Probability Distributions, Viterbi Decoding, and All That

Jordan Boyd-Graber
COS/LIN 280

October 1, 2008

How do we estimate a probability?

- Suppose we want to estimate $P(w_n = \text{“dog”} | z_z = \text{“NN”})$.

How do we estimate a probability?

- Suppose we want to estimate $P(w_n = \text{“dog”} | z_z = \text{“NN”})$.

dog	**dog**	cat	horse	cow
cat	horse	cow	fly	mouse
fly	**dog**	cat	fly	**dog**
mouse	**dog**	fly	cat	cow

How do we estimate a probability?

- Suppose we want to estimate $P(w_n = \text{“dog”} | z_z = \text{“NN”})$.

dog	**dog**	cat	horse	cow
cat	horse	cow	fly	mouse
fly	**dog**	cat	fly	**dog**
mouse	**dog**	fly	cat	cow

- Maximum likelihood (ML) estimate of the probability is:

$$\hat{\theta}_i = \frac{n_i}{\sum_k n_k} \tag{1}$$

How do we estimate a probability?

- Suppose we want to estimate $P(w_n = \text{“dog”} | z_z = \text{“NN”})$.

dog	**dog**	cat	horse	cow
cat	horse	cow	fly	mouse
fly	**dog**	cat	fly	**dog**
mouse	**dog**	fly	cat	cow

- Maximum likelihood (ML) estimate of the probability is:

$$\hat{\theta}_i = \frac{n_i}{\sum_k n_k} \tag{1}$$

- Is this reasonable?

How do we estimate a probability?

- In computational linguistics, we often have a *prior* notion of what our probability distributions are going to look like (for example, non-zero, sparse, uniform, etc.).
- This estimate of a probability distribution is called the maximum a posteriori (MAP) estimate:

$$\theta_{\text{MAP}} = \text{argmax}_{\theta} f(x|\theta)g(\theta) \tag{2}$$

How do we estimate a probability?

- For a multinomial distribution (i.e. a discrete distribution, like over words):
$$\theta_i = \frac{n_i + \alpha_i}{\sum_k n_k + \alpha_k} \tag{3}$$
- α_i is called a smoothing factor, a pseudocount, etc.

How do we estimate a probability?

- For a multinomial distribution (i.e. a discrete distribution, like over words):
$$\theta_i = \frac{n_i + \alpha_i}{\sum_k n_k + \alpha_k} \tag{3}$$
- 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.

How do we estimate a probability?

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

$$\theta_i = \frac{n_i + \alpha_i}{\sum_k n_k + \alpha_k} \tag{3}$$

- 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.
- To geek out, the set $\{\alpha_1, \ldots, \alpha_N\}$ parameterizes a Dirichlet distribution, which is itself a distribution over distributions and is the conjugate prior of the Multinomial (don't need to know this).

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): $\beta_{i,j} = p(z_n = j | z_{n-1} = i)$
- β An emission matrix (matrix of size K by V): $\beta_{k,v} = p(x_n = v | z_n = k)$

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): $\beta_{i,j} = p(z_n = j | z_{n-1} = i)$
- β An emission matrix (matrix of size K by V): $\beta_{k,v} = p(x_n = v | z_n = k)$

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)

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

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.

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 ...

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	0.011	0.011	0.121	0.011	0.011

Word	get	gotta	her	here	i
Frequency	1.1	1.1	0.1	0.1	0.1
Probability	0.121	0.121	0.011	0.011	0.011

Word	into	it	life	love	my
Frequency	0.1	0.1	0.1	1.1	0.1
Probability	0.011	0.011	0.011	0.121	0.011

Word	of	old	people	stared	stood
Frequency	0.1	0.1	0.1	1.1	1.1
Probability	0.011	0.011	0.011	0.121	0.121

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, $\{x_1, \ldots, x_L\}$, we want to find a sequence $\{z_1 \ldots z_L\}$ 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 l.
- Base case:

$$\delta_1(k) = \pi_k \beta_{k,x_i} \tag{4}$$

- Recursion:

$$\delta_n(k) = \max_j \left(\delta_{n-1}(j)\theta_{j,k}\right)\beta_{k,x_n} \tag{5}$$

- The complexity of this is now K^2L.
- But just computing the max isn't enough. We also have to remember where we came from. (Breadcrumbs from best previous state.)

$$\Psi_n = \text{argmax}_j \delta_{n-1}(j)\theta_{j,k} \tag{6}$$

- The complexity of this is now $K^2 L$.
- But just computing the max isn't enough. We also have to remember where we came from. (Breadcrumbs from best previous state.)

$$\Psi_n = \text{argmax}_j \delta_{n-1}(j)\theta_{j,k} \tag{6}$$

- 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

POS	$\log \delta_1(j)$		$\log \delta_1(\text{CONJ})$
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	$\log \delta_1(j)$		$\log \delta_1(\text{CONJ})$
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	$\log \delta_1(j)$	$\log \delta_1(j)\theta_{j,\mathrm{CONJ}}$	$\log \delta_1(\mathrm{CONJ})$
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	$\log \delta_1(j)$	$\log \delta_1(j)\theta_{j,\text{CONJ}}$	$\log \delta_1(\text{CONJ})$
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	$\log \delta_1(j)$	$\log \delta_1(j)\theta_{j,\text{CONJ}}$	$\log \delta_1(\text{CONJ})$
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	$\log \delta_1(j)$	$\log \delta_1(j)\theta_{j,\text{CONJ}}$	$\log \delta_1(\text{CONJ})$
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	$\log \delta_1(j)$	$\log \delta_1(j)\theta_{j,\mathrm{CONJ}}$	$\log \delta_1(\mathrm{CONJ})$
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	$\log \delta_1(j)$	$\log \delta_1(j)\theta_{j,\text{CONJ}}$	$\log \delta_1(\text{CONJ})$
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	$\log \delta_1(j)$	$\log \delta_1(j)\theta_{j,\mathrm{CONJ}}$	$\log \delta_1(\mathrm{CONJ})$
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

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

POS	$\log \delta_1(j)$	$\log \delta_1(j)\theta_{j,\text{CONJ}}$	$\log \delta_1(\text{CONJ})$
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

$$\log \delta_1(k) = -5.21 - \log \beta_{\text{CONJ, and}} = -5.21 - 0.81$$

POS	$\log \delta_1(j)$	$\log \delta_1(j)\theta_{j,\text{CONJ}}$	$\log \delta_1(\text{CONJ})$
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	

Rule-based tagger

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

```
import nltk
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)
sent = nltk.corpus.brown.sents(categories=['c'])[13]
correct_sent = nltk.corpus.brown.tagged_sents(categories=['c']
regexp_tagger.tag(sent)
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.

```
import nltk
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)
sent = nltk.corpus.brown.sents(categories=['c'])[13]
correct_sent = nltk.corpus.brown.tagged_sents(categories=['c']
regexp_tagger.tag(sent)
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.

Affix Tagger

Now, train an affix tagger, which uses the end of words rather than the whole word.

```
affix_tagger = nltk.AffixTagger(brown_a, affix_length=-2, min_
affix_tagger.tag(sent)
nltk.tag.accuracy(affix_tagger, brown_c)
```

Affix Tagger

Now, train an affix tagger, which uses the end of words rather than the whole word.

```
affix_tagger = nltk.AffixTagger(brown_a, affix_length=-2, min_
affix_tagger.tag(sent)
nltk.tag.accuracy(affix_tagger, brown_c)
```

Accuracy isn't so hot: 0.212

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 isn'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 isn'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