CS224N NLP

Christopher Manning

Spring 2009

Borrows slides from Bob Carpenter, Dan Klein, Roger Levy, Josh Goodman, Dan Jurafsky

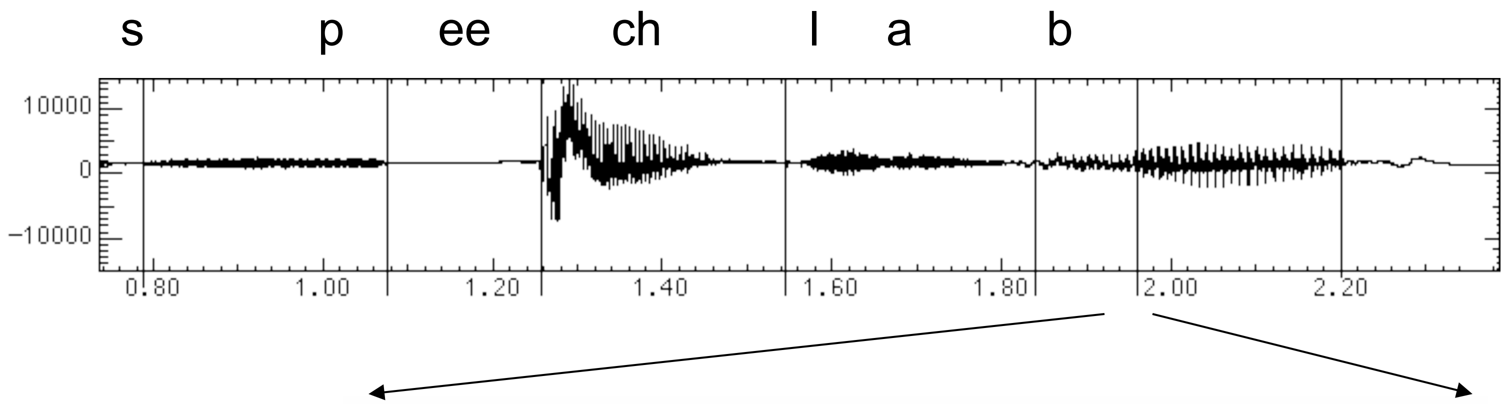
Questions that linguistics should answer

- What kinds of things do people say?
- What do these things say/ask/request about the world?
  - Example: *In addition to this, she insisted that women were regarded as a different existence from men unfairly.*
- Text *corpora* give us data with which to answer these questions
- They are an externalization of linguistic knowledge
- What words, rules, statistical facts do we find?
- How can we build programs that learn effectively from this data, and can then do NLP tasks?

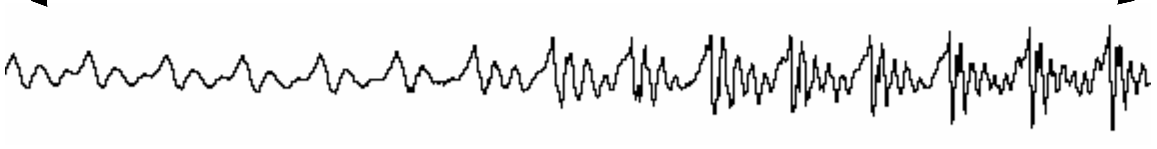
Speech Recognition: Acoustic Waves

- Human speech generates a wave
  - like a loudspeaker moving

- A wave for the words “speech lab” looks like:



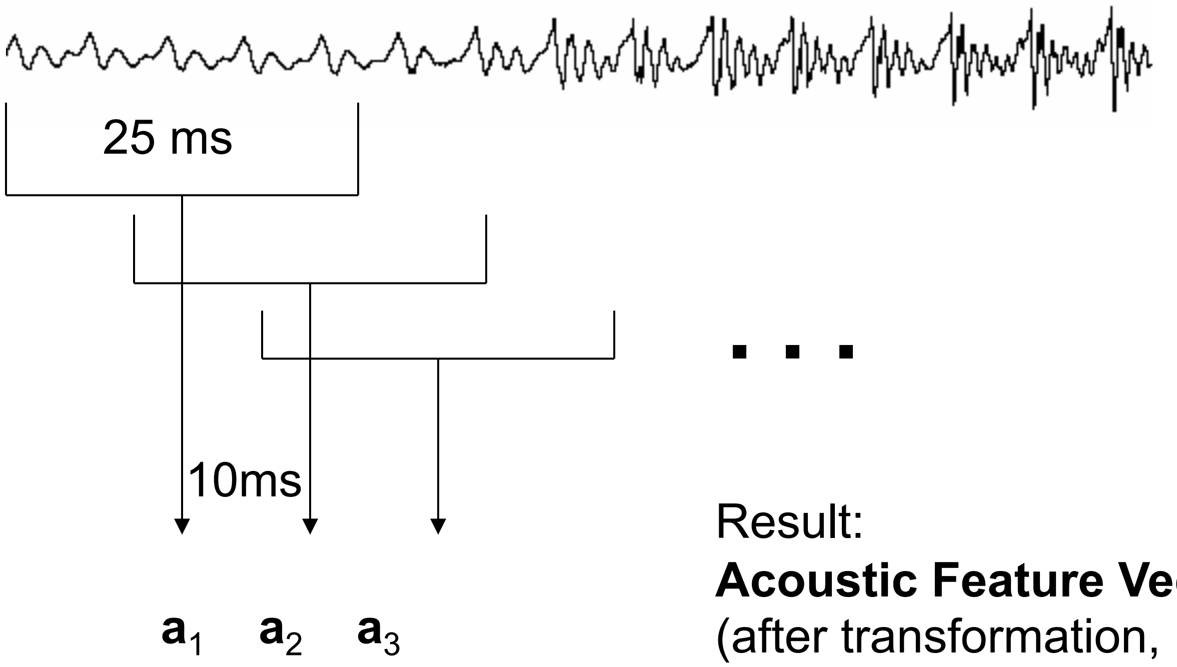
“l” to “a” transition:



Graphs from Simon Arnfield’s web tutorial on speech, Sheffield:
http://www.psyc.leeds.ac.uk/research/cogn/speech/tutorial/

Acoustic Sampling

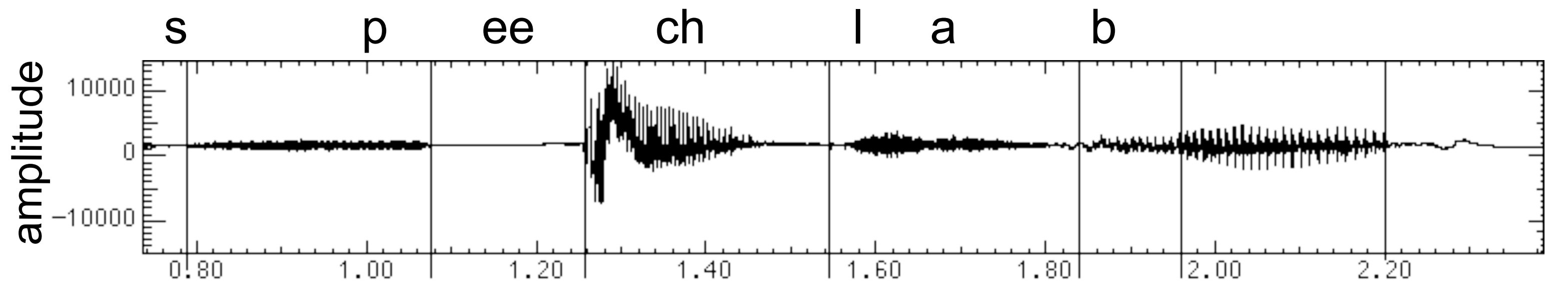
- 10 ms frame (ms = millisecond = 1/1000 second)
- ~25 ms window around frame [wide band] to allow/smooth signal processing – it let's you see formants



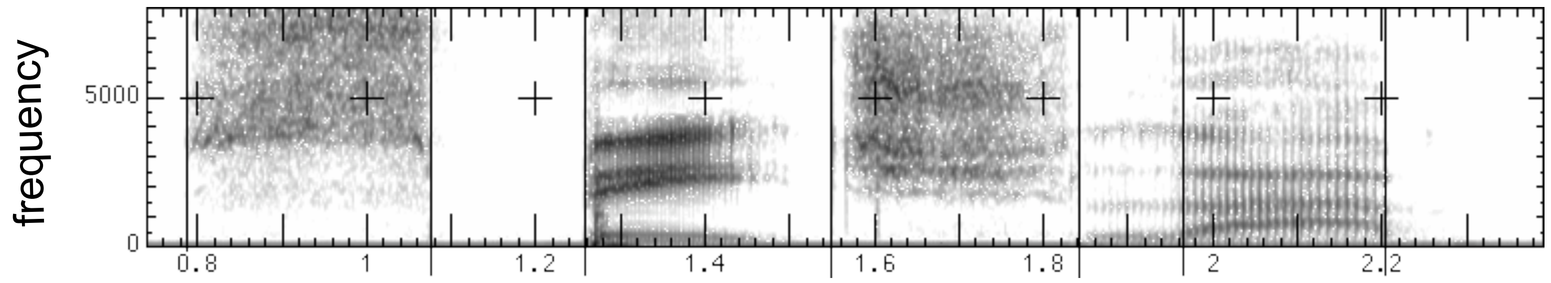
Result:
Acoustic Feature Vectors
(after transformation, numbers in roughly $\mathbf{R}^{14}$)

Spectral Analysis

- Frequency gives pitch; amplitude gives volume
  - sampling at ~8 kHz phone, ~16 kHz mic (kHz=1000 cycles/sec)



- Fourier transform of wave displayed as a spectrogram
  - darkness indicates energy at each frequency
  - hundreds to thousands of frequency samples



The Speech Recognition Problem

- The **Recognition Problem: Noisy channel model**
  - Build generative model of encoding: We started with English words, they were encoded as an audio signal, and we now wish to decode.
  - Find most likely sequence **w** of "words" given the sequence of acoustic observation vectors **a**
  - Use Bayes' rule to create a **generative model** and then decode
  - $\text{ArgMax}_{\mathbf{w}}\ P(\mathbf{w}|\mathbf{a}) = \text{ArgMax}_{\mathbf{w}}\ P(\mathbf{a}|\mathbf{w})\, P(\mathbf{w}) / P(\mathbf{a})$
    $= \text{ArgMax}_{\mathbf{w}}\ P(\mathbf{a}|\mathbf{w})\, P(\mathbf{w})$

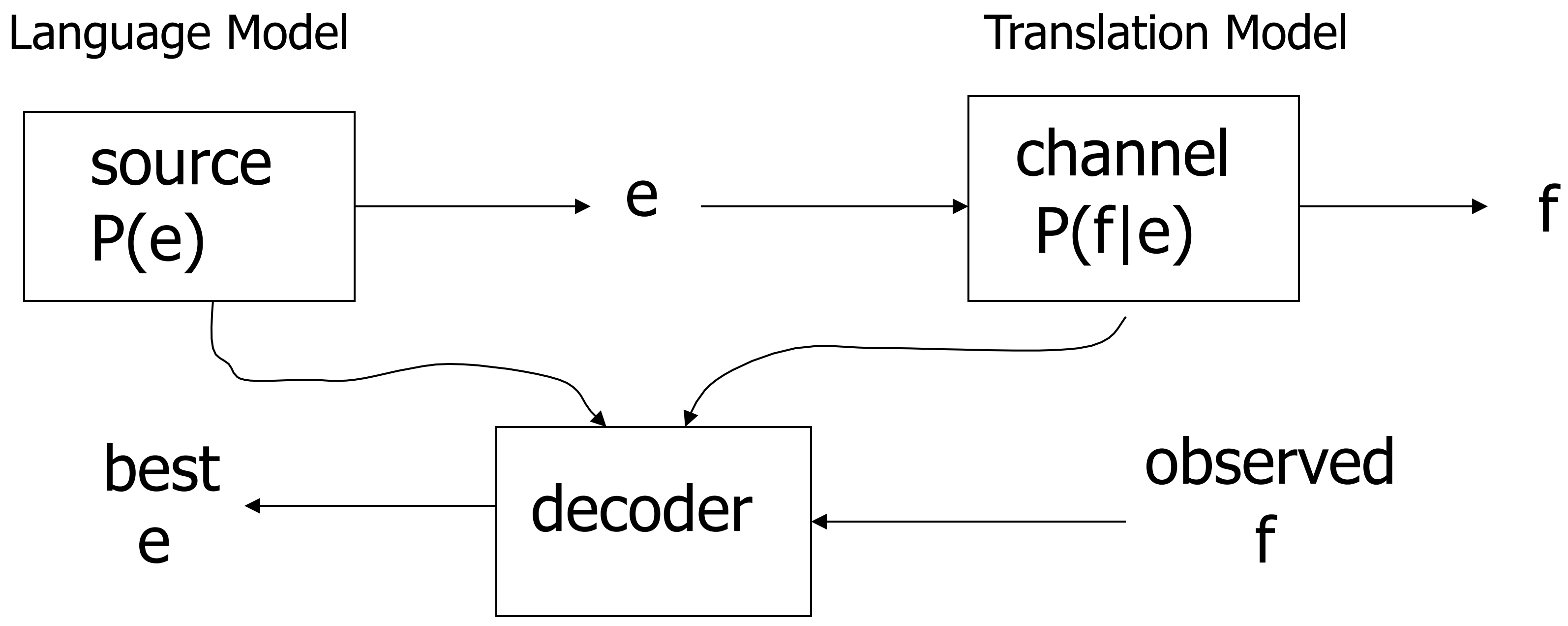
- **Acoustic Model:** $P(\mathbf{a}|\mathbf{w})$
- **Language Model:** $P(\mathbf{w})$ ← A probabilistic theory of a language

- Why is this progress?

MT: Just a Code?

- "Also knowing nothing official about, but having guessed and inferred considerable about, the powerful new mechanized methods in cryptography—methods which I believe succeed even when one does not know what language has been coded—one naturally wonders if the problem of translation could conceivably be treated as a problem in cryptography. When I look at an article in Russian, I say: 'This is really written in English, but it has been coded in some strange symbols. I will now proceed to decode.' "
  - Warren Weaver (1955:18, quoting a letter he wrote in 1947)

MT System Components



$$\underset{e}{\operatorname{argmax}} P(e|f) = \underset{e}{\operatorname{argmax}} P(f|e)P(e)$$

Other Noisy-Channel Processes

- Handwriting recognition

$$P(text \mid strokes) \propto P(text)P(strokes \mid text)$$

- OCR

$$P(text \mid pixels) \propto P(text)P(pixels \mid text)$$

- Spelling Correction

$$P(text \mid typos) \propto P(text)P(typos \mid text)$$

Probabilistic Language Models

- Want to build models which assign scores to sentences.
  - P(I saw a van) >> P(eyes awe of an)
  - Not really grammaticality: P(artichokes intimidate zippers) ≈ 0

- One option: empirical distribution over sentences?
  - Problem: doesn't generalize (at all)

- Two major components of generalization
  - ***Backoff***: sentences generated in small steps which can be recombined in other ways
  - ***Discounting***: allow for the possibility of unseen events

N-Gram Language Models

- No loss of generality to break sentence probability down with the chain rule

$$P(w_1 w_2 \ldots w_n) = \prod_i P(w_i \mid w_1 w_2 \ldots w_{i-1})$$

- Too many histories!
  - P(??? | No loss of generality to break sentence) ?
  - P(??? | the water is so transparent that) ?

- N-gram solution: assume each word depends only on a short linear history (a Markov assumption)

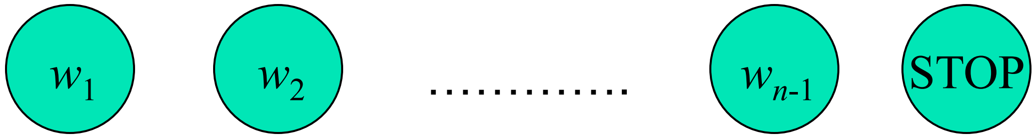
$$P(w_1 w_2 \ldots w_n) = \prod_i P(w_i \mid w_{i-k} \ldots w_{i-1})$$

Unigram Models

- Simplest case: unigrams

$$P(w_1 w_2 \ldots w_n) = \prod_i P(w_i)$$

- Generative process: pick a word, pick a word, …
- As a graphical model:

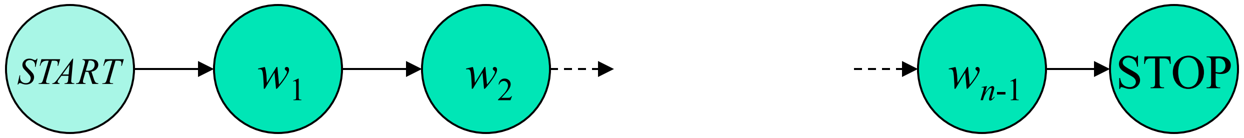


- To make this a proper distribution over sentences, we have to generate a special STOP symbol last. (Why?)
- Examples:
  - [fifth, an, of, futures, the, an, incorporated, a, a, the, inflation, most, dollars, quarter, in, is, mass.]
  - [thrift, did, eighty, said, hard, 'm, july, bullish]
  - [that, or, limited, the]
  - []
  - [after, any, on, consistently, hospital, lake, of, of, other, and, factors, raised, analyst, too, allowed, mexico, never, consider, fall, bungled, davison, that, obtain, price, lines, the, to, sass, the, the, further, board, a, details, machinists, the, companies, which, rivals, an, because, longer, oakes, percent, a, they, three, edward, it, currier, an, within, in, three, wrote, is, you, s., longer, institute, dentistry, pay, however, said, possible, to, rooms, hiding, eggs, approximate, financial, canada, the, so, workers, advancers, half, between, nasdaq]

Bigram Models

- Big problem with unigrams: P(the the the the) >> P(I like ice cream)!
- Condition on previous word:

$$P(w_1 w_2 \ldots w_n) = \prod_i P(w_i \mid w_{i-1})$$



- Any better?
  - [texaco, rose, one, in, this, issue, is, pursuing, growth, in, a, boiler, house, said, mr., gurria, mexico, 's, motion, control, proposal, without, permission, from, five, hundred, fifty, five, yen]
  - [outside, new, car, parking, lot, of, the, agreement, reached]
  - [although, common, shares, rose, forty, six, point, four, hundred, dollars, from, thirty, seconds, at, the, greatest, play, disingenuous, to, be, reset, annually, the, buy, out, of, american, brands, vying, for, mr., womack, currently, sharedata, incorporated, believe, chemical, prices, undoubtedly, will, be, as, much, is, scheduled, to, conscientious, teaching]
  - [this, would, be, a, record, november]

Regular Languages?

- N-gram models are (weighted) regular processes
  - You can extend to trigrams, fourgrams, …
  - Why can't we model language like this?
    - Linguists have many arguments why language can't be regular.
    - Long-distance effects:

      "The computer which I had just put into the machine room on the fifth floor crashed."
  - Why CAN we often get away with n-gram models?

- PCFG language models do model tree structure (later):
  - [This, quarter, 's, surprisingly, independent, attack, paid, off, the, risk, involving, IRS, leaders, and, transportation, prices, .]
  - [It, could, be, announced, sometime, .]
  - [Mr., Toseland, believes, the, average, defense, economy, is, drafted, from, slightly, more, than, 12, stocks, .]

Estimating bigram probabilities: The maximum likelihood estimate

- <s> I am Sam </s>
- <s> Sam I am </s>
- <s> I do not like green eggs and ham </s>

$P(\texttt{I} \mid \texttt{<s>}) = \frac{2}{3} = .67 \quad P(\texttt{Sam} \mid \texttt{<s>}) = \frac{1}{3} = .33 \quad P(\texttt{am} \mid \texttt{I}) = \frac{2}{3} = .67$

$P(\texttt{</s>} \mid \texttt{Sam}) = \frac{1}{2} = 0.5 \quad P(\texttt{Sam} \mid \texttt{am}) = \frac{1}{2} = .5 \quad P(\texttt{do} \mid \texttt{I}) = \frac{1}{3} = .33$

$$P(w_n|w_{n-N+1}^{n-1}) = \frac{C(w_{n-N+1}^{n-1}w_n)}{C(w_{n-N+1}^{n-1})}$$

- This is the Maximum Likelihood Estimate, because it is the one which maximizes P(Training set|Model)

Berkeley Restaurant Project sentences

- can you tell me about any good cantonese restaurants close by
- mid priced thai food is what i'm looking for
- tell me about chez panisse
- can you give me a listing of the kinds of food that are available
- i'm looking for a good place to eat breakfast
- when is caffe venezia open during the day

Raw bigram counts

- Out of 9222 sentences

	i	want	to	eat	chinese	food	lunch	spend
i	5	827	0	9	0	0	0	2
want	2	0	608	1	6	6	5	1
to	2	0	4	686	2	0	6	211
eat	0	0	2	0	16	2	42	0
chinese	1	0	0	0	0	82	1	0
food	15	0	15	0	1	4	0	0
lunch	2	0	0	0	0	1	0	0
spend	1	0	1	0	0	0	0	0

Raw bigram probabilities

- Normalize by unigrams:

i	want	to	eat	chinese	food	lunch	spend
2533	927	2417	746	158	1093	341	278

- Result:

	i	want	to	eat	chinese	food	lunch	spend
i	0.002	0.33	0	0.0036	0	0	0	0.00079
want	0.0022	0	0.66	0.0011	0.0065	0.0065	0.0054	0.0011
to	0.00083	0	0.0017	0.28	0.00083	0	0.0025	0.087
eat	0	0	0.0027	0	0.021	0.0027	0.056	0
chinese	0.0063	0	0	0	0	0.52	0.0063	0
food	0.014	0	0.014	0	0.00092	0.0037	0	0
lunch	0.0059	0	0	0	0	0.0029	0	0
spend	0.0036	0	0.0036	0	0	0	0	0

Evaluation

- What we want to know is:
  - Will our model prefer good sentences to bad ones?
    - That is, does it assign *higher probability* to “real” or “frequently observed” sentences than “ungrammatical” or “rarely observed” sentences?
  - As a component of Bayesian inference, will it help us discriminate correct utterances from noisy inputs?
- We train parameters of our model on a **training set**.
- To evaluate how well our model works, we look at the model’s performance on some new data
- This is what happens in the real world; we want to know how our model performs on data we haven’t seen
- So a **test set**. A dataset which is different from our training set. Preferably totally unseen/unused.

Measuring Model Quality

- For Speech: Word Error Rate (WER) $\frac{\text{insertions} + \text{deletions} + \text{substitutions}}{\text{true sentence size}}$

Correct answer: Andy saw a part of the movie

Recognizer output: And he saw apart of the movie

WER: 4/7
= 57%

- The "right" measure:
  - Task error driven
  - For speech recognition
  - For a specific recognizer!

- Extrinsic, task-based evaluation is in principle best, but …
- For general evaluation, we want a measure which references only good text, not mistake text

Measuring Model Quality

- The Shannon Game:
  - How well can we predict the next word?

    When I order pizza, I wipe off the ____

    Many children are allergic to ____

    I saw a ____

  - Unigrams are terrible at this game. (Why?)

grease	0.5
sauce	0.4
dust	0.05
….	
mice	0.0001
….	
the	1e-100

- The "Entropy" Measure
  - Really: average cross-entropy of a text according to a model

$$H(S \mid M) = \frac{\log_2 P_M(S)}{|S|} = \frac{\sum_i \log_2 P_M(s_i)}{\sum_i |s_i|}$$

$$\sum_j \log_2 P_M(w_j \mid w_{j-1})$$

Measuring Model Quality

- Problem with entropy:
  - 0.1 bits of improvement doesn’t sound so good
  - Solution: perplexity

$$P(S \mid M) = 2^{H(S|M)} = \sqrt[n]{\frac{1}{\prod_{i=1}^{n} P_M(w_i \mid h)}}$$

  - Intrinsic measure: may not reflect task performance (but is helpful as a first thing to measure and optimize on)

  - Note: Even though our models require a stop step, people typically don’t count it as a symbol when taking these averages.

  - E.g.,

N-gram Order	Unigram	Bigram	Trigram
Perplexity	962	170	109

The Shannon Visualization Method

- Generate random sentences:
- Choose a random bigram <s>, w according to its probability
- Now choose a random bigram (w, x) according to its probability
- And so on until we choose </s>
- Then string the words together
- <s> I

  I want

  want to

  to eat

  eat Chinese

  Chinese food

  food </s>

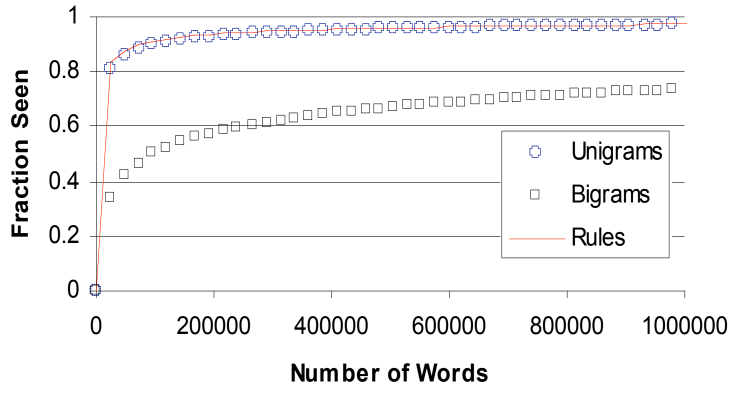
What's in our text corpora

- **Common words in *Tom Sawyer* (71,370 words)**
- **the: 3332, and: 2972, a: 1775, to: 1725, of: 1440, was: 1161, it: 1027, in: 906, that: 877, he: 877, I: 783, his: 772, you: 686, Tom: 679**

Word Frequency	Frequency of Frequency
1	3993
2	1292
3	664
4	410
5	243
6	199
7	172
8	131
9	82
10	91
11–50	540
51–100	99
>100	102

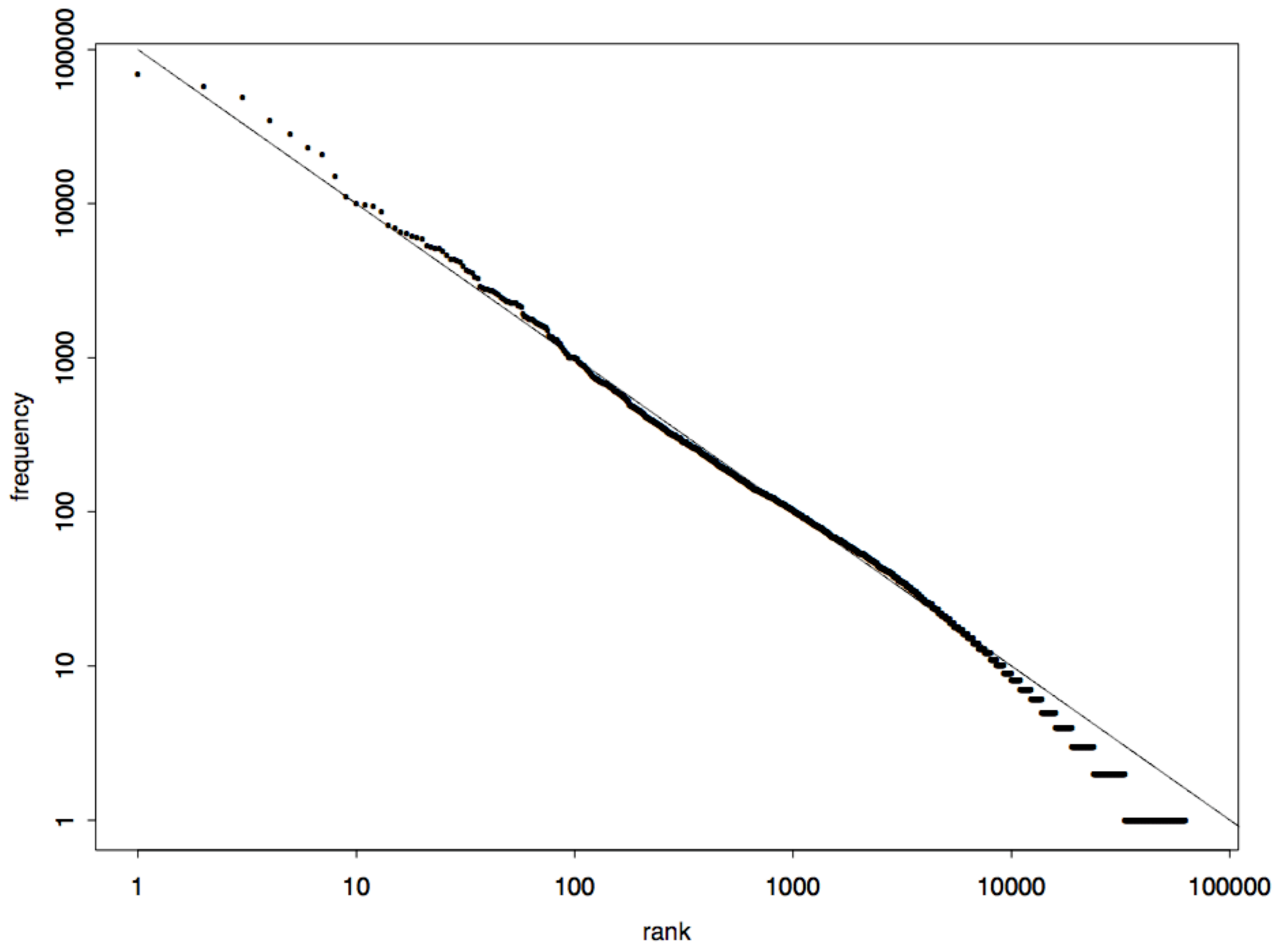
Sparsity

- Problems with n-gram models:
  - New words appear regularly:
    - Synaptitute
    - 132,701.03
    - fuzzificational
  - New bigrams: even more often
  - Trigrams or more – still worse!



- Zipf's Law
  - Types (words) vs. tokens (word occurences)
  - Broadly: most word types are rare ones
  - Specifically:
    - Rank word types by token frequency
    - Frequency inversely proportional to rank: $f = k/r$
    - Statistically: word distributions are *heavy tailed*
  - Not special to language: randomly generated character strings have this property (try it!)

Zipf's Law (on the Brown corpus)

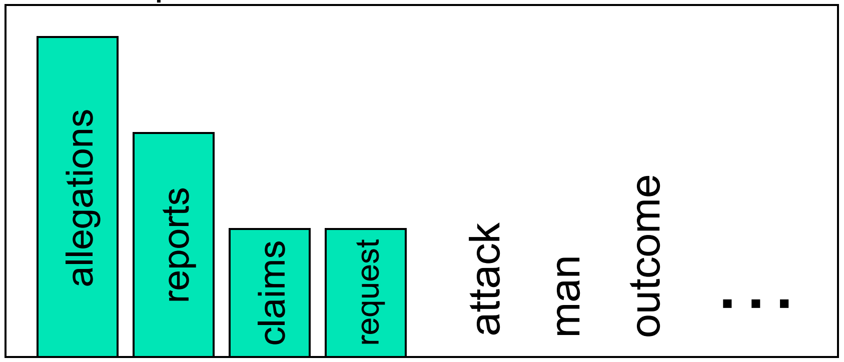


Smoothing

- We often want to make estimates from sparse statistics:

P(w | denied the)
3 allegations
2 reports
1 claims
1 request

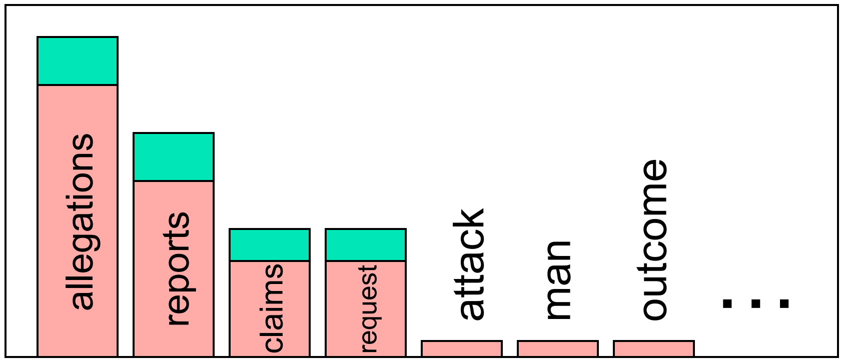
7 total



- Smoothing flattens spiky distributions so they generalize better

P(w | denied the)
2.5 allegations
1.5 reports
0.5 claims
0.5 request
2 other

7 total



- Very important all over NLP, but easy to do badly!
- Illustration with bigrams (h = previous word, could be anything).

Smoothing

- Estimating multinomials
  - We want to know what words follow some history h
  - There's some true distribution P(w | h)
  - We saw some small sample of N words from P(w | h)
  - We want to reconstruct a useful approximation of P(w | h)
  - Counts of events we didn't see are always too low (0 < N P(w | h))
  - Counts of events we did see are *in aggregate* to high
- Example:

P(w | denied the)	P(w | affirmed the)
3 allegations	1 award
2 reports	
1 claims	
1 speculation	
…	
1 request	
13 total	

- Two issues:
  - Discounting: how to reserve mass what we haven't seen
  - Interpolation: how to allocate that mass amongst unseen events

Five types of smoothing

- Today we'll cover
  - Add-δ smoothing (Laplace)
  - Simple interpolation
  - Good-Turing smoothing
  - Katz smoothing
  - Kneser-Ney smoothing
- Or less if we run out of time … and then you'll just have to read the textbook!

Smoothing: Add-One, Add-δ (for bigram models)

c	number of word tokens in training data
$c(w)$	count of word w in training data
$c(w_{-1},w)$	joint count of the w_{-1},w bigram
V	total vocabulary size (assumed known)
N_k	number of word types with count k

- One class of smoothing functions *(discounting):*
  - Add-one / delta:

$$P_{ADD-\delta}(w \mid w_{-1}) = \frac{c(w_{-1},w) + \delta(1/V)}{c(w_{-1}) + \delta}$$

  - In Bayesian statistics terms, this is equivalent to assuming a uniform Dirichlet prior

Add-One Estimation

- Idea: pretend we saw every word once more than we actually did [Laplace]

$$P(w \mid h) = \frac{c(w,h)+1}{c(h)+V}$$

  - Think of it as taking items with observed count r > 1 and treating them as having count r* < r
  - Holds out V/(N+V) for "fake" events
    - N_{1+}/N of which is distributed back to seen words
    - N_0/(N+V) actually passed on to unseen words (most of it!)
    - Actually tells us not only how much to hold out, but where to put it
  - Works astonishingly poorly in practice

- Quick fix: add some small δ instead of 1 [Lidstone, Jefferys]
  - Slightly better, holds out less mass, still a bad idea

Berkeley Restaurant Corpus: Laplace smoothed bigram counts

	i	want	to	eat	chinese	food	lunch	spend
i	6	828	1	10	1	1	1	3
want	3	1	609	2	7	7	6	2
to	3	1	5	687	3	1	7	212
eat	1	1	3	1	17	3	43	1
chinese	2	1	1	1	1	83	2	1
food	16	1	16	1	2	5	1	1
lunch	3	1	1	1	1	2	1	1
spend	2	1	2	1	1	1	1	1

Laplace-smoothed bigrams

$$P^*(w_n|w_{n-1}) = \frac{C(w_{n-1}w_n)+1}{C(w_{n-1})+V}$$

	i	want	to	eat	chinese	food	lunch	spend
i	0.0015	0.21	0.00025	0.0025	0.00025	0.00025	0.00025	0.00075
want	0.0013	0.00042	0.26	0.00084	0.0029	0.0029	0.0025	0.00084
to	0.00078	0.00026	0.0013	0.18	0.00078	0.00026	0.0018	0.055
eat	0.00046	0.00046	0.0014	0.00046	0.0078	0.0014	0.02	0.00046
chinese	0.0012	0.00062	0.00062	0.00062	0.00062	0.052	0.0012	0.00062
food	0.0063	0.00039	0.0063	0.00039	0.00079	0.002	0.00039	0.00039
lunch	0.0017	0.00056	0.00056	0.00056	0.00056	0.0011	0.00056	0.00056
spend	0.0012	0.00058	0.0012	0.00058	0.00058	0.00058	0.00058	0.00058

Reconstituted counts

$$c^*(w_{n-1}w_n) = \frac{[C(w_{n-1}w_n)+1] \times C(w_{n-1})}{C(w_{n-1})+V}$$

	i	want	to	eat	chinese	food	lunch	spend
i	3.8	527	0.64	6.4	0.64	0.64	0.64	1.9
want	1.2	0.39	238	0.78	2.7	2.7	2.3	0.78
to	1.9	0.63	3.1	430	1.9	0.63	4.4	133
eat	0.34	0.34	1	0.34	5.8	1	15	0.34
chinese	0.2	0.098	0.098	0.098	0.098	8.2	0.2	0.098
food	6.9	0.43	6.9	0.43	0.86	2.2	0.43	0.43
lunch	0.57	0.19	0.19	0.19	0.19	0.38	0.19	0.19
spend	0.32	0.16	0.32	0.16	0.16	0.16	0.16	0.16

Quiz Question!

- Suppose I'm making a language model with a vocabulary size of 20,000 words
- In my training data, I saw the bigram *comes across* 10 times
  - 5 times it was followed by *as*
  - 5 times it was followed by other words (*like, less, again, most, in*)

- What is the MLE of P(*as*|*comes across*)?
- What is the add-1 estimate of P(*as*|*comes across*)?

How Much Mass to Withhold?

- Remember the key discounting problem:
  - What count should *r** should we use for an event that occurred *r* times in N samples?
  - *r* is too big
- Idea: held-out data [Jelinek and Mercer]
  - Get another N samples
  - See what the average count of items occuring r times is (e.g. doubletons on average might occur 1.78 times)
  - Use those averages as *r**
- Works better than fixing counts to add in advance

Backoff and Interpolation

- *Discounting* says, “I saw event X *n* times, but I will really treat it as if I saw it fewer than *n* times
- *Backoff* (and *interpolation*) says, “In certain cases, I will condition on less of my context than in other cases”
  - The sensible thing is to condition on less in contexts that you haven’t learned much about

- **Backoff:** use trigram if you have it, otherwise bigram, otherwise unigram
- **Interpolation:** mix all three

Linear Interpolation

- One way to ease the sparsity problem for n-grams is to use less-sparse n-1-gram estimates
- General linear interpolation:

$$P(w \mid w_{-1}) = [1 - \lambda(w, w_{-1})]\hat{P}(w \mid w_{-1}) + [\lambda(w, w_{-1})]P(w)$$

  - Having a single global mixing constant is generally not ideal:

$$P(w \mid w_{-1}) = [1 - \lambda]\hat{P}(w \mid w_{-1}) + [\lambda]P(w)$$

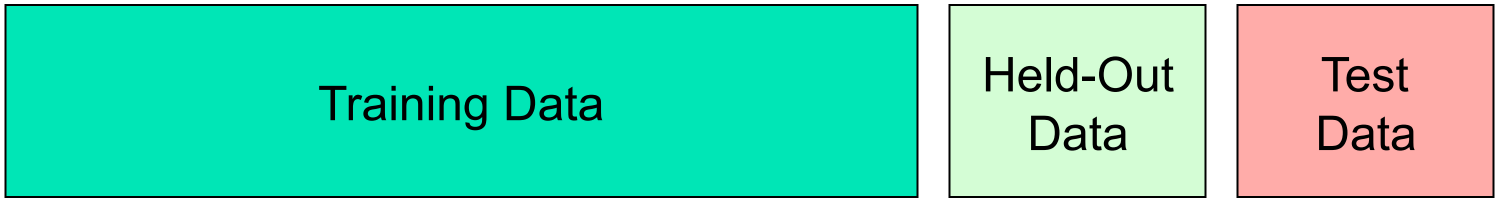
  - But it actually works surprisingly well – simplest competent approach

  - A better yet still simple alternative is to vary the mixing constant as a function of the conditioning context

$$P(w \mid w_{-1}) = [1 - \lambda(w_{-1})]\hat{P}(w \mid w_{-1}) + \lambda(w_{-1})P(w)$$

Held-Out Data

- Important tool for getting models to generalize:



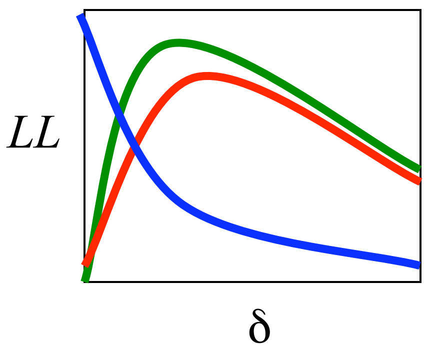
  - When we have a small number of parameters that control the degree of smoothing, we set them to maximize the (log-)likelihood of held-out data

$$LL(w_1 \ldots w_n \mid M(\lambda_1 \ldots \lambda_k)) = \sum_i \log P_{M(\lambda_1 \ldots \lambda_k)}(w_i \mid w_{i-1})$$

  - Can use any optimization technique (line search or EM usually easiest)

- Example:

$$P(w \mid w_{-1}) = [1-\lambda]\hat{P}(w \mid w_{-1}) + [\lambda]P(w)$$



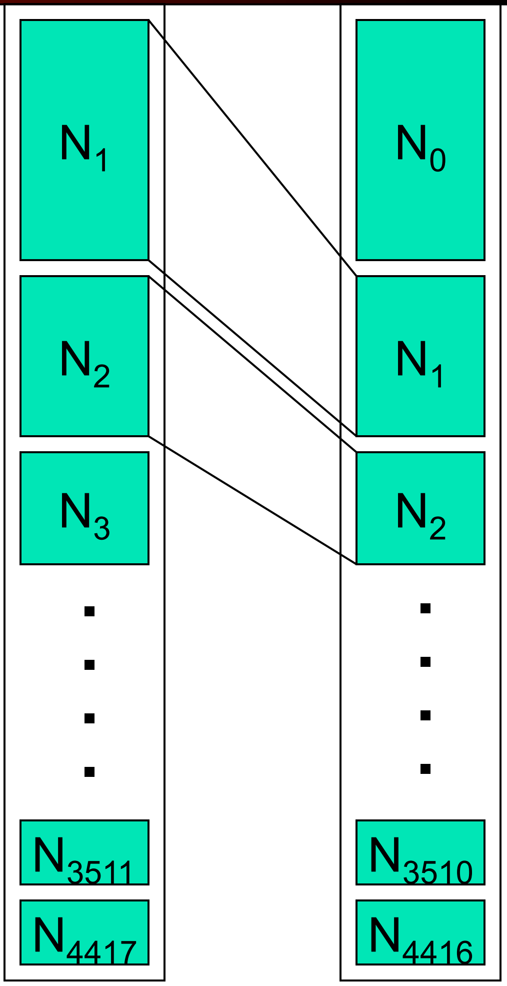
Good-Turing smoothing intuition

- Imagine you are fishing
- You have caught
  - 10 carp, 3 perch, 2 whitefish, 1 trout, 1 salmon, 1 eel = 18 fish
- How likely is it that next species is new (i.e. catfish or bass)
  - 3/18
- Assuming so, how likely is it that next species is trout?
  - Must be less than 1/18

[Slide adapted from Josh Goodman]

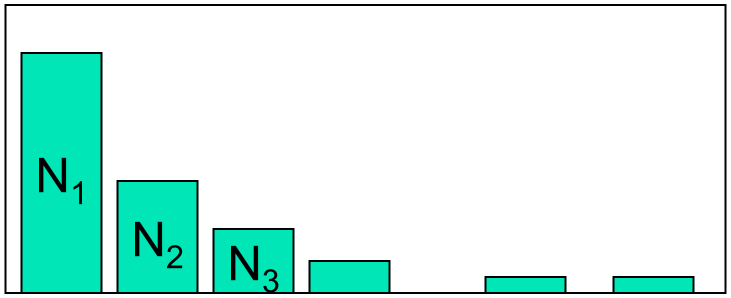
Good-Turing Reweighting I

- We'd like to not need held-out data (why?)
- Idea: leave-one-out validation
  - Take each of the c training words out in turn
  - c training sets of size c-1, held-out of size 1
  - What fraction of held-out words are unseen in training?
    - N_1/c
  - What fraction of held-out words are seen k times in training?
    - $(k+1)N_{k+1}/c$
  - So in the future we expect $(k+1)N_{k+1}/c$ of the words to be those with training count k
  - There are N_k words with training count k
  - Each should occur with probability:
    - $(k+1)N_{k+1}/c/N_k$
  - …or expected count $(k+1)N_{k+1}/N_k$

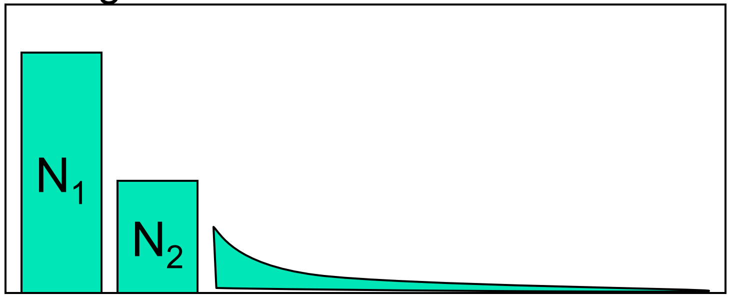


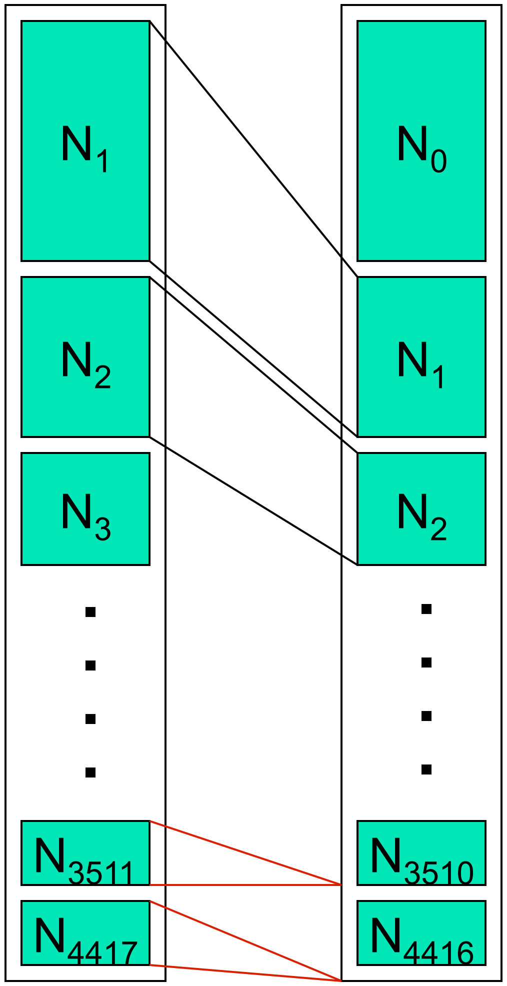
Good-Turing Reweighting II

- Problem: what about "the"? (say c=4417)
  - For small k, $N_k > N_{k+1}$
  - For large k, too jumpy, zeros wreck estimates



  - Simple Good-Turing [Gale and Sampson]: replace empirical N_k with a best-fit power law once count counts get unreliable





Good Turing calculations

	unseen (bass or catfish)	trout
c	0	1
MLE p	$p = \frac{0}{18} = 0$	$\frac{1}{18}$
c^*		$c^*(\text{trout}) = 2 \times \frac{N_2}{N_1} = 2 \times \frac{1}{3} = .67$
GT p^*_{GT}	$p^*_{\text{GT}}(\text{unseen}) = \frac{N_1}{N} = \frac{3}{18} = .17$	$p^*_{\text{GT}}(\text{trout}) = \frac{.67}{18} = \frac{1}{27} = .037$

Good-Turing Reweighting III

- Hypothesis: counts of k should be $k^* = (k+1)N_{k+1}/N_k$

Count in 22M Words	Actual c* (Next 22M)	GT's c*
1	0.448	0.446
2	1.25	1.26
3	2.24	2.24
4	3.23	3.24
Mass on New	9.2%	9.2%

- Katz Smoothing
  - Extends G-T smoothing into a backoff model from higher to lower order contexts
  - Use G-T discounted *bigram* counts (roughly – Katz left large counts alone)
  - Whatever mass is left goes to empirical unigram

$$P_{KATZ}(w \mid w_{-1}) = \frac{c^*(w, w_{-1})}{\sum_w c(w, w_{-1})} + \alpha(w_{-1})\hat{P}(w)$$

Intuition of Katz backoff + discounting

- How much probability to assign to all the zero trigrams?
  - Use GT or some other discounting algorithm to tell us
- How do we divide that probability mass among different words in the vocabulary?

  Use the (n – 1)-gram estimates to tell us
- What do we do for the unigram words not seen in training (i.e., not in our vocabulary
  - The problem of Out Of Vocabulary = OOV words
    - Important, but messy … mentioned at end of class

Kneser-Ney Smoothing I

- Something’s been very broken all this time
  - Shannon game: There was an unexpected ____?
    - delay?
    - Francisco?
  - “Francisco” is more common than “delay”
  - … but “Francisco” always follows “San”

- Solution: Kneser-Ney smoothing
  - In the back-off model, we don’t want the unigram probability of w
  - Instead, probability given that we are observing a novel continuation
  - Every bigram type was a novel continuation the first time it was seen

$$P_{CONTINUATION}(w) = \frac{|\{w_{-1} : c(w, w_{-1}) > 0\}|}{|(w, w_{-1}) : c(w, w_{-1}) > 0|}$$

Kneser-Ney Smoothing II

- One more aspect to Kneser-Ney:
  - Look at the GT counts:

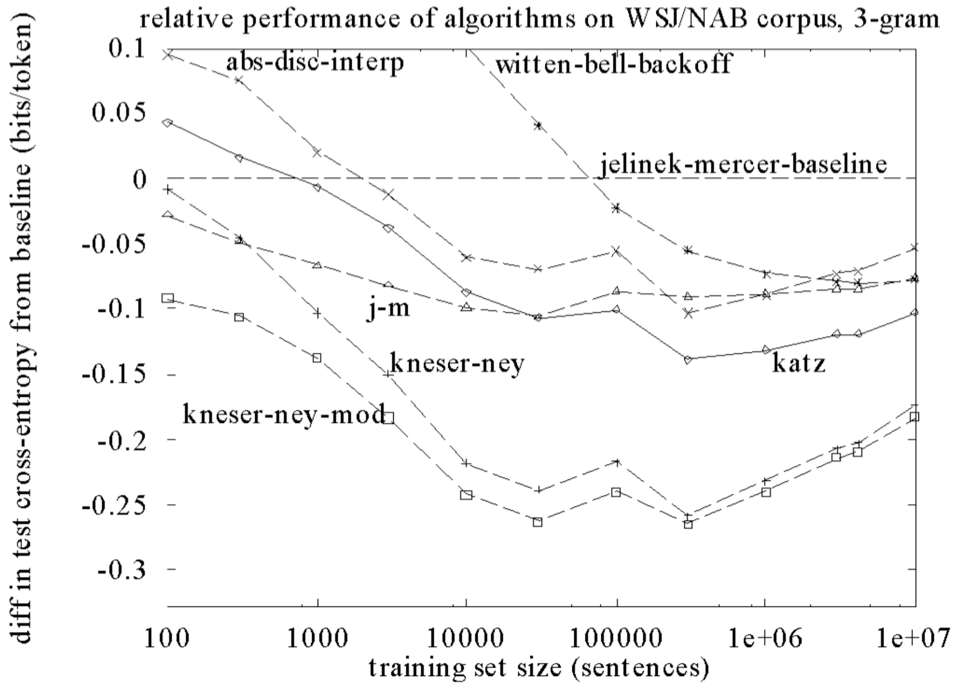
Count in 22M Words	Actual c* (Next 22M)	GT's c*
1	0.448	0.446
2	1.25	1.26
3	2.24	2.24
4	3.23	3.24

- Absolute Discounting
  - Save ourselves some time and just subtract 0.75 (or some d)
  - Maybe have a separate value of d for very low counts

$$P_{KN}(w \mid w_{-1}) = \frac{c(w, w_{-1}) - D}{\sum_{w'} c(w', w_{-1})} + \alpha(w_{-1}) P_{CONTINUATION}(w)$$

What Actually Works?

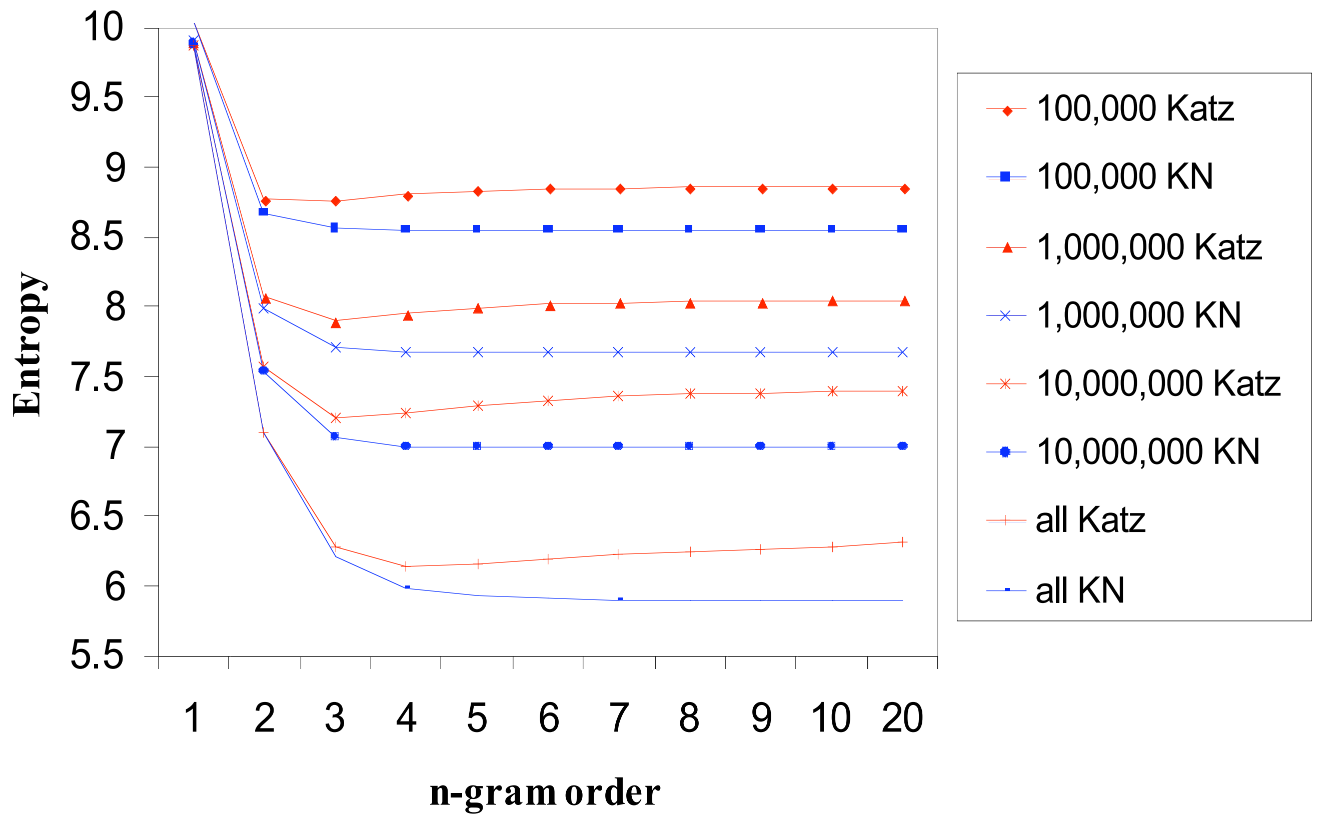
- Trigrams:
  - Unigrams, bigrams too little context
  - Trigrams much better (when there's enough data)
  - 4-, 5-grams usually not worth the cost (which is more than it seems, due to how speech recognizers are constructed)
- Good-Turing-like methods for count adjustment
  - Absolute discounting, Good-Turing, held-out estimation, Witten-Bell
- Kneser-Ney equalization for lower-order models
- See [Chen+Goodman] reading for tons of graphs!



[Graphs from Joshua Goodman]

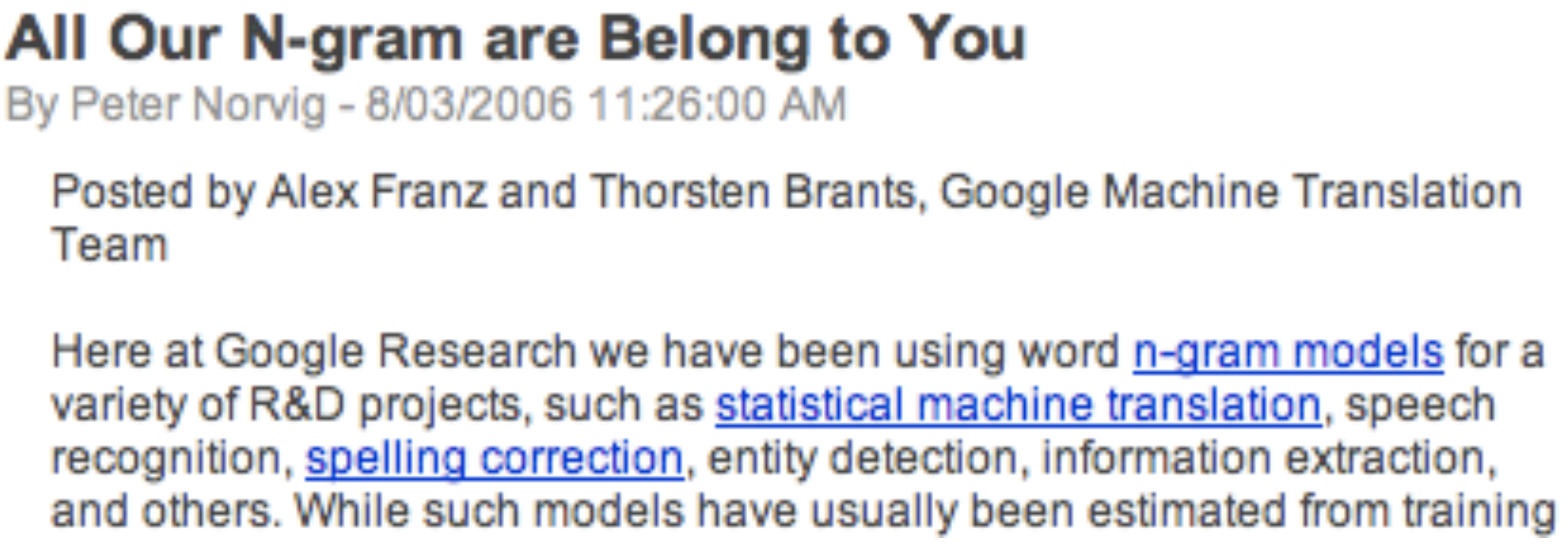
Data >> Method?

- Having more data is always good…



- … but so is picking a better smoothing mechanism!
- N > 3 often not worth the cost (greater than you'd think)

Google N-Gram Release

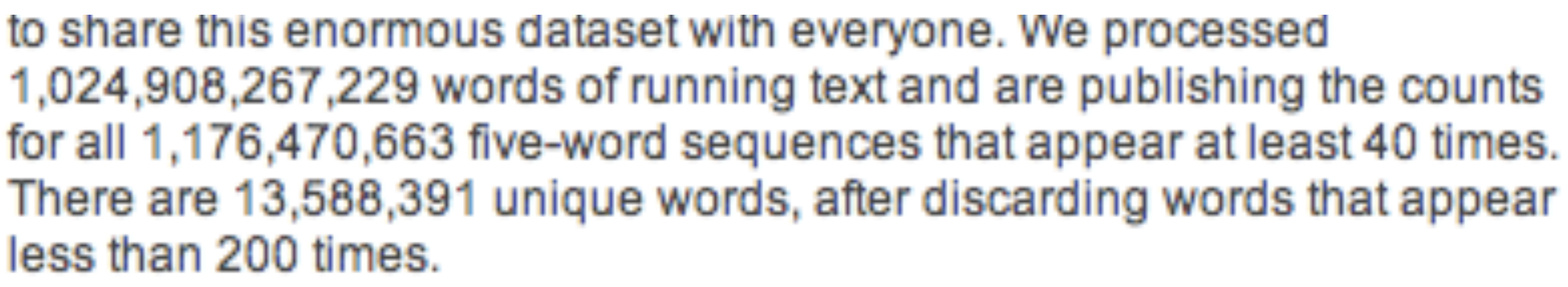


All Our N-gram are Belong to You

By Peter Norvig - 8/03/2006 11:26:00 AM

Posted by Alex Franz and Thorsten Brants, Google Machine Translation Team

Here at Google Research we have been using word n-gram models for a variety of R&D projects, such as statistical machine translation, speech recognition, spelling correction, entity detection, information extraction, and others. While such models have usually been estimated from training



to share this enormous dataset with everyone. We processed 1,024,908,267,229 words of running text and are publishing the counts for all 1,176,470,663 five-word sequences that appear at least 40 times. There are 13,588,391 unique words, after discarding words that appear less than 200 times.

Google N-Gram Release

- `serve as the incoming 92`
- `serve as the incubator 99`
- `serve as the independent 794`
- `serve as the index 223`
- `serve as the indication 72`
- `serve as the indicator 120`
- `serve as the indicators 45`
- `serve as the indispensable 111`
- `serve as the indispensible 40`
- `serve as the individual 234`

Beyond N-Gram LMs

- Caching Models
  - Recent words more likely to appear again

$$P_{CACHE}(w \mid history) = \lambda P(w \mid w_{-1} w_{-2}) + (1-\lambda)\frac{c(w \in history)}{|\,history\,|}$$

  - Can be disastrous in practice for speech (why?)

- Skipping Models

$$P_{SKIP}(w \mid w_{-1} w_{-2}) = \lambda_1 \hat{P}(w \mid w_{-1} w_{-2}) + \lambda_2 P(w \mid w_{-1} \,\_\_) + \lambda_3 P(w \mid \_\_\, w_{-2})$$

- Clustering Models: condition on word classes when words are too sparse
- Trigger Models: condition on bag of history words (e.g., maxent)
- Structured Models: use parse structure (we'll see these later)

- Language Modeling toolkits
  - SRILM
  - CMU-Cambridge LM Toolkit
  - IRST LM Toolkit

Unknown words: Open versus closed vocabulary tasks

- If we know all the words in advance
  - Vocabulary V is fixed
  - **Closed vocabulary task.** Easy
    - Common in speech recognition.
- Often we don’t know the set of all words
  - Out Of Vocabulary = OOV words
  - **Open vocabulary task**
- Instead: create an unknown word token <UNK>
  - Training of <UNK> probabilities
    - Create a fixed lexicon L of size V. [Can we work out right size for it??]
    - At text normalization phase, any training word not in L changed to <UNK>
    - There may be no such instance if L covers the training data
  - Now we train its probabilities
    - If low counts are mapped to <UNK>, we may train it like a normal word
    - Otherwise, techniques like Good-Turing estimation are applicable
  - At decoding time
    - If text input: Use UNK probabilities for any word not in training

Practical Considerations

- The unknown word symbol <UNK>
  - In many cases, open vocabularies use multiple types of OOVs (e.g., numbers & proper names)
  - For the programming assignment:
    - OK to assume there is only one unknown word type, UNK
    - UNK be quite common in new text!
    - UNK stands for all unknown word types (define probability event model thus – it's a union of basic outcomes)
  - To model the probability of individual new words occurring, you can use spelling models for them, but people usually don't
- Numerical computations
  - We usually do everything in log space (log probabilities)
  - Avoid underflow
  - (also adding is faster than multiplying)

$$p_1 \times p_2 \times p_3 \times p_4 = \exp(\log p_1 + \log p_2 + \log p_3 + \log p_4)$$