around/IN the corner.
- I never got around/RP to getting a car.
- A new Prius costs around/RB $25K.

How hard is it?

- Usually assume a separate initial tokenization process that separates and/or disambiguates punctuation, including detecting sentence boundaries.
- Degree of ambiguity in English (based on Brown corpus)
  - 11.5% of word types are ambiguous.
  - 40% of word tokens are ambiguous.
- Average POS tagging disagreement amongst expert human judges for the Penn treebank was 3.5%
- Based on correcting the output of an initial automated tagger, which was deemed to be more accurate than tagging from scratch.
- Baseline: Picking the most frequent tag for each specific word type gives about 90% accuracy 93.7% if use model for unknown words for Penn Treebank tagset.

Approaches

- Rule-Based: Human crafted rules based on lexical and other linguistic knowledge.
- Learning-Based: Trained on human annotated corpora like the Penn Treebank.
  - Statistical models: Hidden Markov Model (HMM), Maximum Entropy Markov Model (MEMM), Conditional Random Field (CRF)
  - Rule learning: Transformation Based Learning (TBL)
- Generally, learning-based approaches have been found to be more effective overall, taking into account the total amount of human expertise and effort involved.

Approaches

- Rule-Based: Human crafted rules based on lexical and other linguistic knowledge.
- Learning-Based: Trained on human annotated corpora like the Penn Treebank.
    - Statistical models: Hidden Markov Model (HMM), Maximum Entropy Markov Model (MEMM), Conditional Random Field (CRF)
    - Rule learning: Transformation Based Learning (TBL)
- Generally, learning-based approaches have been found to be more effective overall, taking into account the total amount of human expertise and effort involved.

Outline

1. What is POS Tagging and why do we care?
2. HMM Intuition
3. HMM Recapitulation
4. HMM Estimation
5. NLTK Taggers

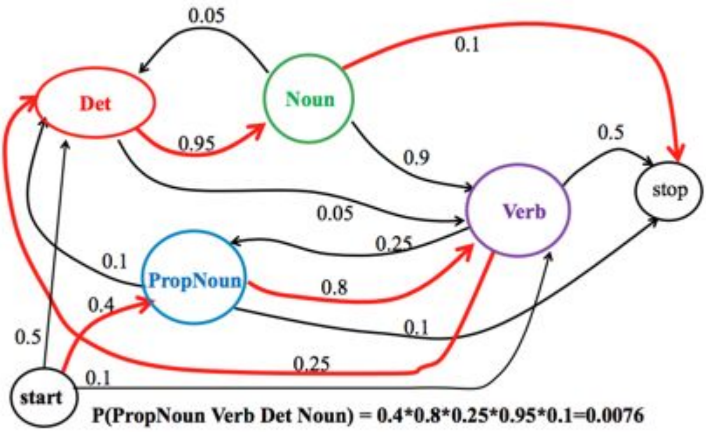
HMM Definition

- A finite state machine with probabilistic state transitions.
- Makes Markov assumption that next state only depends on the current state and independent of previous history.

Generative Model

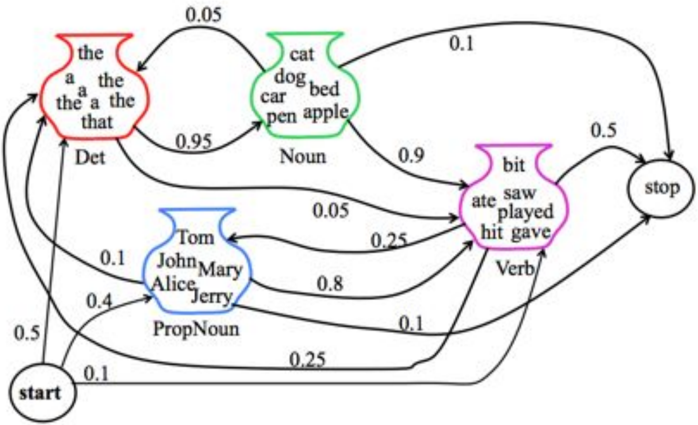
- Probabilistic generative model for sequences.
- Assume an underlying set of hidden (unobserved) states in which the model can be (e.g. parts of speech).
- Assume probabilistic transitions between states over time (e.g. transition from POS to another POS as sequence is generated).
- Assume a probabilistic generation of tokens from states (e.g. words generated for each POS).

Cartoon



0.05
0.1
Det
Noun
0.95
0.9
0.5
stop
Verb
0.05
0.25
0.1
PropNoun
0.8
0.4
0.1
0.5
0.25
0.1
start
P(PropNoun Verb Det Noun) = 0.4*0.8*0.25*0.95*0.1=0.0076

Cartoon



0.05
0.1
the
a
the
a
the
a
the
that
Det
cat
dog
bed
car
pen
apple
Noun
0.95
0.9
0.5
bit
ate
saw
played
hit
gave
Verb
stop
0.05
0.25
Tom
John
Mary
Alice
Jerry
PropNoun
0.1
0.8
0.4
0.1
0.5
0.25
0.1
start

Outline

1. What is POS Tagging and why do we care?
2. HMM Intuition
3. HMM Recapitulation
4. HMM Estimation
5. NLTK Taggers

HMM Definition

Assume K parts of speech, a lexicon size of V, a series of observations $\{x_1, \dots, x_N\}$, and a series of unobserved states $\{z_1, \dots, z_N\}$.

- π A distribution over start states (vector of length K): $\pi_i = p(z_1 = i)$
- θ Transition matrix (matrix of size K by K): $\theta_{i,j} = p(z_n = j | z_{n-1} = i)$
- β An emission matrix (matrix of size K by V): $\beta_{j,w} = p(x_n = w | z_n = j)$

HMM Definition

Assume K parts of speech, a lexicon size of V, a series of observations $\{x_1, \ldots, x_N\}$, and a series of unobserved states $\{z_1, \ldots, z_N\}$.

- π A distribution over start states (vector of length K): $\pi_i = p(z_1 = i)$
- θ Transition matrix (matrix of size K by K): $\theta_{i,j} = p(z_n = j | z_{n-1} = i)$
- β An emission matrix (matrix of size K by V): $\beta_{j,w} = p(x_n = w | z_n = j)$

Two problems: How do we move from data to a model? (Estimation) How do we move from a model and unlabled data to labeled data? (Inference)

Outline

1 What is POS Tagging and why do we care?

2 HMM Intuition

3 HMM Recapitulation

4 HMM Estimation

5 NLTK Taggers

Reminder: How do we estimate a probability?

- For a multinomial distribution (i.e. a discrete distribution, like over words):

$$\theta_i = \frac{n_i + \alpha_i}{\sum_k n_k + \alpha_k} \tag{1}$$

- α_i is called a smoothing factor, a pseudocount, etc.

Reminder: How do we estimate a probability?

- For a multinomial distribution (i.e. a discrete distribution, like over words):

$$\theta_i = \frac{n_i + \alpha_i}{\sum_k n_k + \alpha_k} \tag{1}$$

- α_i is called a smoothing factor, a pseudocount, etc.
- When $\alpha_i = 1$ for all i, it's called "Laplace smoothing" and corresponds to a uniform prior over all multinomial distributions.

Training Sentences

here	come	old	flattop
MOD	V	MOD	N

a	crowd	of	people	stopped	and	stared
DET	N	PREP	N	V	CONJ	V

gotta	get	you	into	my	life
V	V	PRO	PREP	PRO	V

and	I	love	her
CONJ	PRO	V	PRO

Training Sentences

x	here	come	old	flattop
	MOD	V	MOD	N

a	crowd	of	people	stopped	and	stared
DET	N	PREP	N	V	CONJ	V

gotta	get	you	into	my	life
V	V	PRO	PREP	PRO	V

and	I	love	her
CONJ	PRO	V	PRO

Training Sentences

x	here	come	old	flattop
z	MOD	V	MOD	N

a	crowd	of	people	stopped	and	stared
DET	N	PREP	N	V	CONJ	V

gotta	get	you	into	my	life
V	V	PRO	PREP	PRO	V

and	I	love	her
CONJ	PRO	V	PRO

Initial Probability π

POS	Frequency	Probability
MOD	1.1	0.234
DET	1.1	0.234
CONJ	1.1	0.234
N	0.1	0.021
PREP	0.1	0.021
PRO	0.1	0.021
V	1.1	0.234

Remember, we're taking MAP estimates, so we add 0.1 (arbitrarily chosen) to each of the counts before normalizing to create a probability distribution. This is easy; one sentence starts with an adjective, one with a determiner, one with a verb, and one with a conjunction.

Training Sentences

here	come	old	flattop
MOD	V	MOD	N

a	crowd	of	people	stopped	and	stared
N	PREP	N	V	CONJ	V	

gotta	get	you	into	my	life
V	V	PRO	PREP	PRO	N

and	I	love	her
CONJ	PRO	V	PRO

Training Sentences

here	come	old	flattop
MOD	V	MOD	N

a	crowd	of	people	stopped	and	stared
N	PREP	N	V	CONJ	V	

gotta	get	you	into	my	life
V	V	PRO	PREP	PRO	N

and	I	love	her
CONJ	PRO	V	PRO

Training Sentences

here	come	old	flattop
MOD	V	MOD	N

a	crowd	of	people	stopped	and	stared
N	PREP	N	V	CONJ	V	

gotta	get	you	into	my	life
V	V	PRO	PREP	PRO	N

and	I	love	her
CONJ	PRO	V	PRO

Transition Probability θ

- We can ignore the words; just look at the parts of speech. Let's compute one row, the row for verbs.
- We see the following transitions: V $\rightarrow$ MOD, V $\rightarrow$ CONJ, V $\rightarrow$ V, V $\rightarrow$ PRO, and V $\rightarrow$ PRO

POS	Frequency	Probability
MOD	1.1	0.193
DET	0.1	0.018
CONJ	1.1	0.193
N	0.1	0.018
PREP	0.1	0.018
PRO	2.1	0.368
V	1.1	0.193

- And do the same for each part of speech ...

Training Sentences

here	come	old	flattop
MOD	V	MOD	N

a	crowd	of	people	stopped	and	stared
N	PREP	N	V	CONJ	V	

gotta	get	you	into	my	life
V	V	PRO	PREP	PRO	N

and	I	love	her
CONJ	PRO	V	PRO

Training Sentences

here	come	old	flattop
MOD	V	MOD	N

a	crowd	of	people	stopped	and	stared
N	PREP	N	V	CONJ	V	

gotta	get	you	into	my	life
V	V	PRO	PREP	PRO	N

and	I	love	her
CONJ	PRO	V	PRO

Emission Probability β

Let's look at verbs . . .

Word	a	and	come	crowd	flattop
Frequency	0.1	0.1	1.1	0.1	0.1
Probability	0.0125	0.0125	0.1375	0.0125	0.0125

Word	get	gotta	her	here	i
Frequency	1.1	1.1	0.1	0.1	0.1
Probability	0.1375	0.1375	0.0125	0.0125	0.0125

Word	into	it	life	love	my
Frequency	0.1	0.1	0.1	1.1	0.1
Probability	0.0125	0.0125	0.0125	0.1375	0.0125

Word	of	old	people	stared	stopped
Frequency	0.1	0.1	0.1	1.1	1.1
Probability	0.0125	0.0125	0.0125	0.1375	0.1375

Viterbi Algorithm

- Given an unobserved sequence of length L, $\{x_1, \ldots, x_L\}$, we want to find a sequence $\{z_1 \ldots z_L\}$ with the highest probability.

Viterbi Algorithm

- Given an unobserved sequence of length L, $\{x_1, \dots, x_L\}$, we want to find a sequence $\{z_1 \dots z_L\}$ with the highest probability.
- It's impossible to compute K^L possibilities.
- So, we use dynamic programming to compute best sequence for each subsequence from 0 to t that ends in state k.
- Memoization: fill a table of solutions of sub-problems
- Solve larger problems by composing sub-solutions
- Base case:

$$\delta_1(k) = \pi_k \beta_{k,x_i} \tag{2}$$

- Recursion:

$$\delta_n(k) = \max_j \left(\delta_{n-1}(j)\theta_{j,k}\right)\beta_{k,x_n} \tag{3}$$

- The complexity of this is now K^2L.
- In class: example that shows why you need all $O(KL)$ table cells (garden pathing)
- But just computing the max isn't enough. We also have to remember where we came from. (Breadcrumbs from best previous state.)

$$\Psi_n = \text{argmax}_j \delta_{n-1}(j)\theta_{j,k} \tag{4}$$

- The complexity of this is now K^2L.
- In class: example that shows why you need all $O(KL)$ table cells (garden pathing)
- But just computing the max isn't enough. We also have to remember where we came from. (Breadcrumbs from best previous state.)

$$\Psi_n = \text{argmax}_j \delta_{n-1}(j)\theta_{j,k} \tag{4}$$

- Let's do that for the sentence "come and get it"

POS	π_k	β_{k,x_1}	$\log \delta_1(k)$
MOD	0.234	0.024	-5.18
DET	0.234	0.032	-4.89
CONJ	0.234	0.024	-5.18
N	0.021	0.016	-7.99
PREP	0.021	0.024	-7.59
PRO	0.021	0.016	-7.99
V	0.234	0.121	-3.56

come and get it

Why logarithms?

1. More interpretable than a float with lots of zeros.
2. Underflow is less of an issue
3. Addition is cheaper than multiplication

$$log(ab) = log(a) + log(b) \tag{5}$$

POS	$\log \delta_1(j)$		$\log \delta_2(\text{CONJ})$
MOD	-5.18		
DET	-4.89		
CONJ	-5.18		
N	-7.99		
PREP	-7.59		
PRO	-7.99		
V	-3.56		

come **and** get it

POS	$\log \delta_1(j)$		$\log \delta_2(\text{CONJ})$
MOD	-5.18		
DET	-4.89		
CONJ	-5.18		???
N	-7.99		
PREP	-7.59		
PRO	-7.99		
V	-3.56		

come **and** get it

POS	$\log \delta_1(j)$	$\log \delta_1(j)\theta_{j,\text{CONJ}}$	$\log \delta_2(\text{CONJ})$
MOD	-5.18		
DET	-4.89		
CONJ	-5.18		???
N	-7.99		
PREP	-7.59		
PRO	-7.99		
V	-3.56		

come **and** get it

POS	$\log \delta_1(j)$	$\log \delta_1(j)\theta_{j,\text{CONJ}}$	$\log \delta_2(\text{CONJ})$
MOD	-5.18		
DET	-4.89		
CONJ	-5.18		???
N	-7.99		
PREP	-7.59		
PRO	-7.99		
V	-3.56		

come **and** get it

$$\log\left(\delta_0(\text{V})\theta_{\text{V, CONJ}}\right) = \log\delta_0(k) + \log\theta_{\text{V, CONJ}} = -3.56 + -1.65$$

POS	$\log \delta_1(j)$	$\log \delta_1(j)\theta_{j,\text{CONJ}}$	$\log \delta_2(\text{CONJ})$
MOD	-5.18		
DET	-4.89		
CONJ	-5.18		???
N	-7.99		
PREP	-7.59		
PRO	-7.99		
V	-3.56	-5.21	

come **and** get it

POS	$\log \delta_1(j)$	$\log \delta_1(j)\theta_{j,\text{CONJ}}$	$\log \delta_2(\text{CONJ})$
MOD	-5.18		
DET	-4.89		
CONJ	-5.18		???
N	-7.99	≤ -7.99	
PREP	-7.59	≤ -7.59	
PRO	-7.99	≤ -7.99	
V	-3.56	-5.21	

come **and** get it

POS	$\log \delta_1(j)$	$\log \delta_1(j)\theta_{j,\text{CONJ}}$	$\log \delta_2(\text{CONJ})$
MOD	-5.18	-8.48	
DET	-4.89	-7.72	
CONJ	-5.18	-8.47	???
N	-7.99	≤ -7.99	
PREP	-7.59	≤ -7.59	
PRO	-7.99	≤ -7.99	
V	-3.56	-5.21	

come **and** get it

POS	$\log \delta_1(j)$	$\log \delta_1(j)\theta_{j,\text{CONJ}}$	$\log \delta_2(\text{CONJ})$
MOD	-5.18	-8.48	
DET	-4.89	-7.72	
CONJ	-5.18	-8.47	???
N	-7.99	≤ -7.99	
PREP	-7.59	≤ -7.59	
PRO	-7.99	≤ -7.99	
V	-3.56	-5.21	

come **and** get it

POS	$\log \delta_1(j)$	$\log \delta_1(j)\theta_{j,\text{CONJ}}$	$\log \delta_2(\text{CONJ})$
MOD	-5.18	-8.48	
DET	-4.89	-7.72	
CONJ	-5.18	-8.47	
N	-7.99	≤ -7.99	
PREP	-7.59	≤ -7.59	
PRO	-7.99	≤ -7.99	
V	-3.56	-5.21	

come **and** get it

$$\log \delta_1(k) = -5.21 - \log \beta_{\text{CONJ, and}} =$$

POS	$\log \delta_1(j)$	$\log \delta_1(j)\theta_{j,\text{CONJ}}$	$\log \delta_2(\text{CONJ})$
MOD	-5.18	-8.48	
DET	-4.89	-7.72	
CONJ	-5.18	-8.47	
N	-7.99	≤ -7.99	
PREP	-7.59	≤ -7.59	
PRO	-7.99	≤ -7.99	
V	-3.56	-5.21	

come **and** get it

$$\log \delta_1(k) = -5.21 - \log \beta_{\text{CONJ, and}} = -5.21 - 0.64$$

POS	$\log \delta_1(j)$	$\log \delta_1(j)\theta_{j,\text{CONJ}}$	$\log \delta_2(\text{CONJ})$
MOD	-5.18	-8.48	
DET	-4.89	-7.72	
CONJ	-5.18	-8.47	-6.02
N	-7.99	≤ -7.99	
PREP	-7.59	≤ -7.59	
PRO	-7.99	≤ -7.99	
V	-3.56	-5.21	

come **and** get it

POS	$\delta_1(k)$	$\delta_2(k)$	b_2	$\delta_3(k)$	b_3	$\delta_4(k)$	b_4
MOD	-5.18						
DET	-4.89						
CONJ	-5.18	-6.02	V				
N	-7.99						
PREP	-7.59						
PRO	-7.99						
V	-3.56						
WORD	come	and		get		it	

POS	$\delta_1(k)$	$\delta_2(k)$	b_2	$\delta_3(k)$	b_3	$\delta_4(k)$	b_4
MOD	-5.18	-0.00	X				
DET	-4.89	-0.00	X				
CONJ	-5.18	-6.02	V				
N	-7.99	-0.00	X				
PREP	-7.59	-0.00	X				
PRO	-7.99	-0.00	X				
V	-3.56	-0.00	X				
WORD	come	and		get		it	

POS	$\delta_1(k)$	$\delta_2(k)$	b_2	$\delta_3(k)$	b_3	$\delta_4(k)$	b_4
MOD	-5.18	-0.00	X	-0.00	X		
DET	-4.89	-0.00	X	-0.00	X		
CONJ	-5.18	-6.02	V	-0.00	X		
N	-7.99	-0.00	X	-0.00	X		
PREP	-7.59	-0.00	X	-0.00	X		
PRO	-7.99	-0.00	X	-0.00	X		
V	-3.56	-0.00	X	-9.03	CONJ		
WORD	come	and		get		it	

POS	$\delta_1(k)$	$\delta_2(k)$	b_2	$\delta_3(k)$	b_3	$\delta_4(k)$	b_4
MOD	-5.18	-0.00	X	-0.00	X	-0.00	X
DET	-4.89	-0.00	X	-0.00	X	-0.00	X
CONJ	-5.18	-6.02	V	-0.00	X	-0.00	X
N	-7.99	-0.00	X	-0.00	X	-0.00	X
PREP	-7.59	-0.00	X	-0.00	X	-0.00	X
PRO	-7.99	-0.00	X	-0.00	X	-14.6	V
V	-3.56	-0.00	X	-9.03	CONJ	-0.00	X
WORD	come	and		get		it	

Outline

1 What is POS Tagging and why do we care?

2 HMM Intuition

3 HMM Recapitulation

4 HMM Estimation

5 NLTK Taggers

Rule-based tagger

First, we'll try to tell the computer explicitly how to tag words based on patterns that appear within the words.

```
patterns = [
(r'.*ing$', 'VBG'),                # gerunds
(r'.*ed$', 'VBD'),                 # simple past
(r'.*es$', 'VBZ'),                 # 3rd singular present
(r'.*ould$', 'MD'),                # modals
(r'.*\'s$', 'NN$'),                # possessive nouns
(r'.*s$', 'NNS'),                  # plural nouns
(r'^-?[0-9]+(.[0-9]+)?$', 'CD'),   # cardinal numbers
(r'.*', 'NN')                      # nouns (default)
]
regexp_tagger = nltk.RegexpTagger(patterns)
brown_c = nltk.corpus.brown.tagged_sents(categories=['c'])
nltk.tag.accuracy(regexp_tagger, brown_c)
```

Rule-based tagger

First, we'll try to tell the computer explicitly how to tag words based on patterns that appear within the words.

```
patterns = [
(r'.*ing$', 'VBG'),                # gerunds
(r'.*ed$', 'VBD'),                 # simple past
(r'.*es$', 'VBZ'),                 # 3rd singular present
(r'.*ould$', 'MD'),                # modals
(r'.*\'s$', 'NN$'),                # possessive nouns
(r'.*s$', 'NNS'),                  # plural nouns
(r'^-?[0-9]+(.[0-9]+)?$', 'CD'),   # cardinal numbers
(r'.*', 'NN')                      # nouns (default)
]
regexp_tagger = nltk.RegexpTagger(patterns)
brown_c = nltk.corpus.brown.tagged_sents(categories=['c'])
nltk.tag.accuracy(regexp_tagger, brown_c)
```

This doesn't do so hot; only 0.181 accuracy, but it requires no training data.

Unigram Tagger

Next, we'll create unigram taggers.

```
brown_a = nltk.corpus.brown.tagged_sents(categories=['a'])
brown_ab = nltk.corpus.brown.tagged_sents(categories=['a', 'b'])
unigram_tagger = nltk.UnigramTagger(brown_a)
unigram_tagger_bigger = nltk.UnigramTagger(brown_ab)
unigram_tagger.tag(sent)
nltk.tag.accuracy(unigram_tagger, brown_c)
nltk.tag.accuracy(unigram_tagger_bigger, brown_c)
```

Unigram Tagger

Next, we'll create unigram taggers.

```
brown_a = nltk.corpus.brown.tagged_sents(categories=['a'])
brown_ab = nltk.corpus.brown.tagged_sents(categories=['a', 'b'])
unigram_tagger = nltk.UnigramTagger(brown_a)
unigram_tagger_bigger = nltk.UnigramTagger(brown_ab)
unigram_tagger.tag(sent)
nltk.tag.accuracy(unigram_tagger, brown_c)
nltk.tag.accuracy(unigram_tagger_bigger, brown_c)
```

If we train on categories=['a','b'], then accuracy goes from 0.727 to 0.763.

Bigram Tagger

Next is a bigram tagger, which uses pairs of words rather than single words to assign a part of speech.

```
bigram_tagger = nltk.BigramTagger(brown_a, cutoff=0)
bigram_tagger.tag(sent)
nltk.tag.accuracy(bigram_tagger, brown_c)
```

Bigram Tagger

Next is a bigram tagger, which uses pairs of words rather than single words to assign a part of speech.

```
bigram_tagger = nltk.BigramTagger(brown_a, cutoff=0)
bigram_tagger.tag(sent)
nltk.tag.accuracy(bigram_tagger, brown_c)
```

Accuracy is even worse: 0.087

Combining Taggers

Instead of using the bigram's potentially sparse data, we use the better model when we can but fall back on the simpler models when the data aren't there.

```
t0 = nltk.DefaultTagger('NN')
t1 = nltk.UnigramTagger(brown_a, backoff=t0)
t2 = nltk.BigramTagger(brown_a, backoff=t1)
nltk.tag.accuracy(t2, brown_c)
```

Combining Taggers

Instead of using the bigram's potentially sparse data, we use the better model when we can but fall back on the simpler models when the data aren't there.

```
t0 = nltk.DefaultTagger('NN')
t1 = nltk.UnigramTagger(brown_a, backoff=t0)
t2 = nltk.BigramTagger(brown_a, backoff=t1)
nltk.tag.accuracy(t2, brown_c)
```

The accuracy gets to the best we've had so far: 0.779

Wrap up

- POS Tagging: important preprocessing step
- HMM: tool used for many different purposes
  - Speech recognition
  - Information extraction
  - Robotics
- Simpler "get it done" taggers in NLTK
- In class
  - Estimating transition and emission parameters from data
  - Homework questions