TTIC 31190: Natural Language Processing

Kevin Gimpel

Winter 2016

Lecture 6: Language Modeling

Announcements

- Assignment 1 due tonight
- Assignment 2 will be posted today, due Feb. 2
- Midterm scheduled for Thursday, Feb. 18
- Project proposal due Tuesday, Feb. 23
  - short (<1 page)
  - briefly describe project idea and plan (with timeline)
  - one proposal per group (groups can be size 1 or 2)

Distributional Word Vectors

- simplest way to create word vectors:
  count occurrences of context words

Counting Context Words

sugar, a sliced lemon, a tablespoonful of	**apricot**	preserve or jam, a pinch each of,
their enjoyment. Cautiously she sampled her first	**pineapple**	and another fruit whose taste she likened
well suited to programming on the digital	**computer**.	In finding the optimal R-stage policy from
for the purpose of gathering data and	**information**	necessary for the study authorized in the

	aardvark	computer	data	pinch	result	sugar	…
apricot	0	0	0	1	0	1	…
pineapple	0	0	0	1	0	1	…
digital	0	2	1	0	1	0	…
information	0	1	6	0	4	0	…
…							

Word-Context Matrix

- assume a **vocabulary** V and a **context vocabulary** V_C (V_C is a subset of V)
- build the **word-context matrix** C
  - C is a $|V|$-by-$|V_C|$ matrix of nonnegative counts
  - entry (i, j) contains the number of times context word j appeared within w words of word i in a corpus
- then build the **PMI matrix** P

Pointwise Mutual Information (PMI)

- do two events *x* and *y* co-occur more often than if they were independent?

$$\mathrm{pmi}(x;y) = \log \frac{p(x,y)}{p(x)p(y)}$$

- here, *x* is the center word and *y* is the word in the context window

- each probability can be estimated from counts collected from a corpus

Computing PMI

$$\mathrm{pmi}(i;j) = \log \frac{p(i,j)}{p(i)p(j)}$$

center word: index into vocabulary *V*

context word: index into context vocabulary V_C

we start with the word-context count matrix *C*:

C_{ij} = number of times context word *j* appears in window of word *i*

Computing PMI

$$\text{pmi}(i;j) = \log \frac{p(i,j)}{p(i)p(j)}$$

C_{ij} = number of times context word *j* appears in window of word *i*

estimate of joint probability: $p(i,j) = \dfrac{C_{ij}}{\sum_{i'=1}^{|V|} \sum_{j'=1}^{|V_C|} C_{i'j'}}$

estimates of center word and context word marginal probabilities:

$$p(i) = \frac{\sum_{j=1}^{|V_C|} C_{ij}}{\sum_{i'=1}^{|V|} \sum_{j'=1}^{|V_C|} C_{i'j'}}$$

$$p(j) = \frac{\sum_{i=1}^{|V|} C_{ij}}{\sum_{i'=1}^{|V|} \sum_{j'=1}^{|V_C|} C_{i'j'}}$$

same denominator for all terms

pmi(*hong, kong*) _____ pmi(*hong, then*)

< > = ?

pmi(*hong, kong*) **>** pmi(*hong, then*)

7.9 0.1

PMIs (1% of English Wikipedia, window size = 3)

word	context word	PMI
hong	kong	7.9
neither	nor	6.9
footballer	plays	6.0
1980s	1970s	5.3
musician	session	5.0
benefit	doubt	4.5
gain	failed	4.0
five	stars	3.5
miles	distance	3.0
prior	unlike	2.0
position	affairs	1.0
local	processes	0.5
fire	less	0.01

PMIs (1% of English Wikipedia, window size = 10)

word	context word	PMI
san	francisco	5.7
san	diego	5.7
san	juan	4.7
san	california	3.7
san	san	3.6
san	santa	3.3

word	context word	PMI
down	laid	3.8
down	shot	3.0
down	turned	2.9
down	broken	2.6
down	step	2.6
down	shooting	2.5

Evaluating word vectors

- extrinsic:
  - question answering, spell checking, essay grading
- intrinsic:
  - correlation between vector similarity and human word similarity judgments
    - WordSim353: 353 noun pairs rated 0-10

      sim(plane,car)=5.77
  - TOEFL multiple-choice vocabulary tests

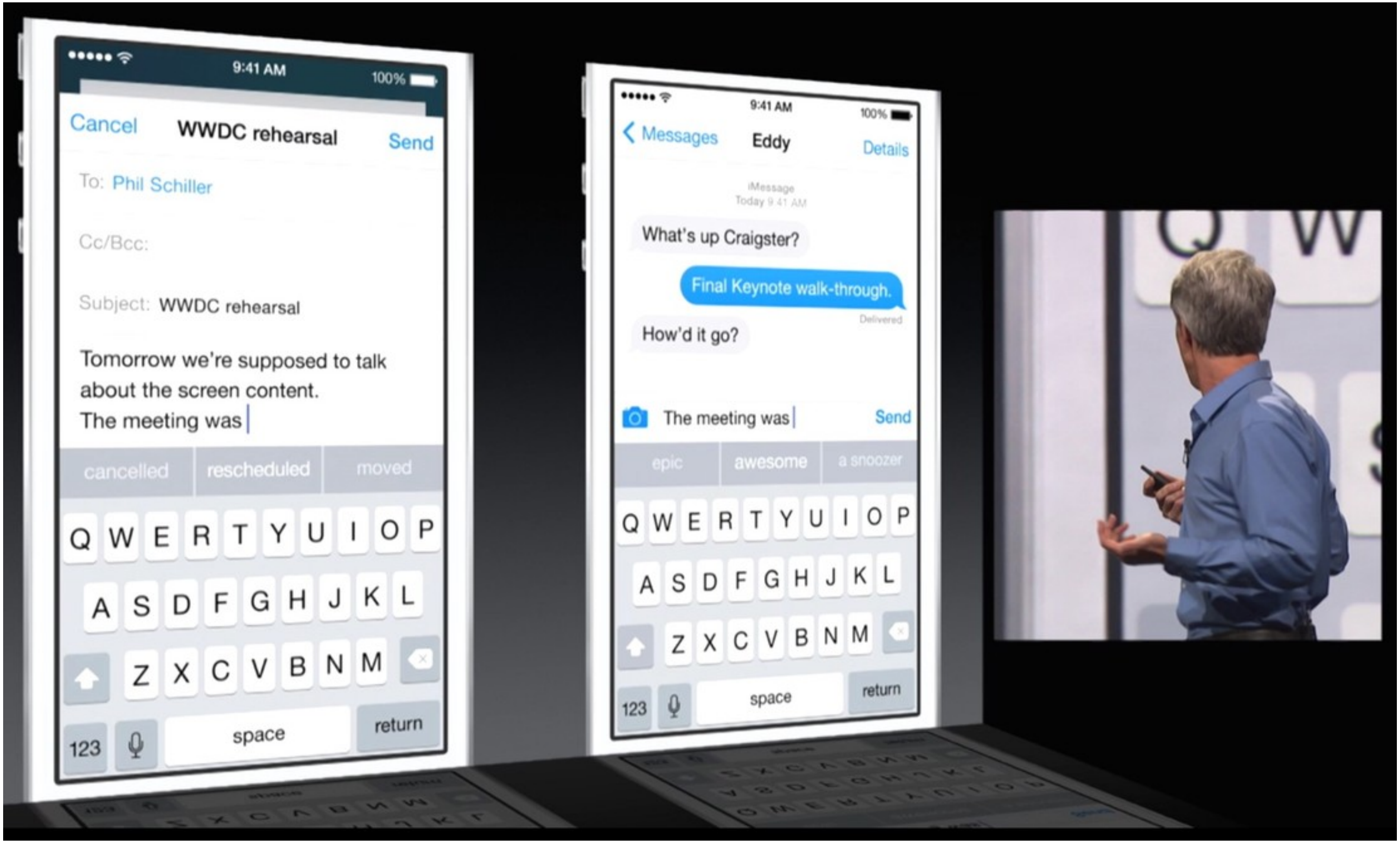
Roadmap

- classification
- words
- lexical semantics
- language modeling
- sequence labeling
- syntax and syntactic parsing
- neural network methods in NLP
- semantic compositionality
- semantic parsing
- unsupervised learning
- machine translation and other applications

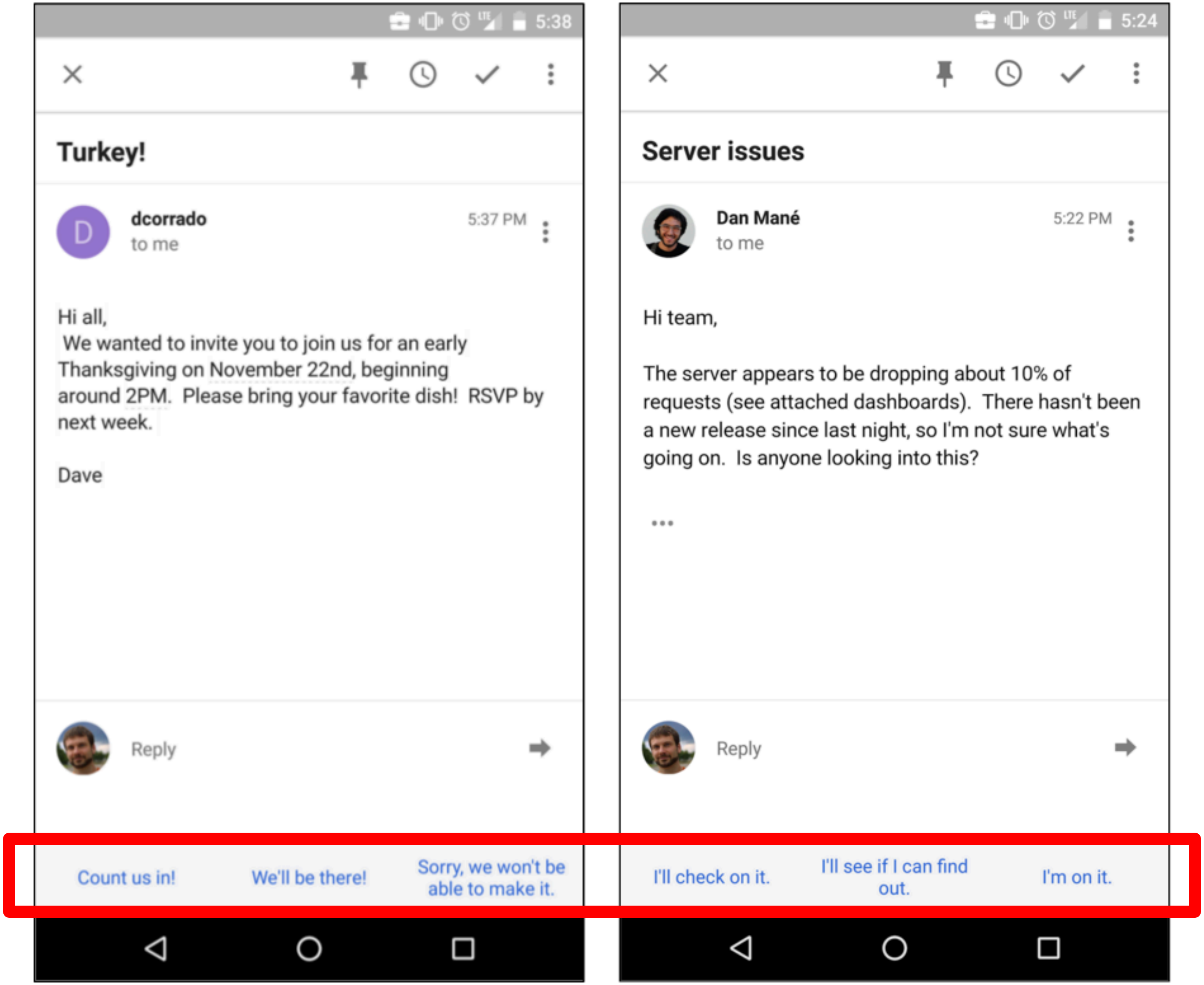
Probabilistic Language Models

- Today's goal: assign a probability to a sentence
- Why?
  - machine translation:
    - P(**high** winds tonite) > P(**large** winds tonite)
  - spelling correction:
    - The office is about fifteen **minuets** from my house
      - P(about fifteen **minutes** from) > P(about fifteen **minuets** from)
  - speech recognition:
    - P(I saw a van) >> P(eyes awe of an)
  - summarization, question answering, etc.!

Automatic Completion



Automatic Completion


5:38
Turkey!
dcorrado
to me
5:37 PM
Hi all,
We wanted to invite you to join us for an early Thanksgiving on November 22nd, beginning around 2PM. Please bring your favorite dish! RSVP by next week.
Dave
Reply
Count us in!
We'll be there!
Sorry, we won't be able to make it.
5:24
Server issues
Dan Mané
to me
5:22 PM
Hi team,
The server appears to be dropping about 10% of requests (see attached dashboards). There hasn't been a new release since last night, so I'm not sure what's going on. Is anyone looking into this?
...
Reply
I'll check on it.
I'll see if I can find out.
I'm on it.

Probabilistic Language Modeling

- goal: compute the probability of a sequence of words:

  $P(W) = P(w_1,w_2,w_3,w_4,w_5 \ldots w_n)$

- related task: probability of next word:

  $P(w_5|w_1,w_2,w_3,w_4)$

- a model that computes either of these:

  $P(W)$ or $P(w_n|w_1,w_2 \ldots w_{n-1})$

  is called a **language model (LM)**

How to compute P(W)

- How to compute this joint probability:
  - P(its, water, is, so, transparent, that)
- Intuition: let's rely on the Chain Rule of Probability

Reminder: Chain Rule

- recall definition of conditional probability:

P(B|A) = P(A,B)/P(A)   rewriting: **P(A,B) = P(A)P(B|A)**

- more variables:

P(A,B,C,D) = P(A)P(B|A)P(C|A,B)P(D|A,B,C)

- in general:

$$P(x_1,x_2,x_3,\ldots,x_n) = P(x_1)P(x_2|x_1)P(x_3|x_1,x_2)\ldots P(x_n|x_1,\ldots,x_{n-1})$$

Chain Rule applied to computing joint probability of words in sentence

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

P("*its water is so transparent*") =

P(*its*) × P(*water* | *its*) × P(*is* | *its water*)

× P(*so* | *its water is*) × P(*transparent* | *its water is so*)

J&M/SLP3

How to estimate these probabilities

- could we just count and divide?

$$P(\text{the} \mid \text{its water is so transparent that}) = \frac{Count(\text{its water is so transparent that the})}{Count(\text{its water is so transparent that})}$$

- no! too many possible sentences!
- we'll never see enough data for estimating these

Markov Assumption

Andrei Markov

- simplifying assumption:

$$P(\text{the} \mid \text{its water is so transparent that}) \approx P(\text{the} \mid \text{that})$$

- or maybe:

$$P(\text{the} \mid \text{its water is so transparent that}) \approx P(\text{the} \mid \text{transparent that})$$

Markov Assumption

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

- i.e., we approximate each component in the product:

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

Simplest case: Unigram model

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

automatically generated sentences from a unigram model:

```
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
```

J&M/SLP3

Bigram model

condition on the previous word:

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

automatically generated sentences from a bigram model:

```
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

this would be a record november
```

J&M/SLP3

n-gram models

- we can extend to trigrams, 4-grams, 5-grams
- in general this is an insufficient model of language
  - because language has **long-distance dependencies**:

"The computer which I had just put into the machine room on the fifth floor crashed."

- but we can often get away with n-gram models

Estimating bigram probabilities

- The Maximum Likelihood Estimate

$$P(w_i \mid w_{i-1}) = \frac{count(w_{i-1}, w_i)}{count(w_{i-1})}$$

$$P(w_i \mid w_{i-1}) = \frac{c(w_{i-1}, w_i)}{c(w_{i-1})}$$

An example

$$P(w_i \mid w_{i-1}) = \frac{c(w_{i-1}, w_i)}{c(w_{i-1})}$$

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

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

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

More examples: 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

- counts from 9,222 sentences
- e.g., "*i want*" occurs 827 times

	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

J&M/SLP3

Raw bigram probabilities

- normalize by unigram counts:

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

- bigram probabilities:

	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

J&M/SLP3

Bigram estimates of sentence probabilities

$P(\textit{<s> I want english food </s>}) =$

$P(I \mid \textit{<s>})$

$\times\ P(\textit{want} \mid I)$

$\times\ P(\textit{english} \mid \textit{want})$

$\times\ P(\textit{food} \mid \textit{english})$

$\times\ P(\textit{</s>} \mid \textit{food})$

$= .000031$

Practical Issues

- we do everything in log space
  - avoid underflow
  - (also adding is faster than multiplying)

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

Language Modeling Toolkits

- SRILM
  - http://www.speech.sri.com/projects/srilm/
- KenLM
  - https://kheafield.com/code/kenlm/

Google N-Gram Release, August 2006

AUG
3

All Our N-gram are Belong to You

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,

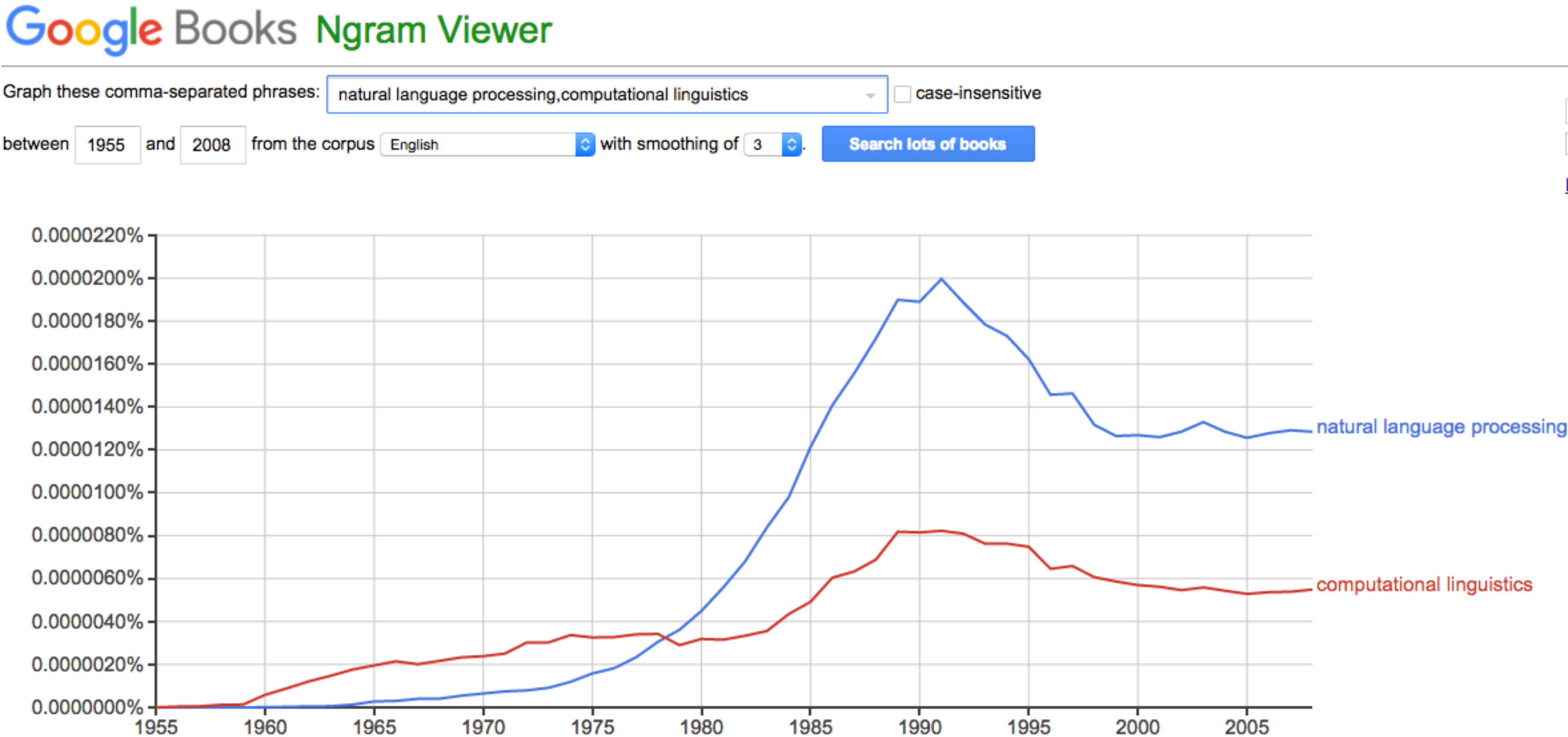
…

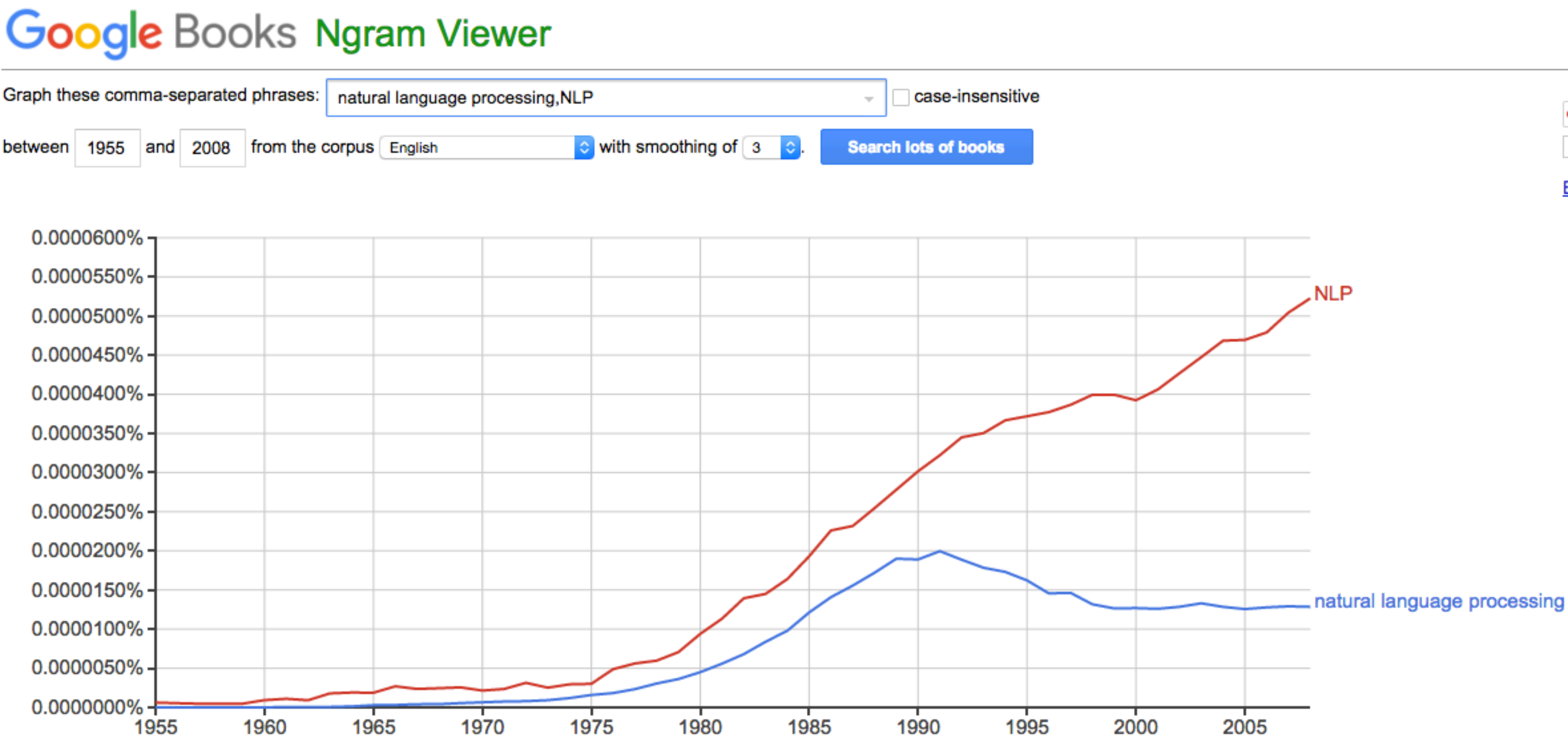
That's why we decided to share this enormous dataset with everyone. We processed 1,024,908,267,229 words of running text and are publishing the counts for all 1,176,470,663 five-word sequences that appear at least 40 times. There are 13,588,391 unique words, after discarding words that appear less than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

http://googleresearch.blogspot.com/2006/08/all-our-n-gram-are-belong-to-you.html


Google Books Ngram Viewer
Graph these comma-separated phrases:
natural language processing,computational linguistics
case-insensitive
between 1955 and 2008 from the corpus English with smoothing of 3.
Search lots of books
0.0000220%
0.0000200%
0.0000180%
0.0000160%
0.0000140%
0.0000120%
0.0000100%
0.0000080%
0.0000060%
0.0000040%
0.0000020%
0.0000000%
1955
1960
1965
1970
1975
1980
1985
1990
1995
2000
2005
natural language processing
computational linguistics



Google Books Ngram Viewer
Graph these comma-separated phrases:
natural language processing,NLP
case-insensitive
between
1955
and
2008
from the corpus
English
with smoothing of
3
Search lots of books
0.0000600%
0.0000550%
0.0000500%
0.0000450%
0.0000400%
0.0000350%
0.0000300%
0.0000250%
0.0000200%
0.0000150%
0.0000100%
0.0000050%
0.0000000%
1955
1960
1965
1970
1975
1980
1985
1990
1995
2000
2005
NLP
natural language processing

Evaluation: How good is our model?

- does our language model prefer good sentences to bad ones?
  - assign higher probability to “real” or “frequently observed” sentences
    - than “ungrammatical” or “rarely observed” sentences?

Extrinsic evaluation of N-gram models

- best evaluation for comparing models A and B
  - put each model in a task
    - spelling corrector, speech recognizer, MT system
  - run the task, get an accuracy for A and for B
    - how many misspelled words corrected properly
    - how many words translated correctly
  - compare accuracy for A and B

Difficulty of extrinsic evaluation of N-gram models

- extrinsic evaluation is time-consuming
  - days or weeks depending on system
- so, sometimes use intrinsic evaluation: **perplexity**
  - bad approximation
    - unless the test data looks **just** like the training data
    - so **generally only useful in pilot experiments**
  - but is helpful to think about

Intuition of Perplexity

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

I always order pizza with cheese and ____

The 33rd President of the US was ____

I saw a ____

mushrooms 0.1

pepperoni 0.1

anchovies 0.01

…

fried rice 0.0001

…

and 1e-100

  - unigrams are terrible at this game (why?)
- a better model of a text is one which assigns a higher probability to the word that actually occurs

Perplexity (PP)

best language model is one that best predicts unseen test set

- gives the highest P(sentence)

perplexity = inverse probability of test set, normalized by number of words:

$$PP(W) = P(w_1 w_2 \ldots w_N)^{-\frac{1}{N}}$$

$$= \sqrt[N]{\frac{1}{P(w_1 w_2 \ldots w_N)}}$$

chain rule:

$$\mathrm{PP}(W) = \sqrt[N]{\prod_{i=1}^{N} \frac{1}{P(w_i|w_1 \ldots w_{i-1})}}$$

for bigrams:

$$\mathrm{PP}(W) = \sqrt[N]{\prod_{i=1}^{N} \frac{1}{P(w_i|w_{i-1})}}$$

minimizing perplexity is the same as maximizing probability

Perplexity as branching factor

- given a sentence consisting of random digits
- what is the perplexity of this sentence according to a model that assigns probability 1/10 to each digit?

$$
\begin{aligned}
\mathrm{PP}(W) &= P(w_1 w_2 \ldots w_N)^{-\frac{1}{N}} \\
&= (\frac{1}{10}^N)^{-\frac{1}{N}} \\
&= \frac{1}{10}^{-1} \\
&= 10
\end{aligned}
$$

J&M/SLP3

Lower perplexity = better model

- train: 38 million words
- test: 1.5 million words

n-gram order:	unigram	bigram	trigram
perplexity:	962	170	109

Approximating Shakespeare

1 gram	–To him swallowed confess hear both. Which. Of save on trail for are ay device and rote life have –Hill he late speaks; or! a more to leg less first you enter
2 gram	–Why dost stand forth thy canopy, forsooth; he is this palpable hit the King Henry. Live king. Follow. –What means, sir. I confess she? then all sorts, he is trim, captain.
3 gram	–Fly, and will rid me these news of price. Therefore the sadness of parting, as they say, ’tis done. –This shall forbid it should be branded, if renown made it empty.
4 gram	–King Henry. What! I will go seek the traitor Gloucester. Exeunt some of the watch. A great banquet serv’d in; –It cannot be but so.

J&M/SLP3

Shakespeare as corpus

- 884,647 tokens, 29,066 types
- Shakespeare produced 300,000 bigram types out of 844 million possible bigrams
  - 99.96% of possible bigrams were never seen (have zero entries in the table)
- 4-grams worse: what's coming out looks like Shakespeare because it ***is*** Shakespeare

Wall Street Journal

1 gram	Months the my and issue of year foreign new exchange's september were recession exchange new endorsed a acquire to six executives
2 gram	Last December through the way to preserve the Hudson corporation N. B. E. C. Taylor would seem to complete the major central planners one point five percent of U. S. E. has already old M. X. corporation of living on information such as more frequently fishing to keep her
3 gram	They also point to ninety nine point six billion dollars from two hundred four oh six three percent of the rates of interest stores as Mexico and Brazil on market conditions

The perils of overfitting

- N-grams only work well for word prediction if the test corpus looks like the training corpus
  - in real life, it often doesn’t
  - we need to train robust models that generalize!
  - one kind of generalization: Zeros!
    - things that don’t ever occur in the training set
      - but occur in the test set

Zeros

training set:

... denied the allegations
... denied the reports
... denied the claims
... denied the request

P(*offer* | *denied the*) = 0

test set:

... denied the offer
... denied the loan

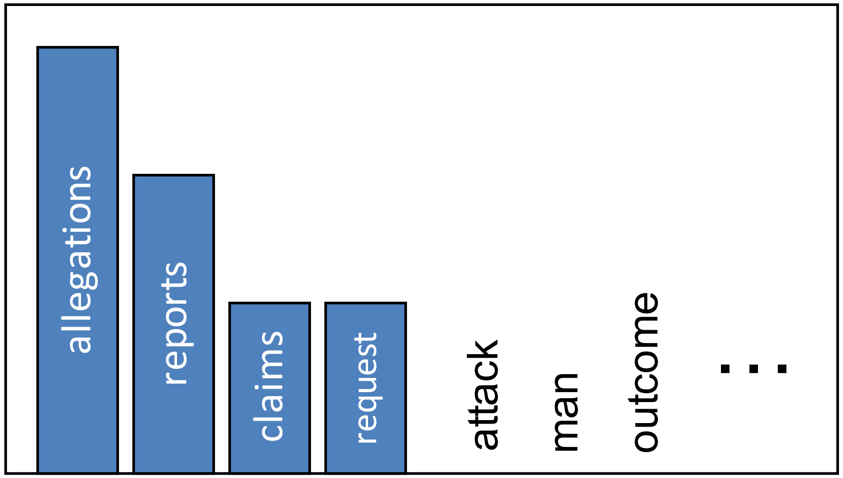
Zero probability bigrams

- test set bigrams with zero probability → assign 0 probability to entire test set!
- cannot compute perplexity (can't divide by 0)!

Intuition of smoothing (from Dan Klein)

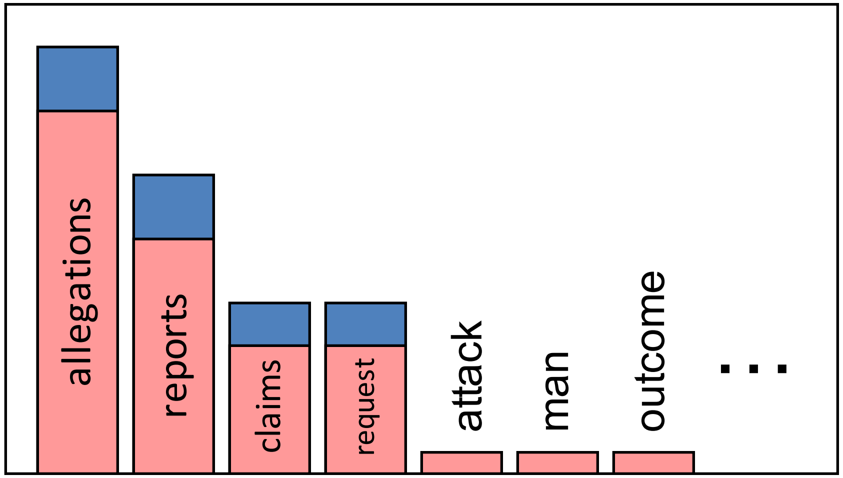
- When we have sparse statistics:

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



- Steal probability mass to generalize better:

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



“Add-1” estimation

- also called Laplace smoothing
- pretend we saw each word one more time than we did
- just add 1 to all counts!

- MLE estimate:

$$P_{MLE}(w_i \mid w_{i-1}) = \frac{c(w_{i-1}, w_i)}{c(w_{i-1})}$$

- Add-1 estimate:

$$P_{Add-1}(w_i \mid w_{i-1}) = \frac{c(w_{i-1}, w_i) + 1}{c(w_{i-1}) + V}$$

J&M/SLP3

Maximum Likelihood Estimates

- The maximum likelihood estimate
  - of some parameter of a model M from a training set T
  - maximizes the likelihood of the training set T given the model M
- Suppose the word “bagel” occurs 400 times in a corpus of a million words
- What is the probability that a random word from some other text will be “bagel”?
- MLE estimate is 400/1,000,000 = .0004
- This may be a bad estimate for some other corpus
  - But it is the **estimate** that makes it **most likely** that “bagel” will occur 400 times in a million word corpus.

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

J&M/SLP3

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

J&M/SLP3

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

Compare with raw bigram counts

	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

	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

Add-1 estimation is a blunt instrument

- so add-1 isn’t used for N-grams:
  - we’ll see better methods
- but add-1 is used to smooth other NLP models
  - text classification
  - domains where the number of zeros isn’t so huge

Backoff and Interpolation

- sometimes it helps to use **less** context
  - condition on less context for contexts you haven't learned much about
- **backoff**:
  - use trigram if you have good evidence, otherwise bigram, otherwise unigram
- **interpolation**:
  - mixture of unigram, bigram, trigram (etc.) models

- interpolation works better

Linear Interpolation

- simple interpolation:

$$\begin{aligned}\hat{P}(w_n|w_{n-2}w_{n-1}) &= \lambda_1 P(w_n|w_{n-2}w_{n-1}) \\ &+\lambda_2 P(w_n|w_{n-1}) \\ &+\lambda_3 P(w_n)\end{aligned} \qquad \sum_i \lambda_i = 1$$

- lambdas are functions of context:

$$\begin{aligned}\hat{P}(w_n|w_{n-2}w_{n-1}) &= \lambda_1(w_{n-2}^{n-1}) P(w_n|w_{n-2}w_{n-1}) \\ &+\lambda_2(w_{n-2}^{n-1}) P(w_n|w_{n-1}) \\ &+\lambda_3(w_{n-2}^{n-1}) P(w_n)\end{aligned}$$

J&M/SLP3

How to set the lambdas?

- use a **held-out** corpus:

Training Data | Held-Out Data | Test Data

- choose lambdas to maximize probability of held-out data:
  - fix N-gram probabilities (on the training data)
  - then search for λs that give largest probability to held-out set:

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

  - subtlety: what happens if we use training data to learn λs?

J&M/SLP3

Unknown words: open vs. closed vocabulary tasks

- if we know all the words in advance:
  - vocabulary *V* is fixed
  - “closed vocabulary” task
- often we don’t know this
  - **out-of-vocabulary (OOV)** words
  - “open vocabulary” task
- so, create an unknown word token <UNK>
  - at training time:
    - randomly change some instances of rare words to <UNK>
    - then estimate its probabilities like a normal word
  - at test time:
    - replace OOV words with <UNK>

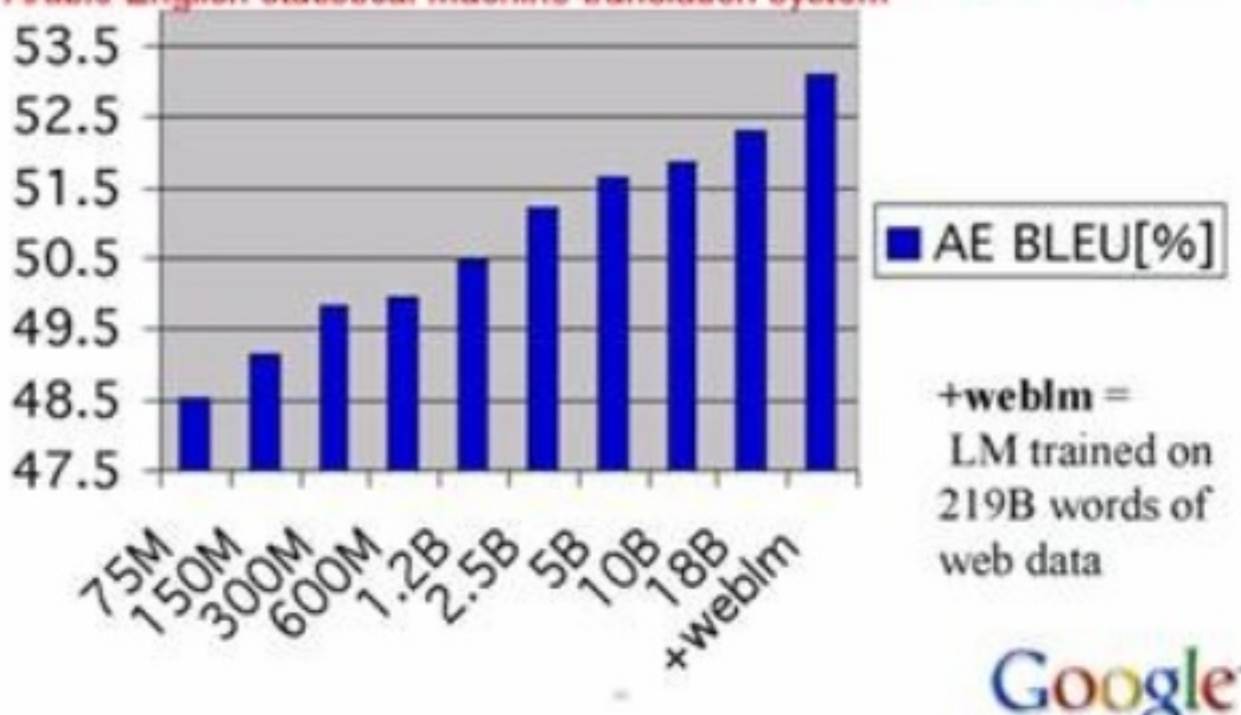
Huge web-scale n-grams

- how to deal with, e.g., Google N-gram corpus?
- pruning:
  - only store N-grams with count > threshold.
    - remove singletons of higher-order n-grams
  - entropy-based pruning
- efficiency
  - efficient data structures like tries
  - bloom filters: approximate language models
  - store words as indexes, not strings
    - use Huffman coding to fit large numbers of words into 2 bytes
  - quantize probabilities (4-8 bits instead of 8-byte float)

Google trillion word language model

More data is better data…

Impact on size of language model training data (in words) on quality of Arabic-English statistical machine translation system



Smoothing for Web-scale N-grams

- "Stupid backoff" (Brants et al., 2007)
- no discounting, just use relative frequencies

$$S(w_i \mid w_{i-k+1}^{i-1}) = \begin{cases} \dfrac{\text{count}(w_{i-k+1}^{i})}{\text{count}(w_{i-k+1}^{i-1})} & \text{if } \text{count}(w_{i-k+1}^{i}) > 0 \\ 0.4 S(w_i \mid w_{i-k+2}^{i-1}) & \text{otherwise} \end{cases}$$

$$S(w_i) = \frac{\text{count}(w_i)}{N}$$

N-gram Smoothing Summary

- Add-1 estimation:
  - OK for text categorization, not for language modeling
- most commonly used method:
  - modified interpolated Kneser-Ney
- for very large N-gram collections like the Web:
  - stupid backoff

Advanced Language Modeling

- discriminative models:
  - choose n-gram weights to improve a task, not to fit the training set
- syntactic language models
- caching models
  - recently used words are more likely to appear

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

  - these perform very poorly for speech recognition (why?)