

Language Modeling

Instructor: Wei Xu

Many slides adapted from Dan Jurafsky, and some from Yejin Choi

Language Modeling

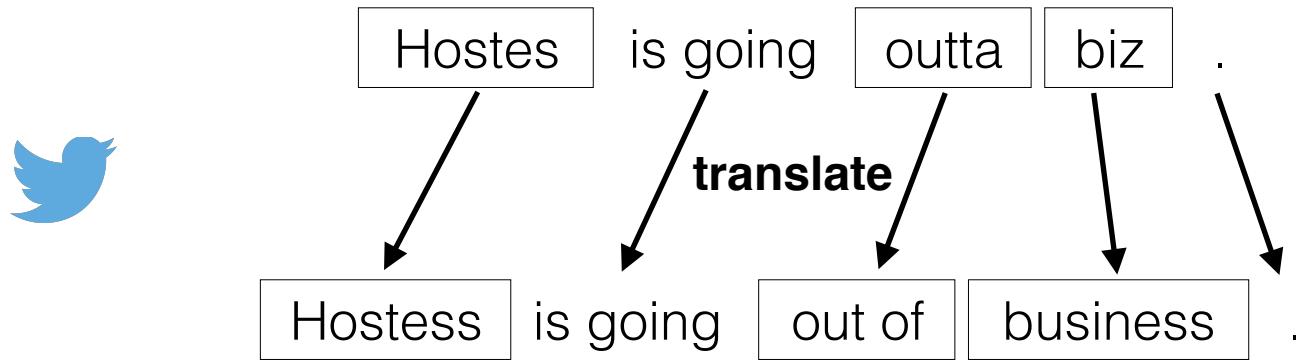
Introduction to N-grams

Probabilistic Language Models

- Today's goal: assign a probability to a sentence
 - Machine Translation:
 - $P(\text{high winds tonite}) > P(\text{large winds tonite})$
 - Spell Correction
 - The office is about fifteen **minuets** from my house
 - $P(\text{about fifteen minutes from}) > P(\text{about fifteen minuets from})$
 - Speech Recognition
 - $P(\text{I saw a van}) \gg P(\text{eyes awe of an})$
 - + Summarization, question-answering, etc., etc.!!

Why?

Noisy Text Normalization (Error Correction)



Wei Xu, Joel Tetreault, Martin Chodorow, Ralph Grishman, Le Zhao.

“Exploiting Syntactic and Distributional Information for Spelling Correction with Web-Scale N-gram Models” In EMNLP (2011)

Wei Xu, Alan Ritter, Ralph Grishman. “Gathering and Generating Paraphrases from Twitter with Application to Normalization” In BUCC (2013)

Timothy Baldwin, Marie-Catherine de Marneffe, Bo Han, Young-Bum Kim, Alan Ritter, Wei Xu. “Shared Tasks of the 2015 Workshop on Noisy User-generated Text: Twitter Lexical Normalization and Named Entity Recognition” In WNUT (2015)

Natural Language Generation (Monolingual MT)



Palpatine:

If you will not be turned, you will be destroyed!



If you will not be turn'd, you will be undone!

Luke:

Father, please! Help me!



Father, I pray you! Help me!



Wei Xu, Alan Ritter, Bill Dolan, Ralph Grishman, Colin Cherry. "Paraphrasing for Style" In COLING (2012)

Quanze Chen, Chenyang Lei, Wei Xu, Ellie Pavlick, Chris Callison-Burch.

"Poetry of the Crowd: A Human Computation Algorithm to Convert Prose into Rhyming Verse" In HCOMP (2014)



Poetry of the Crowd : A Human Computation Algorithm to Convert Prose into Rhyming Verse

Quanze Chen, Chenyang Lei, Wei Xu, Ellie Pavlick and Chris Callison-Burch

The Problem

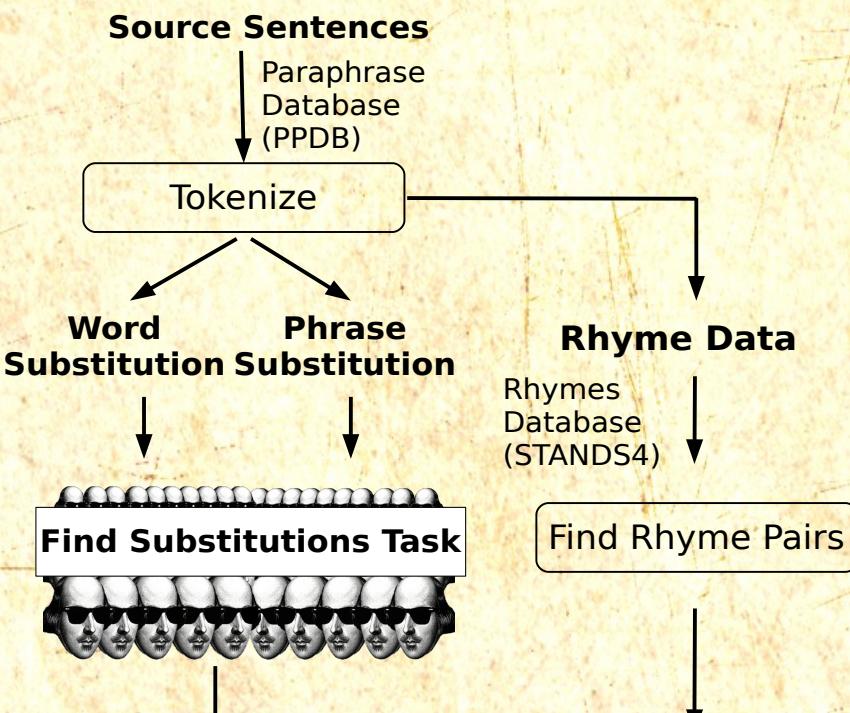
Is it possible to efficiently utilize crowdsourcing to generate poetry?

Abstract

Poetry composition is a very complex task that requires a poet to satisfy multiple constraints concurrently. We believe that the task can be augmented by combining the creative abilities of humans with computational algorithms that efficiently constrain and permute available choices. We present a hybrid method for generating poetry from prose that combines crowdsourcing with natural language processing (NLP) machinery. We test the ability of crowd workers to accomplish the technically challenging and creative task of writing sonnets.

Substitution:

- Figure out a set of possible substitutions to use in target



Rhyme:

Challenge: It's hard to find rhymes across multiple sentences.

Solution:

- Collect rhyme information for text or expanded set that covers many sentences
- Find the intersection of the potential rhyming pairs

Preliminary Results

- 179 rhyming pairs from 1000 sentences
- 911 rhyming pairs after filtering

Meter:

Challenge: It's difficult to find a large set of choice words that have the right meter.

Solution:

- Obtain the stress patterns from pronunciation dictionary
- Label predicted meter patterns trained on sonnet data.

Probabilistic Language Modeling

- Goal: compute the probability of a sentence or sequence of words:

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

- Related task: probability of an upcoming word:

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

- A model that computes either of these:

$P(W)$ or $P(w_n | w_1, w_2 \dots w_{n-1})$ is called a **language model** or **LM**

How to compute $P(W)$

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

Reminder: The Chain Rule

- Recall the definition of conditional probabilities

$$P(B|A) = P(A,B)/P(A) \quad \text{Rewriting: } P(A,B) = P(A)P(B|A)$$

- More variables:

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

- The Chain Rule in General

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

The Chain Rule applied to compute joint probability of words in sentence

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

$P(\text{"its water is so transparent"}) =$

$P(\text{its}) \times P(\text{water}|\text{its}) \times P(\text{is}|\text{its water})$

$\times P(\text{so}|\text{its water is}) \times P(\text{transparent}|\text{its water is so})$

How to estimate these probabilities

- Could we just count and divide?

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

- No! Too many possible sentences!
- We'll never see enough data for estimating these

Markov Assumption



- Simplifying assumption:

Andrei Markov (1856~1922)

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

- Or maybe

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

Markov Assumption

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

- In other words, we approximate each component in the product

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

Simplest case: Unigram model

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

Some 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

words are independent

Big problem with unigrams: $P(\text{the the the the}) >> P(\text{I like ice cream})$

Simplest case: Unigram model

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

Some 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

Bigram model

- Condition on the previous word:

$$P(w_i | w_1 w_2 \dots w_{i-1}) \approx P(w_i | w_{i-1})$$

Some 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

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

Language Modeling

Estimating N-gram Probabilities

Estimating bigram probabilities

- The Maximum Likelihood Estimate (MLE)
 - relative frequency based on the empirical counts on a training set

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

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

c – count

An example

$$P(w_i | w_{i-1}) = \frac{c(w_{i-1}, w_i)}{c(w_{i-1})} \quad \begin{array}{l} <\text{s}> \text{ I am Sam } </\text{s}> \\ <\text{s}> \text{ Sam I am } </\text{s}> \\ <\text{s}> \text{ I do not like green eggs and ham } </\text{s}> \end{array}$$

$$P(\text{I} | <\text{s}>) = \frac{2}{3} = .67$$

$$P(</\text{s}> | \text{Sam}) = \frac{1}{2} = 0.5$$

$$P(\text{Sam} | <\text{s}>) = \frac{1}{3} = .33$$

$$P(\text{Sam} | \text{am}) = \frac{1}{2} = .5$$

$$P(\text{am} | \text{I}) = \frac{2}{3} = .67$$

$$P(\text{do} | \text{I}) = \frac{1}{3} = .33$$

A bigger example: 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

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

- Normalize by unigrams:

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

	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

Bigram estimates of sentence probabilities

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

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

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

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

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

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

$$= .000031$$

What kinds of knowledge?

- $P(\text{english} \mid \text{want}) = .0011$ about the world
- $P(\text{chinese} \mid \text{want}) = .0065$
- $P(\text{to} \mid \text{want}) = .66$ grammar – infinitive verb
- $P(\text{eat} \mid \text{to}) = .28$
- $P(\text{food} \mid \text{to}) = 0$???
- $P(\text{want} \mid \text{spend}) = 0$ grammar
- $P(\text{i} \mid \langle s \rangle) = .25$

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.

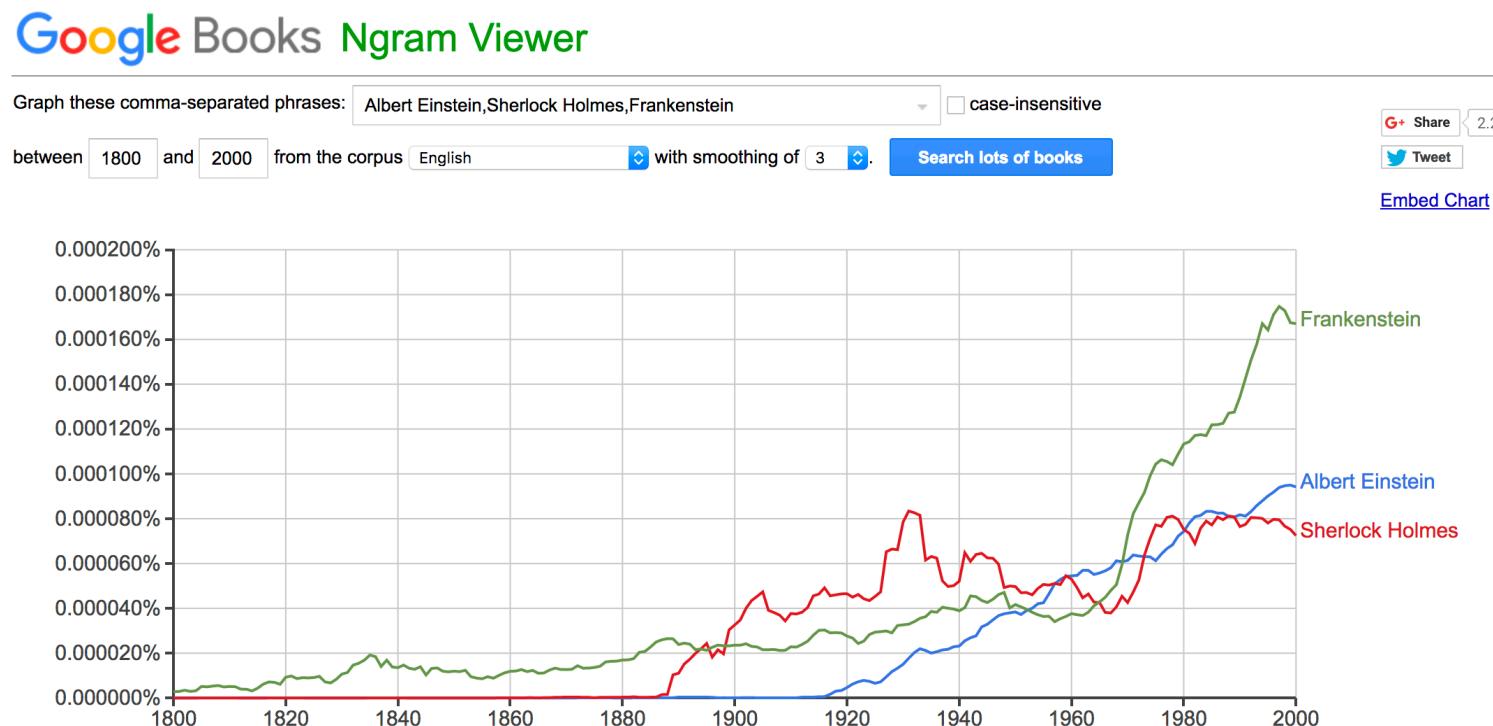
Google N-Gram Release

- 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 indispensable 40
- serve as the individual 234

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

Google Book N-grams

- <https://books.google.com/ngrams>
- <http://storage.googleapis.com/books/ngrams/books/datasetsv2.html>



Language Modeling

Evaluation and Perplexity

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?
- We train parameters of our model on a **training set**.
- We test the model’s performance on data we haven’t seen.
 - A **test set** is an unseen dataset that is different from our training set, totally unused.
 - An **evaluation metric** tells us how well our model does on the test set.

Evaluation: How good is our model?

- The goal isn't to pound out fake sentences!
 - Obviously, generated sentences get “better” as we increase the model order
 - More precisely: using maximum likelihood estimators, higher order is always better likelihood on **training set**, but not **test set**

Training on the test set

- We can't allow test sentences into the training set
- We will assign it an artificially high probability when we set it in the test set
- “Training on the test set”
 - Bad science!
 - And violates the honor code

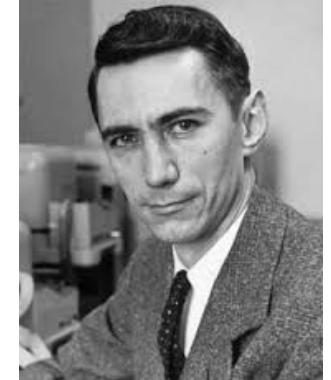
Extrinsic evaluation of N-gram models

- 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 (in-vivo) evaluation of N-gram models

- Extrinsic evaluation
 - Time-consuming; can take days or weeks
- 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 _____
 - 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
 - compute per word log likelihood
(M words, m test sentence s_i)

mushrooms 0.1
pepperoni 0.1
anchovies 0.01
....
fried rice 0.0001
....
and 1e-100

Claude Shannon
(1916~2001)

$$l = \frac{1}{M} \sum_{i=1}^m \log p(s_i)$$

Perplexity

The best language model is one that best predicts an unseen test set

- Gives the highest $P(\text{sentence})$

Perplexity is the inverse probability of the test set, “normalized” by the number of word (Why ?)

$$\begin{aligned} PP(W) &= P(w_1 w_2 \dots w_N)^{-\frac{1}{N}} \\ &= \sqrt[N]{\frac{1}{P(w_1 w_2 \dots w_N)}} \end{aligned}$$

geometric mean

equivalently:

$$PP(W) = 2^{-l}$$

$$\text{where } l = \frac{1}{N} \log P(w_1 w_2 \dots w_N)$$

$$2^{-l} \text{ where } l = \frac{1}{M} \sum_{i=1}^m \log p(s_i)$$

Minimizing perplexity is the same as maximizing probability

Perplexity

The best language model is one that best predicts an unseen test set

- Gives the highest $P(\text{sentence})$

Perplexity is the inverse probability of the test set, “normalized” by the number of words:

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

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

Chain Rule

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

for bigram

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

Minimizing perplexity is the same as maximizing probability

Lower perplexity = better model

- Training 38 million words, test 1.5 million words, Wall Street Journal

N-gram Order	Unigram	Bigram	Trigram
Perplexity	962	170	109

Perplexity as branching factor

- Let's suppose a sentence consisting of random digits [0,1, ..., 9]
- What is the perplexity of this sentence according to a model that assign $P=1/10$ to each digit?

$$\begin{aligned} \text{PP}(W) &= P(w_1 w_2 \dots w_N)^{-\frac{1}{N}} \\ &= \left(\frac{1}{10}\right)^N^{-\frac{1}{N}} \\ &= \frac{1}{10}^{-1} \\ &= 10 \end{aligned}$$

Perplexity as branching factor

- If one could report a model perplexity of 247 ($2^{7.95}$) per word
- In other words, the model is as confused on test data as if it had to choose uniformly and independently among 247 possibilities for each word.
- But,
 - a trigram language model can get perplexity of 247 on the Brown Corpus
 - simply guessing that the next word in the Brown Corpus is 7%, not 1/247 (Q: why??)

Language Modeling

Generalization and
zeros

The Shannon Visualization Method

- 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 $< /s >$

I want to eat Chinese food

Q: How do you write a program and do it?

Approximating Shakespeare

1
gram

–To him swallowed confess hear both. Which. Of save on trail for are ay device and
rote life have

2
gram

–Hill he late speaks; or! a more to leg less first you enter
–Why dost stand forth thy canopy, forsooth; he is this palpable hit the King Henry. Live
king. Follow.

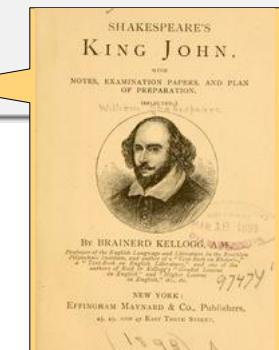
3
gram

–Fly, and will rid me these news of price. Therefore the sadness of parting, as they say,
'tis done.

4
gram

–This shall forbid it should be branded, if renown made it empty.
–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.



Shakespeare as corpus

- $N=884,647$ tokens, $V=29,066$
- Shakespeare produced 300,000 bigram types out of $V^2 = 844$ million possible bigrams.
 - So 99.96% of the possible bigrams were never seen (have zero entries in the table)
- Quadrigrams (4-gram) worse:
 - What's coming out looks like Shakespeare because it *is* Shakespeare

The wall street journal is not Shakespeare (no offense)

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
- Test set
 - ... denied the offer
 - ... denied the loan

$$P(\text{"offer"} \mid \text{denied the}) = 0$$

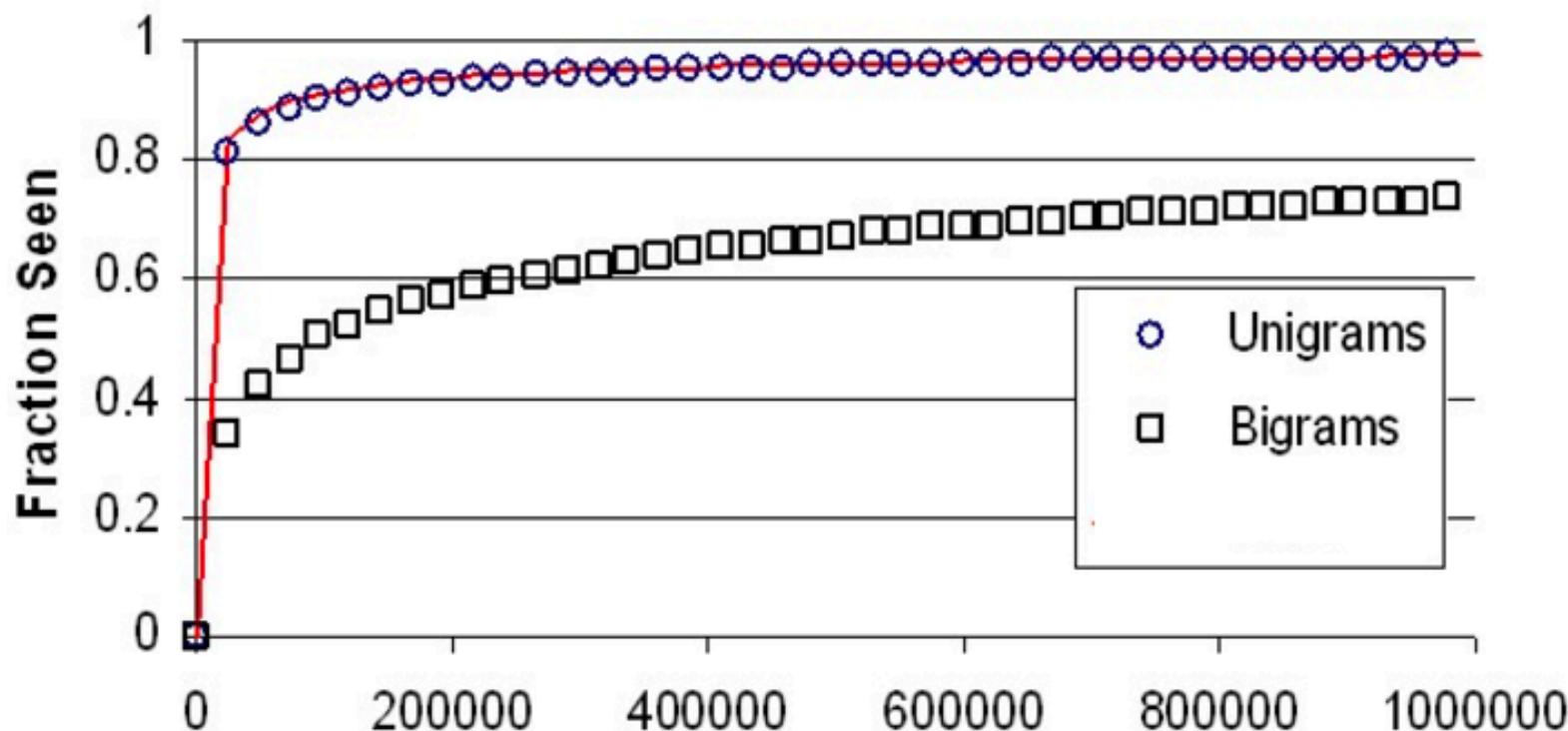
(Recall) Raw bigram counts

- 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

Sparsity

- human language is very creative
- new words appear all the time



Zero probability bigrams

- Bigrams with zero probability
 - mean that we will assign 0 probability to the test set!
- And hence we cannot compute perplexity (can't divide by 0)!

$$\begin{aligned} PP(W) &= P(w_1 w_2 \dots w_N)^{-\frac{1}{N}} \\ &= \sqrt[N]{\frac{1}{P(w_1 w_2 \dots w_N)}} \end{aligned}$$

for bigram

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

Q: How do we deal with ngrams of zero probabilities?