CS224N Section 1

8 April 2005

Bill MacCartney

Welcome and introductions

Format and content of sections

- Mix of theoretical and practical topics
- Emphasis on simple examples worked out in detail
- Aim for lots of interaction, students at blackboard
- I'll come with some topics prepared, but I want to be guided by you

What should we talk about today?

- Information theory: intuitions and examples
  - entropy, joint entropy, conditional entropy, mutual information
  - relative entropy (KL divergence), cross entropy, perplexity
  - proving that KL(p, q) >= 0
- Smoothing: examples, proofs, implementation
  - absolute discounting example
  - how to prove you have a proper probability distribution
  - Good-Turing smoothing tricks
- Java implementation
  - representing huge models efficiently
- What topics are on your minds? What's not clear?

Let's start with a motivating example…

Motivating example

Every six months, the chairman of the Galactic Reserve Board, Alien Greenspine, issues

In Chairman Greenspine's last six reports, he said:

1 "F U, U F!"
2 "F U, U F F F!"
3 "U F, F U!"
4 "F!"
5 "F, F, F!!"
6 "" [* the chairman appeared but said nothing]

(The chairman is terse but nevertheless very expressive.)

You are a bond trader, and Chairman Greenspine will be issuing his next report on Tuesday. If you can successfully predict what the chairman is most likely to say, you can make a killing and buy yourself a friggin Death Star. What's more, if you can not only predict what he's *most* likely to say, but accurately estimate the probability of *any* particular utterance, then you can put together a bundle of swaptions, straddles, and other derivatives virtually certain to earn a spectacular profit -- enough to let you buy an entire star system, and finally take your rightful place among the Masters of the Universe.

What you need is a language model.

A unigram model

Let's take a closer look at what Chairman Greenspine said in the past. We'll omit the punctuation (since it was included only for comic effect) and put a stop symbol after each utterance:

1 F U U F .
2 F U U F F F .
3 U F F U .
4 F .
5 F F F .
6 .

Let C(x) be the observed count of unigram x
Let o(x) be the observed frequency of unigram x

x	C(x)	o(x)
F	12	1/2
U	6	1/4
.	6	1/4
total	24	1

o(x) is a multinomial probability distribution with 2 free parameters. (Why?) We can interpret o(x) as a model of a stochastic process in which the event space is {F, U, .} and events are sampled i.i.d. according to a some "true" multinomial distribution p(x) whose parameters are unknown. The parameters of o(x) (that is, the observed frequencies) are the MLE estimates of the unknown true parameters.

A bigram model

This time we'll put a stop symbol before and after each utterance:

1 . F U U F .
2 . F U U F F F .
3 . U F F U .
4 . F .
5 . F F F .
6 . .

Let C(x, y) be the observed count of bigram xy
Let o(x, y) be the observed frequency of bigram xy

C(x, y)	F	U	.	total
F	5	3	4	12
U	3	2	1	6
.	4	1	1	6
total	12	6	6	24

o(x, y)	F	U	.	total
F	5/24	1/8	1/6	1/2
U	1/8	1/12	1/24	1/4
.	1/6	1/24	1/24	1/4
total	1/2	1/4	1/4	1

(These are called "contingency tables".)

o(x, y) is a multinomial probability distribution with 8 free parameters. (Why?) We can interpret o(x, y) as a model of a stochastic process in which the event space is {F, U, .} × {F, U, .} and events are sampled according to a some "true" multinomial distribution p(x, y) whose parameters are unknown. The parameters of o(x, y) (that is, the observed frequencies) are the MLE estimates of the unknown true parameters.

What are the marginal distributions o(x) and o(y)?

What are the conditional distributions o(y | x) and o(x | y)?

o(y | x) = o(x, y) / o(x)
o(x | y) = o(x, y) / o(y)

o(y | x)	F	U	.	total
F	5/12	1/4	1/3	1
U	1/2	1/3	1/6	1
.	2/3	1/6	1/6	1

o(x | y)	F	U	.
F	5/12	1/2	2/3
U	1/4	1/3	1/6
.	1/3	1/6	1/6
total	1	1	1

Why didn't I show totals in the other direction as well?

Entropy

If X is a random variable whose distribution is p (which we write "X ~ p"), then we can define the entropy of X, H(X), as follows:

$$H(X) = -\sum_x p(x) \lg p(x)$$
$$= -E_p [\lg p(x)]$$

(Sometimes we talk about the entropy of a distribution, H(p), but this is just another way of talking about H(X), assuming X ~p.)

Let's calculate the entropy of our observed unigram distribution, o(x):

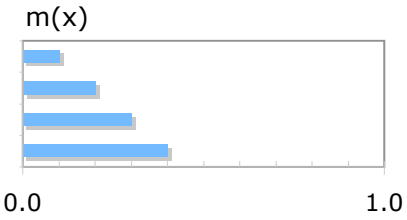
x	o(x)	- o(x) lg o(x)
F	1/2	0.5
U	1/4	0.5
.	1/4	0.5
total	1	1.5

⇐ entropy

Note that entropy is (the negative of) the expected value of the log probability of x. Log probabilities come up a lot in statistical modeling. Since probabilities are always ≤ 1, log probabilities are always ≤ 0. Therefore, entropy is always ≥ 0.

What does entropy mean, intuitively? There are several ways to interpret entropy, but I find it useful to think of it as a measure of the unevenness of the distribution. To get a feel for this, try fiddling with the parameters of the following multinomial distribution:

x	m(x)	- m(x) lg m(x)
lion	0.10	0.33
hippo	0.20	0.46
chimp	0.30	0.52
zebra	0.40	0.53
total	1.00	1.85



What's the lowest the entropy can be, and what parameters achieve that minimum?

What's the highest the entropy can be, and what parameters achieve that maximum?

What if m(x) = 0 for some x? We'll pretend that 0 lg 0 = 0, so that events with probability 0 do not affect the entropy calculation. (If you peek at the formulas in the table above, you'll see how I took care of this.)

Joint Entropy

If random variables X and Y have joint distribution p(x, y), then we can define the joint entropy of X and Y, H(X, Y), as follows:

$$H(X, Y) = -\sum_x p(x, y) \lg p(x, y)$$
$$= -E_p [\lg p(x, y)]$$

Let's calculate the entropy of our observed bigram distribution, o(x, y):

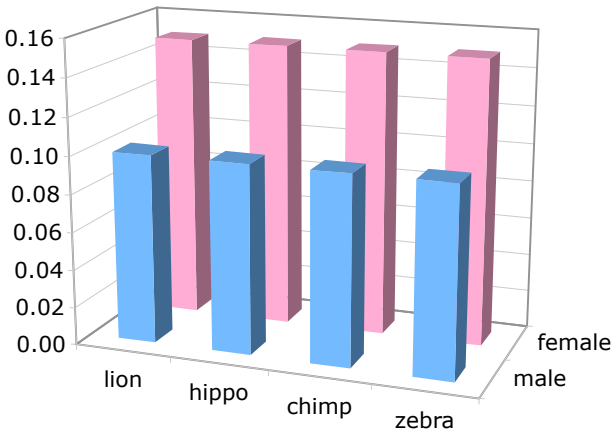
o(x, y)	F	U	.	total
F	5/24	1/8	1/6	1/2
U	1/8	1/12	1/24	1/4
.	1/6	1/24	1/24	1/4
total	1/2	1/4	1/4	1

- o(x, y) · lg o(x, y)	F	U	.	total
F	0.471	0.375	0.431	1.277
U	0.375	0.299	0.191	0.865
.	0.431	0.191	0.191	0.813
total	1.277	0.865	0.813	2.955 ⇐ joint entropy H(X, Y)

Try fiddling with the parameters of the following joint distribution m(x, y), and observe what happens to the joint entropy:

m(x, y)	male	female	total
lion	0.10	0.15	0.25
hippo	0.10	0.15	0.25
chimp	0.10	0.15	0.25
zebra	0.10	0.15	0.25
total	0.40	0.60	1.00

- m(x, y) · lg m(x, y)	male	female	total
lion	0.332	0.411	0.743
hippo	0.332	0.411	0.743
chimp	0.332	0.411	0.743
zebra	0.332	0.411	0.743
total	1.329	1.642	2.971 ⇐ joint entropy H(X, Y)



Conditional Entropy

If random variables X and Y have joint distribution p(x, y), then we can define the conditional entropy of Y given X, H(Y | X), as follows:

$$
\begin{aligned}
H(Y \mid X) &= -\sum_{x,y} p(x, y) \lg p(y \mid x) \\
&= \sum_x p(x) \left[-\sum_y p(y \mid x) \lg p(y \mid x) \right] \\
&= \sum_x p(x) \left[H(Y \mid X = x) \right] \\
&= E_p \left[H(Y \mid X = x) \right] \\
&= H(X, Y) - H(X)
\end{aligned}
$$

Let's return to our observed bigram distribution, o(x, y). What is H(Y | X)?

o(y | x)	F	U	.	total
F	5/12	1/4	1/3	1
U	1/2	1/3	1/6	1
.	2/3	1/6	1/6	1

Here's how to compute H(Y | X) as the expectation of H(Y | X = x):

- o(y | x) · lg o(y | x)	F	U	.	H(Y | X = x)	o(x)	o(x) H(Y | X = x)
F	0.526	0.500	0.528	1.555	1/2	0.777
U	0.500	0.528	0.431	1.459	1/4	0.365
.	0.390	0.431	0.431	1.252	1/4	0.313
					H(Y | X) ⇒	1.455

Or, we can just take the difference of our previously computed values for the joint entropy H(X, Y) and the entropy H(X):

H(X, Y)	2.955
- H(X)	-1.500
H(Y | X)	1.455 ⇐ H(Y | X)

Relative entropy (KL divergence)

If we have two probability distributions, p(x) and q(x), we can define the relative entropy (aka KL divergence) between p and q, D(p, q), as follows:

$$
\begin{aligned}
D(p, q) &= \sum_x p(x) \lg [\, p(x) / q(x) \,] \\
&= E_p [\, \lg [\, p(x) / q(x) \,] \,] \\
&= E_p [\, \lg p(x) - \lg q(x) \,] \\
&= E_p [\, \lg p(x) \,] - E_p [\, \lg q(x) \,] \\
&= H(p, q) - H(p)
\end{aligned}
$$

(The quantity H(p, q) is called the *cross entropy*.)

Relative entropy measures how much two probability distributions differ. (But note that is *not* a symmetric measure!) Identical distributions have zero relative entropy. Non-identical distributions have positive relative entropy. Relative entropy is never negative.

Suppose we want to compare our observed unigram distribution o(x) to some arbitrary model distribution m(x). What is the relative entropy between them?

$$D(o, m) = \sum_x o(x) \lg [\, o(x) / m(x) \,]$$

x	o(x)	m(x)	o(x) lg [o(x) / m(x)]	
F	0.50	0.01	2.82	
U	0.25	0.01	1.16	
.	0.25	0.01	1.16	
S	0.00	0.97	0.00	
total	1	1	5.14	⇐ KL divergence

Try fiddling with the parameters of m(x) and see what it does to the KL divergence. What parameters minimize the divergence? Maximize?

What happens when o(x) = 0? (lg 0 is normally undefined!)

> Events that cannot happen (according to o(x)) do not contribute to KL divergence between o(x) and any other distribution.

What happens when m(x) = 0? (division by 0!)

> If an event x was observed (o(x) > 0) but your model says it can't (m(x) = 0), then your model is infinitely surprised: D(o, m) = ∞.

Why is D(p, q) >= 0? Can you prove it? (Hint: use Jensen's Inequality.)

Absolute discounting

Idea: reduce counts of observed event types by a fixed amount δ, and reallocate the count mass to unobserved event types. Absolute discounting is simple and gives quite good results.

Terminology:

x	an event (a *type*) (e.g. a bigram)
C	count of all observations (*tokens*) (e.g. training size)
C(x)	count of observations (tokens) of type x (e.g. bigram counts)
V	# of event types: $\|\{x\}\|$ (e.g. size of vocabulary)
N_r	# of event types observed r times: $\|\{x: C(x) = r\}\|$
δ	a number between 0 and 1 (e.g. 0.75)

For seen types, we deduct δ from the count mass:

$$P_{ad}(x) = (C(x) - \delta) / C \qquad \text{if } C(x) > 0$$

How much count mass did we harvest by doing this? We took δ from each of $V - N_0$ types, so we have $\delta(V - N_0)$ to redistribute among the N_0 unseen types. So each unseen type will get a count mass of $\delta(V - N_0) / N_0$:

$$P_{ad}(x) = (\delta(V - N_0) / N_0) / C \qquad \text{if } C(x) = 0$$

To see how this works, let's go back to the Alien Greenspine example and look at bigram counts. To bring unseens into the picture, let's suppose Greenspine *could* say another word, S, giving rise to 7 new unseen bigrams:

x	C(x)	$-\delta$		C'(x)	$P_{ad}(x)$	$P_{mle}(x)$
F F	5	-0.75		4.25	0.18	0.21
F .	4	-0.75		3.25	0.14	0.17
. F	4	-0.75		3.25	0.14	0.17
F U	3	-0.75		2.25	0.09	0.13
U F	3	-0.75		2.25	0.09	0.13
U U	2	-0.75		1.25	0.05	0.08
U .	1	-0.75		0.25	0.01	0.04
. U	1	-0.75		0.25	0.01	0.04
. .	1	-0.75		0.25	0.01	0.04
subtotal	24	-6.75		17.25	0.72	1.00
		÷ 7 =				
F S	0		+0.96	0.96	0.04	0.00
U S	0		+0.96	0.96	0.04	0.00
. S	0		+0.96	0.96	0.04	0.00
S F	0		+0.96	0.96	0.04	0.00
S U	0		+0.96	0.96	0.04	0.00
S .	0		+0.96	0.96	0.04	0.00
S S	0		+0.96	0.96	0.04	0.00
total	24	-6.75	+6.75	24.00	1.00	1.00

The probabilities add up to 1, which is good. But in general, how do we prove that a discounting method yields a proper probability distribution?

Also, how do you choose a good value for d? The value 0.75 is often recommended, but you can also use held-out data to tune this parameter.

A shortcoming of absolute discounting is that it requires the assumption of a fixed vocabulary size V. What can be done to mitigate this problem?

Look down the column of P_{ad} probabilities. Anything trouble you?

Proper probability distributions

To prove that a function p is a probability distribution, you must show:

1 $\forall x\ p(x) > 0$

2 $\sum_x p(x) = 1$

The first is generally trivial. The second can be more challenging.

A proof for absolute discounting will illustrate the general idea, which you can then extend to your own discounting scheme.

$\sum_x P_{abs}(x)$

$= \sum_{x:C(x)>0} P_{abs}(x) + \sum_{x:C(x)=0} P_{abs}(x)$

$= \sum_{x:C(x)>0} (C(x) - \delta)/C + \sum_{x:C(x)=0} (V - N_0)\delta/N_0C$

[$V - N_0$ terms] [N_0 terms]

$= [\ \sum_{x:C(x)>0} C(x)\ ]/C - (V - N_0)\delta/C + (V - N_0)\delta/C$

$= [\ \sum_x C(x)\ ] / C$

$= C / C$

$= 1$

Good-Turing smoothing

Idea: redistribute the count mass of types observed r + 1 times evenly among types observed r times. Then, estimate P(x) as r* / C, where r* is an adjusted count for types observed r times. We want r* such that:

$$r^* \times N_r = (r + 1) \times N_{r+1}$$

$$r^* = (r + 1) \times N_{r+1} / N_r$$

$$P_{GT}(x) = ((r + 1) \times N_{r+1} / N_r) / C$$

To see how this works, let's go back to the Alien Greenspine example and look at bigram counts. To make the example more interesting, we'll assume we've also heard Greenspan say, "S S S S S S S S S S S S!!!", giving us another 12 bigram observations, as summarized in the following table of counts:

x	r	N_r	N_{r+1}	r*	$P_{GT}(x)$
S S	11	1	0	0.00	0.00
F F	5	1	0	0.00	0.00
F .	4	2	1	2.50	0.07
. F	4	2	1	2.50	0.07
F U	3	2	2	4.00	0.11
U F	3	2	2	4.00	0.11
U U	2	1	2	6.00	0.17
U .	1	4	1	0.50	0.01
. U	1	4	1	0.50	0.01
. .	1	4	1	0.50	0.01
S .	1	4	1	0.50	0.01
F S	0	5	4	0.80	0.02
U S	0	5	4	0.80	0.02
. S	0	5	4	0.80	0.02
S F	0	5	4	0.80	0.02
S U	0	5	4	0.80	0.02
total	36			25.00	0.69

Houston, we have a problem. The probabilities do not add up to 1. The problem, of course, is that for high values of r (i.e., for high-frequency bigrams), N_{r+1} is quite likely to be 0. A better way to define r* is:

$$r^* = (r + 1) \times E[N_{r+1}] / E[N_r]$$

where $E[N_r]$ means the expected number of event types (bigrams) observed r times. But the next question is, expected according to what distribution? We can re-interpret our first formulation of Good-Turing as an attempt to estimate the expected values of N_r by their observed frequencies, but the observed frequencies yield a very unsmooth distribution. What we need is a smoothed distribution over N_r.

In practice, two approaches are used. One is to use the Good-Turing estimates only for frequencies r < k for some constant cutoff k. Above this, the MLE estimates are used. Suppose we choose cutoff k = 5:

x	r	$P_{GT}(x)$	$P_{mle}(x)$	$P_{GT'}(x)$
S S	11	0.00	0.31	0.31
F F	5	0.00	0.14	0.14
F .	4	0.07	0.11	0.07
. F	4	0.07	0.11	0.07
F U	3	0.11	0.08	0.11
U F	3	0.11	0.08	0.11
U U	2	0.17	0.06	0.17
U .	1	0.01	0.03	0.01
. U	1	0.01	0.03	0.01
. .	1	0.01	0.03	0.01
S .	1	0.01	0.03	0.01
F S	0	0.02	0.00	0.02
U S	0	0.02	0.00	0.02
. S	0	0.02	0.00	0.02
S F	0	0.02	0.00	0.02
S U	0	0.02	0.00	0.02
total	36	0.69	1.00	1.14

Uh-oh, the probabilities do not sum to 1. Well crap, we're never gonna get that Death Star. But wait! It's not hard to fix. One solution is just to renormalize the probabilities by dividing by their sum:

x	r	$P_{GT}(x)$	$P_{mle}(x)$	$P_{GT'}(x)$	$P_{GT''}(x)$
S S	11	0.00	0.31	0.31	0.27
F F	5	0.00	0.14	0.14	0.12
F .	4	0.07	0.11	0.07	0.06
. F	4	0.07	0.11	0.07	0.06
F U	3	0.11	0.08	0.11	0.10
U F	3	0.11	0.08	0.11	0.10
U U	2	0.17	0.06	0.17	0.15
U .	1	0.01	0.03	0.01	0.01
. U	1	0.01	0.03	0.01	0.01
. .	1	0.01	0.03	0.01	0.01
S .	1	0.01	0.03	0.01	0.01
F S	0	0.02	0.00	0.02	0.02
U S	0	0.02	0.00	0.02	0.02
. S	0	0.02	0.00	0.02	0.02
S F	0	0.02	0.00	0.02	0.02
S U	0	0.02	0.00	0.02	0.02
total	36	0.69	1.00	1.14	1.00

Now that's a rock-solid Good-Turing smoothed distribution!

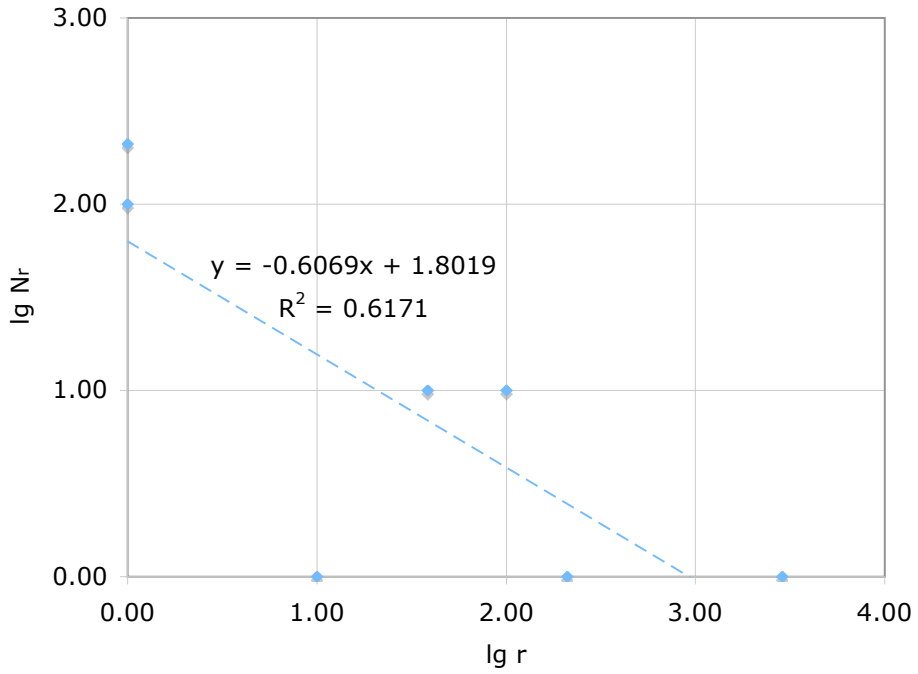
The other approach is to fit some function S through the observed values (r, N_r), and to use the smoothed values $S(r)$ to estimate $E[N_r]$. Gale & Sampson (1995) suggest using a power curve $N_r = ar^b$, with $b < -1$. Because the log of a power curve is a straight line, we can fit S to the data using simple linear regression if we first transform everything into log space. The logarithmic form of the power curve is:

$$\lg N_r = \lg a + b \lg r$$

The observed distribution of N_r, when transformed into log space, looks like this:

r	N_r	lg r	lg N_r
11	1	3.46	0.00
5	1	2.32	0.00
4	2	2.00	1.00
3	2	1.58	1.00
2	1	1.00	0.00
1	4	0.00	2.00
0	5	0.00	2.32

Here's a graph, along with a line fit using Excel's linear regression tool:



The fit is rather poor, but it gives us a and b to use for S: lg a = 1.8019 ⇒ a = 3.4868, and b = -0.6069. Hmm, we don't have b < -1, but then our data is scanty and not very realistic. Let's pretend that the linear regression gave us b = -1.5. (Why? What happens if we use the real value?) So S is:

S(r) = 3.4868 · r^-1.5

Now we can smooth N_r:

r	N_r	S(r)
11	1	0.10
5	1	0.31
4	2	0.44
3	2	0.67
2	1	1.23
1	4	3.49
0	5	#DIV/0!

a	3.49
b	-1.50

We can't evaluate S(r) for r = 0, but it won't be a problem. For r > 0, we'll use S(r) to generate adjusted counts r*:

$$r^* = (r + 1) \times S(r + 1) / S(r)$$

r	N_r	r*
11	1	10.53
5	1	4.56
4	2	3.58
3	2	2.60
2	1	1.63
1	4	0.71
0	5	

The probabilities for seen types will then be estimated by $P(x) = r^* / C$, as before. Now the count mass left over for unseen types is:

$$\text{leftover} = C - \sum_{r>0} N_r \cdot r^* \approx N_1$$

(Why?) If we use the exact form for the leftover count mass, we can allocate it equally among unseen types, and the result will be a proper probability distribution.

Alternatively, we can approximate the leftover count mass by N_1, so that each unseen type gets adjusted count $r^* = N_1/N_0$. But, we'll then need to renormalize the resulting distribution to ensure that it is proper. Let's put it all together:

$$r^* = (r + 1) \times S(r + 1) / S(r) \quad \text{if } r > 0$$
$$r^* = N_1 / N_0 \quad \text{if } r = 0$$
$$P_{GT}(x) = r^* / C$$

x	r	r*	$P_{GT}(x)$	$P_{GT'}(x)$	$P_{mle}(x)$
S S	11	10.53	0.29	0.29	0.31
F F	5	4.56	0.13	0.13	0.14
F .	4	3.58	0.10	0.10	0.11
. F	4	3.58	0.10	0.10	0.11
F U	3	2.60	0.07	0.07	0.08
U F	3	2.60	0.07	0.07	0.08
U U	2	1.63	0.05	0.05	0.06
U .	1	0.71	0.02	0.02	0.03
. U	1	0.71	0.02	0.02	0.03
. .	1	0.71	0.02	0.02	0.03
S .	1	0.71	0.02	0.02	0.03
F S	0	0.80	0.02	0.02	0.00
U S	0	0.80	0.02	0.02	0.00
. S	0	0.80	0.02	0.02	0.00
S F	0	0.80	0.02	0.02	0.00
S U	0	0.80	0.02	0.02	0.00
total	36		0.997	1.00	1.00

Would you like an army of clones with your Death Star?

Java implementation

Any confusion about roulette wheel sampling in GenerateWord?

You want to train a trigram model on 10 million words, but you keep running out of memory.

- Don't use a V × V × V matrix!
- CounterMap? Good, but needs to be elaborated.
- Intern your Strings!
- Maybe use cutoffs?
- Good ol' virtual memory
- Disk-backed models

Questions about Java 1.5?

Tips

Work with a partner!

Make a very small dataset for use during development (esp. debugging).

Investigate and report on the variance of your model results.

Investigate and report on the learning curve (performance as a function of training size).

Use validation set to tune hyperparameters.