Modelling Text

Language Models

Today

- Word prediction task
- Language modeling (N-grams)
    - N-gram intro
    - Model evaluation
    - Smoothing

Word Prediction

- Guess the next word:
    - *... I notice three guys standing on the ???*
- There are many sources of knowledge that can be used to inform this task, including arbitrary world knowledge.
- But it turns out that you can do pretty well by simply looking at the preceding words and keeping track of some simple counts.

Word Prediction

- We can formalize this task with so-called N-gram models.
- N-grams are token sequences of length N.
- Our earlier example contains the following 2-grams (aka bigrams):
  - *... I notice three guys standing on the ???*

  → (I notice), (notice three), (three guys), (guys standing), (standing on), (on the)
- Given knowledge of counts of N-grams such as these, we can guess likely next words in a sequence.

N-Gram Models

- More formally, we can use knowledge of the counts of *N*-grams to assess the conditional probability of candidate words as the next word in a sequence
- Or we can use them to assess the probability of an entire sequence of words
    - Pretty much the same thing as we'll see...

Applications

- It turns out that being able to predict the next word (or any linguistic unit) in a sequence is an extremely useful thing to be able to do
- As we'll see, it lies at the core of the following applications
    - Automatic speech recognition
    - Handwriting and character recognition
    - Spelling correction
    - Machine translation
    - And many more

Counting: Corpora

- So, what happens when we look at large bodies of text instead of single utterances?

- Brown et al. (1992) large corpus of English text
  - 583 million wordform tokens
  - 293,181 wordform types

- Google
  - Crawl of 1,024,908,267,229 English tokens
  - 13,588,391 wordform types
    - That seems like a lot of types... After all, even large dictionaries of English have only around 500k types. Why so many here?

- Numbers
- Misspellings
- Names
- Acronyms
- etc

Language Modeling

- Back to word prediction
- We can model the word prediction task as the ability to assess the conditional probability of a word given the previous words in the sequence

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

- We'll call a statistical model that can assess this a *Language Model*

Language Modeling

- How might we go about calculating such a conditional probability?
  - One way is to use the definition of conditional probabilities and look for counts. So, to get

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

- By definition, that's

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

→ We can get each of those from counts in a large corpus.

Very Easy Estimate

- How to estimate?

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

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

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

Very Easy Estimate

- According to Google those counts are 5/9
  - Unfortunately... 2 of those were to these slides... So maybe it’s really
  - 3/7
  - In any case, that’s not terribly convincing due to the small numbers involved

Language Modeling

- Unfortunately, for most sequences and for most text collections we won’t get good estimates from this method
  - What we’re likely to get is 0
  - Or worse: $\frac{0}{0}$
- Clearly, we’ll have to be a little cleverer
  - Let’s use the chain rule of probability
  - And a particularly useful independence assumption

Markov Assumption

For each component in the product, replace with the approximation (assuming a prefix of length N)

$$P(w_n \mid w_1^{n-1}) \approx P(w_n \mid w_{n-N+1}^{n-1})$$

Bigram version

$$P(w_n \mid w_1^{n-1}) \approx P(w_n \mid w_{n-1})$$

Estimating Bigram Probabilities

- The Maximum Likelihood Estimate (MLE)

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

An Example

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

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

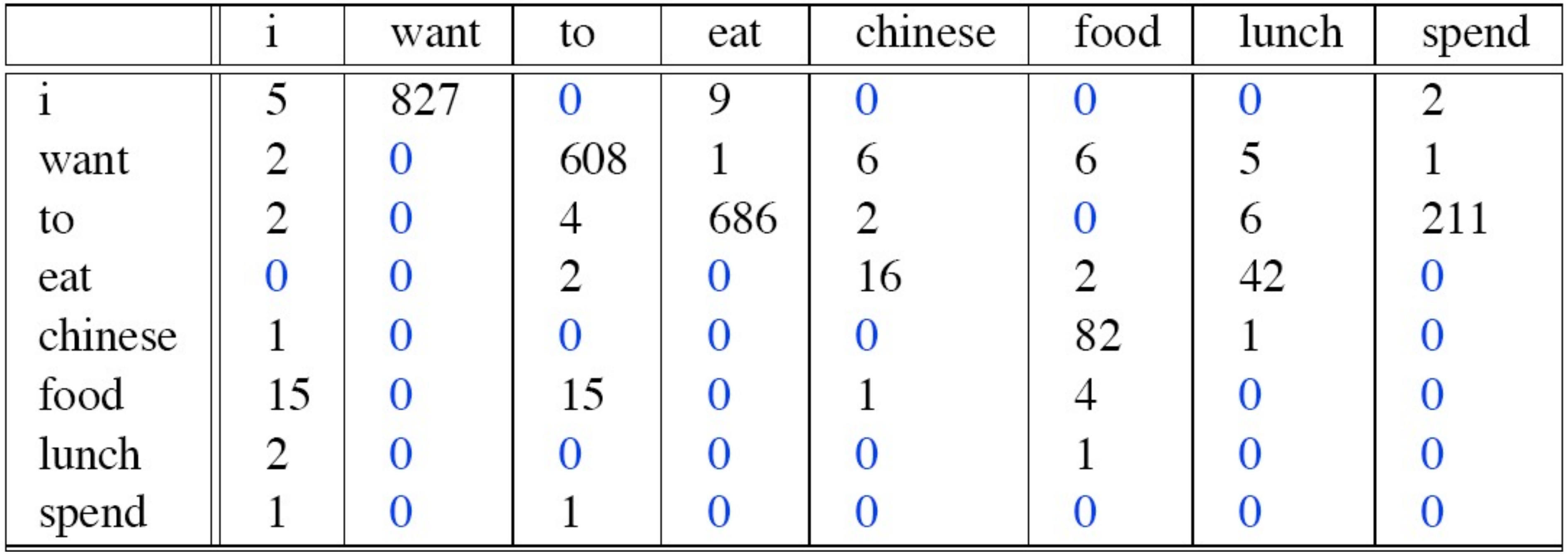
$$P(\texttt{</s>} \mid \texttt{Sam}) = \tfrac{1}{2} = 0.5 \qquad P(\texttt{Sam} \mid \texttt{am}) = \tfrac{1}{2} = .5 \qquad P(\texttt{do} \mid \texttt{I}) = \tfrac{1}{3} = .33$$

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

Bigram Counts (Berkeley Restaurant Corpus)

https://web.stanford.edu/~jurafsky/icslp-red.pdf

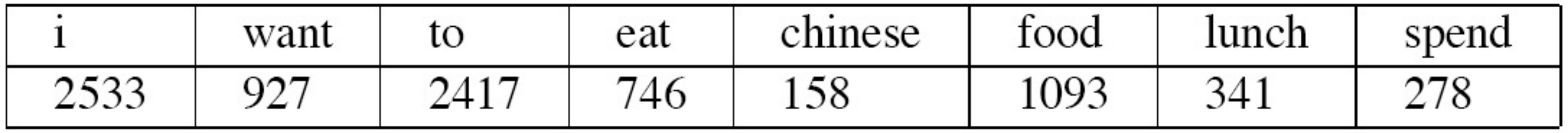
- Out of 9222 sentences
  - Eg. “I want” occurred 827 times



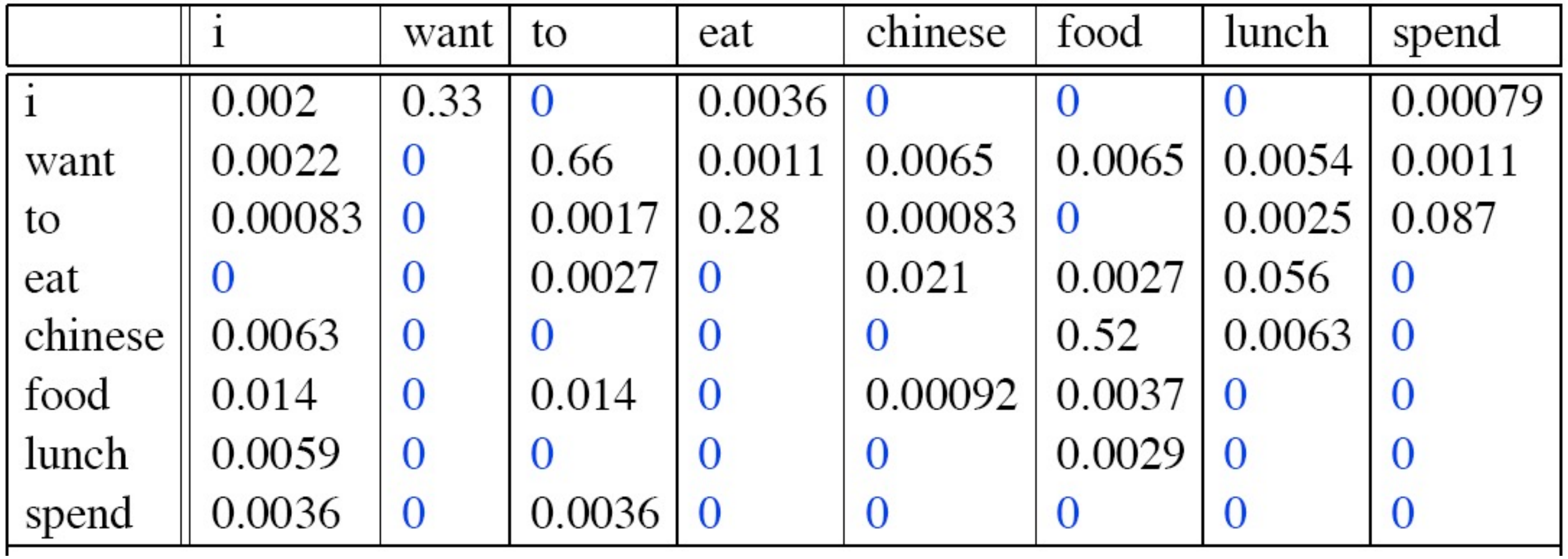
	i	want	to	eat	chinese	food	lunch	spend
i	5	827	0	9	0	0	0	2
want	2	0	608	1	6	6	5	1
to	2	0	4	686	2	0	6	211
eat	0	0	2	0	16	2	42	0
chinese	1	0	0	0	0	82	1	0
food	15	0	15	0	1	4	0	0
lunch	2	0	0	0	0	1	0	0
spend	1	0	1	0	0	0	0	0

Bigram Probabilities

- Divide bigram counts by prefix unigram counts to get probabilities.



i	want	to	eat	chinese	food	lunch	spend
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> I want english food </s>) =

  P(I|<s>)*

  P(want|I)*

  P(english|want)*

  P(food|english)*

  P(</s>|food)*

  = .000031

Kinds of Knowledge

- As crude as they are, N-gram probabilities capture a range of interesting facts about language.

  - P(english|want) = .0011
  - P(chinese|want) = .0065
  - P(to|want) = .66
  - P(eat | to) = .28
  - P(food | to) = 0
  - P(want | spend) = 0
  - P (i | <s>) = .25

World knowledge

Syntax

Discourse

Shannon’s Method

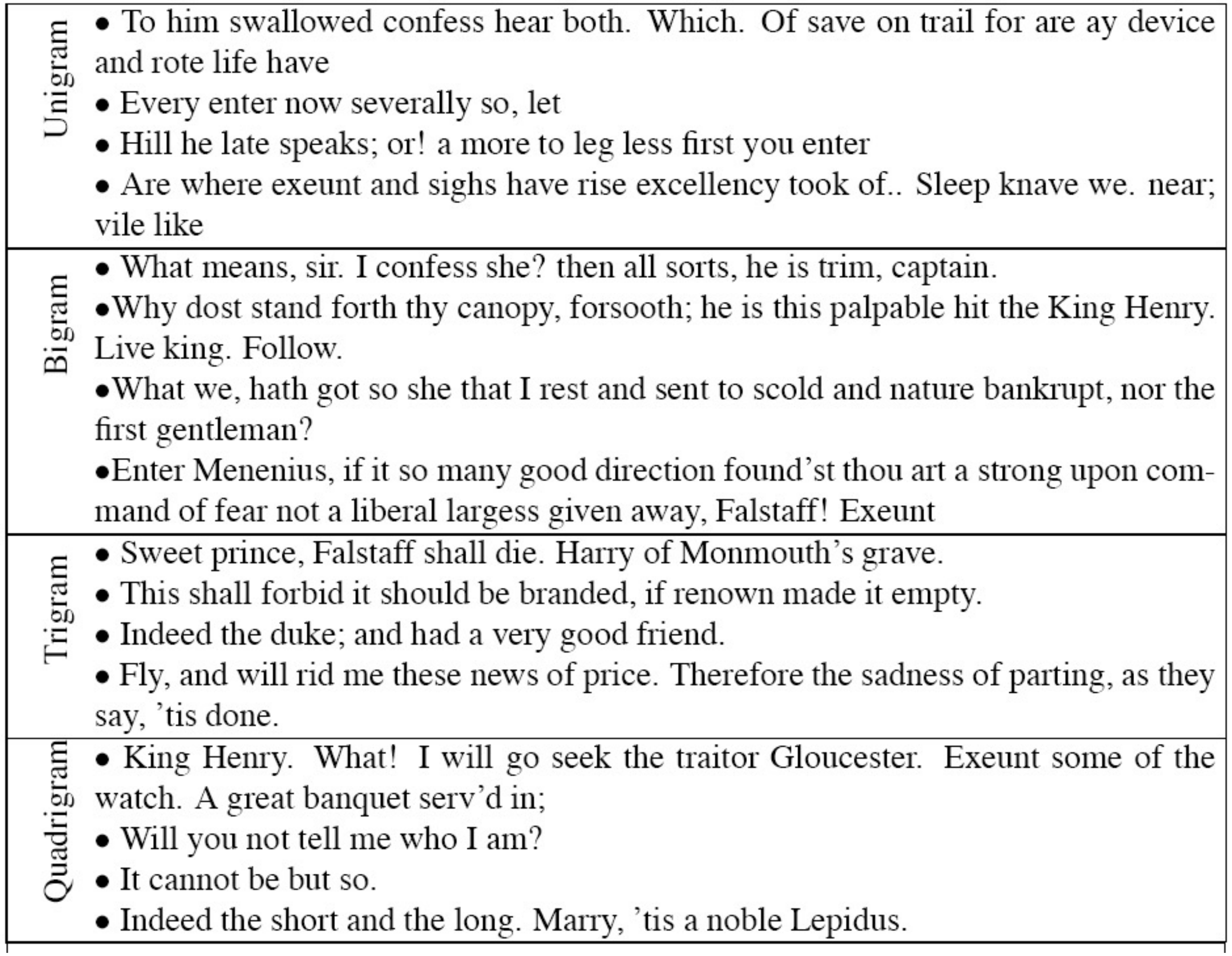
- Language models are generative models
- Assigning probabilities to sentences is all well and good, but it’s not terribly illuminating . A more interesting task is to turn the model around and use it to generate random sentences that are *like* the sentences from which the model was derived.
- Generally attributed to Claude Shannon

Shannon’s Method

- Sample a random bigram (<s>, w) according to its probability
- Now sample a random bigram (w, x) according to its probability
  - Where the prefix w matches the suffix of the previous bigram
- And so on until we randomly choose a (y, </s>)
- Then string the words together

```
<s> I
      I want
         want to
              to eat
                 eat Chinese
                     Chinese food
                            food  </s>
```

Shakespeare $(N = 884{,}647 \text{ tokens}, V = 29{,}066)$



Unigram	• To him swallowed confess hear both. Which. Of save on trail for are ay device and rote life have • Every enter now severally so, let • Hill he late speaks; or! a more to leg less first you enter • Are where exeunt and sighs have rise excellency took of.. Sleep knave we. near; vile like
Bigram	• What means, sir. I confess she? then all sorts, he is trim, captain. •Why dost stand forth thy canopy, forsooth; he is this palpable hit the King Henry. Live king. Follow. •What we, hath got so she that I rest and sent to scold and nature bankrupt, nor the first gentleman? •Enter Menenius, if it so many good direction found'st thou art a strong upon command of fear not a liberal largess given away, Falstaff! Exeunt
Trigram	• Sweet prince, Falstaff shall die. Harry of Monmouth's grave. • This shall forbid it should be branded, if renown made it empty. • Indeed the duke; and had a very good friend. • Fly, and will rid me these news of price. Therefore the sadness of parting, as they say, 'tis done.
Quadrigram	• King Henry. What! I will go seek the traitor Gloucester. Exeunt some of the watch. A great banquet serv'd in; • Will you not tell me who I am? • It cannot be but so. • Indeed the short and the long. Marry, 'tis a noble Lepidus.

N-Gramme für Harry Potter

`` I've warned you , Snivellus , '' said Dumbledore happily to a thunderstruck Umbridge .

Snape was still drifting weirdly ahead of Black , and began to spin . Next second he was gathering speed in a steep dive , racing the ball.

Filch's jowls wobbled with silent laughter .
`` Guilty conscience , eh ? '' he said cheerfully . He was in a very good mood until lunchtime , when he thought he 'd stretch his legs and walk across the road to buy himself a bun from the bakery . He 'd forgotten all about the people in cloaks until he passed a group of them next to the baker 's . He eyed them angrily as he passed . He guessed that many of them had believed Rita Skeeter 's article about how disturbed and possibly dangerous he was .

Speech and Language Processing

Evaluating Language Models

Evaluation

- How do we know if our models are any good?
  - And how do we know if one model is better than another?

- Well Shannon’s game gives us an intuition
  - The generated texts from the higher order models sure look better. That is, they sound more like the text the model was obtained from.
  - But what does that mean? Can we make that notion operational?

Evaluation

- Standard method
  - Train parameters of our model on a **training set**
  - Look at model performance on some new data
    - This is exactly what happens in the real world; we want to know how our model performs on data we haven't seen
  - So, use a **test set**: A dataset which is different than our training set, but is drawn from the same source
  - Then we need an **evaluation metric** to tell us how well our model is doing on the test set
    - One such metric is **perplexity** (to be introduced below)

      ➔ intrinsic evaluation

Perplexity

- Perplexity is the probability of the test set (assigned by the language model), normalized by the number of words:

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

Perplexity

- Perplexity is the probability of the test set (assigned by the language model), normalized by the number of words:

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

- Chain rule:

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

- For bigrams:

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

Perplexity

- Perplexity is the probability of the test set (assigned by the language model), normalized by the number of words:

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

- Chain rule:

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

- For bigrams:

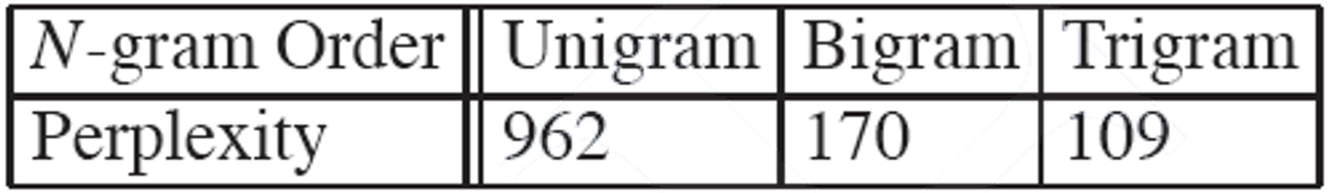
$$\mathrm{PP}(W) = \sqrt[N]{\prod_{i=1}^{N} \frac{1}{P(w_i | w_{i-1})}}$$

- Minimizing perplexity is the same as maximizing probability

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

Lower perplexity means a better model

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



N-gram Order	Unigram	Bigram	Trigram
Perplexity	962	170	109

Evaluating *N*-Gram Models

- Best evaluation for a language model
  - Put model A into an application
    - For example, a speech recognizer
  - Evaluate the performance of the application with model A
  - Put model B into the application and evaluate
  - Compare performance of the application with the two models

  → ***Extrinsic evaluation***

Speech and Language Processing

Smoothing

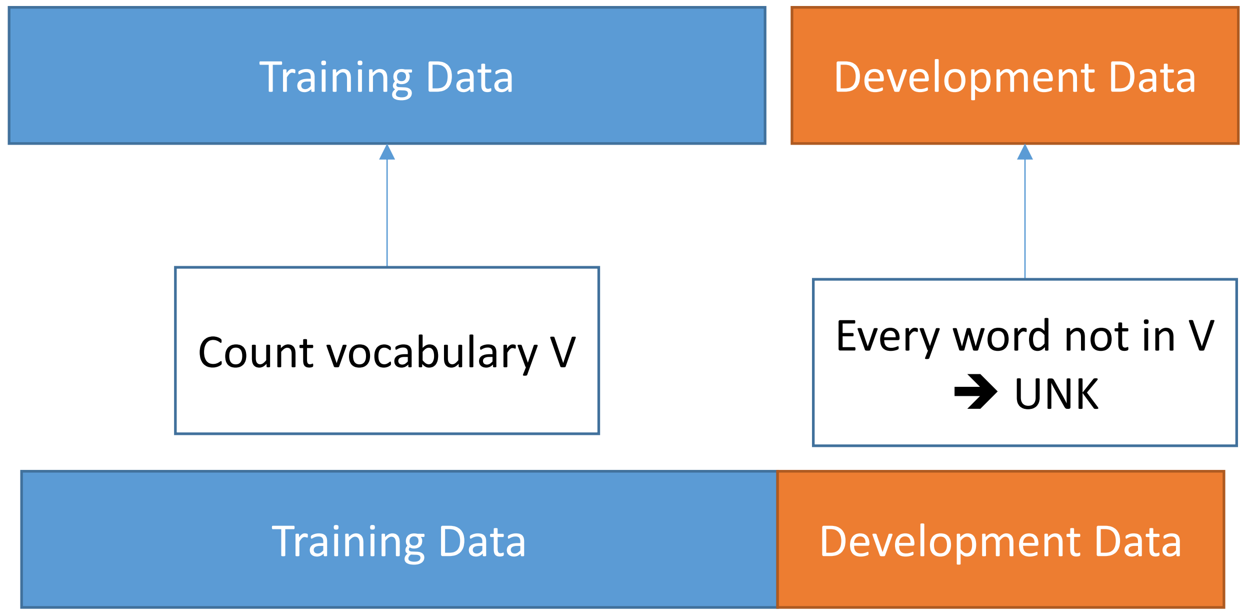
Problems with language models

- Many words in the test set **never** occurred within our training data
  - ➔ Problem of unknown words <UNK>

- Many bigrams (N-Grams) never occur in the training data, or only appeared very infrequently
  - ➔ Unreliable counts or zero counts

Unknown Words

- One way to deal with unknown words is to introduce one (or more) extra entries to our vocabulary <UNK>



- Calculate probabilities in the combination of training and development data (including UNK)

Zero Counts

- Back to Shakespeare
  - Recall that Shakespeare produced 300,000 bigram types out of $V^2 = 844 \text{ million}$ possible bigrams...
  - So, 99.96% of the possible bigrams were never seen (have zero entries in the table)
  - Does that mean that any sentence that contains one of those bigrams should have a probability of 0?

Zero Counts

- Some of those zeros are really zeros...
  - Things that really can’t or shouldn’t happen
- On the other hand, some of them are just rare events
  - If the training corpus had been a little bigger, they would have had a count (probably a count of 1)
- Zipf’s Law (long tail phenomenon):
  - A small number of events occur with high frequency
  - A large number of events occur with low frequency
  - You can quickly collect statistics on the high frequency events
  - You might have to wait an arbitrarily long time to get valid statistics on low frequency events
- Result:
  - Our estimates are sparse! We have no counts at all for the vast bulk of things we want to estimate!
- Answer:
  - Estimate the likelihood of unseen (zero count) N-grams

Laplace Smoothing

- Also called add-one smoothing
- Very simple: Just add one to all the counts!
- For bigrams:
  - We add 1 to every bigram count (no matter if we saw it or not!)

$$p(w_{i-1}w_i) = \frac{\mathrm{count(w_{i-1}, w_i) + 1}}{\sum_{w_{\hat{\imath}}} count(w_{i-1}, w_{\hat{\imath}}) + 1} = \frac{\mathrm{count(w_{i-1}, w_i) + 1}}{count(w_{i-1}) + V}$$

Bigram Counts (Berkeley Restaurant Corpus)

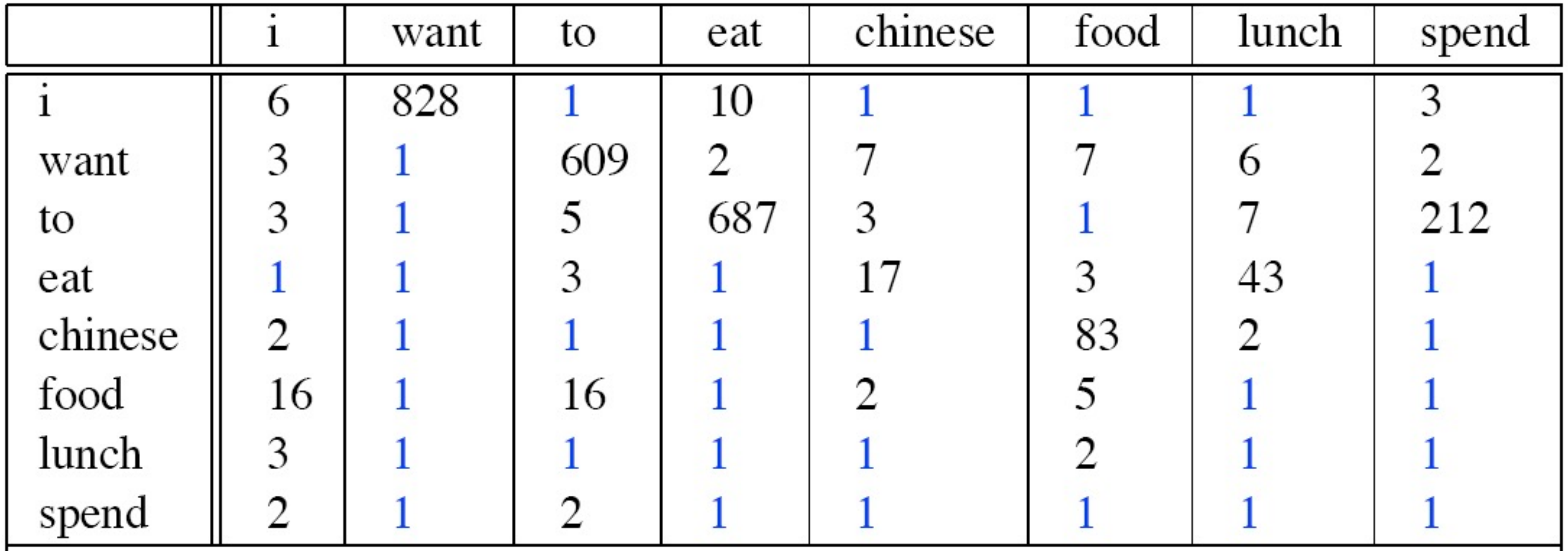
Recall

https://web.stanford.edu/~jurafsky/icslp-red.pdf

- Out of 9222 sentences
  - Eg. “I want” occurred 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

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 Bigram Probabilities

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

	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	$\boxed{0.26}$	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

Bigram Probabilities

Recall

- Divide bigram counts by prefix unigram counts to get probabilities.

i	want	to	eat	chinese	food	lunch	spend
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	$\boxed{0.66}$	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

Big Change to the Counts!

- P(to|want) from .66 to .26 !
  - So, in general, Laplace is a blunt instrument
  - Could use more fine-grained method (add-k)

- Laplace smoothing usually not used for N-grams, as we have much better methods

- Despite its flaws Laplace (add-k) is however still used to smooth other probabilistic models in NLP, especially
  - For pilot studies
  - in domains where the number of zeros isn't so huge
  - Naïve Bayes
  - Hidden Markov Models

Better Smoothing

- Intuition used by many smoothing algorithms
  - Good-Turing
  - Kneser-Ney
  - Witten-Bell

- Is to use the count of things we've seen once to help estimate the count of things we've never seen

Good-Turing

Josh Goodman Intuition

- You are fishing and have caught
  - 10 carp, 3 catfish, 2 elephant, 1 trout, 1 salmon, 1 eel = 18 fish
- How likely is it that next species is trout?
  - 1/18
- How likely is it that the next fish caught is from a new species (one not seen in our previous catch)?
  - Use things-we-saw-once to estimate new things
  - $N_1 = 3 \Rightarrow 3/18$ 1 trout, 1 salmon, 1 eel
- Assuming so, how likely is it that next species is trout?
  - Less than 1/18

Good-Turing

- Counts: c
- Notation: N_c is the frequency of observed frequency c
  - $N_{10} = 1, \quad N_1 = 3$
    - Number of fish species seen 10 times is 1 (carp)
    - Number of fish species seen 1 time is 3 (trout, salmon, eel)
- Good-Turing Probability: P^*_{GT}

$$P^*_{GT}(X_0) = P^*_{GT}(\text{things with frequency zero in training}) = \frac{N_1}{N}$$

$$P^*_{GT}(X_c) = \frac{c^*(X_c)}{N}$$

- New Good-Turing adjusted count: $c*$

$$c^* = (c+1)\frac{N_{c+1}}{N_c}$$

10 carp,
3 catfish,
2 elephant,
1 trout
1 salmon
1 Eel

GT Fish Example

$$P^*_{GT}(\text{things with frequency zero in training}) = \frac{N_1}{N}$$

$$c^* = (c+1)\frac{N_{c+1}}{N_c}$$

In general: For an event X_c which has been seen c times in training:

$$P^*_{GT}(X_c) = \frac{c^*(X_c)}{N}$$

X	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: Bigrams

- The probability of a bigram, smoothed with GT is then:

$$p^*(w_j|w_i) = \frac{c^*_{GT}(w_i, w_j)}{\sum_{w_k} c^*_{GT}(w_i, w_k)}$$

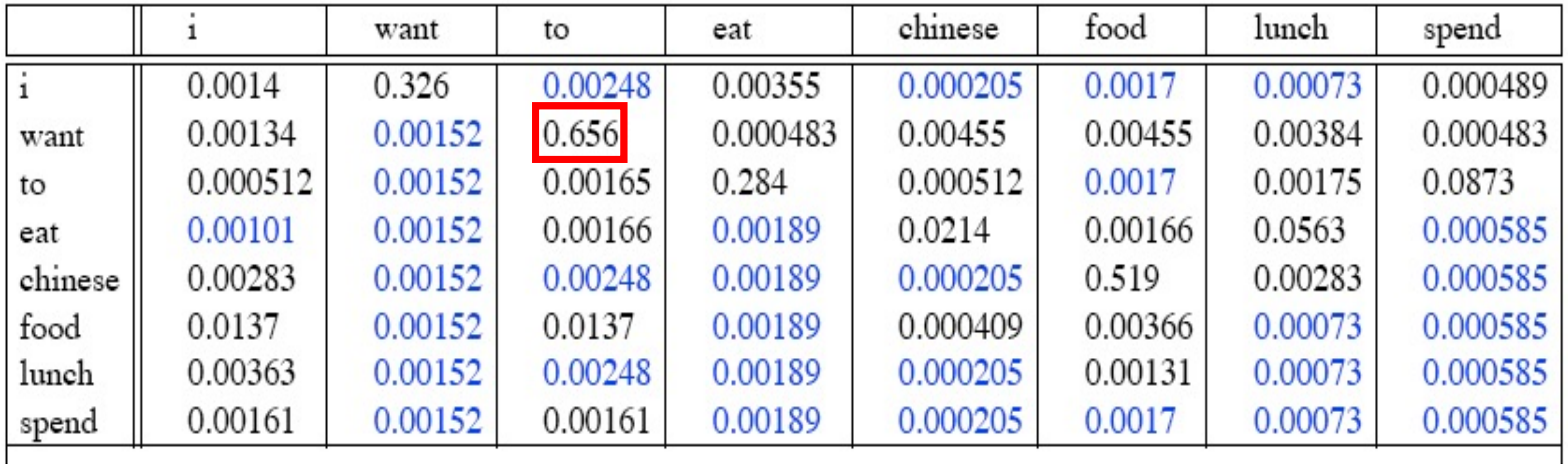
- With the smoothed counts as:

$$c^*_{GT}(X_c) = (c+1)\frac{N_{c+1}}{N_c}$$

And: $N_0 = V^2 - \sum_{i=1} N_i$

GT Smoothed Bigram Probabilities

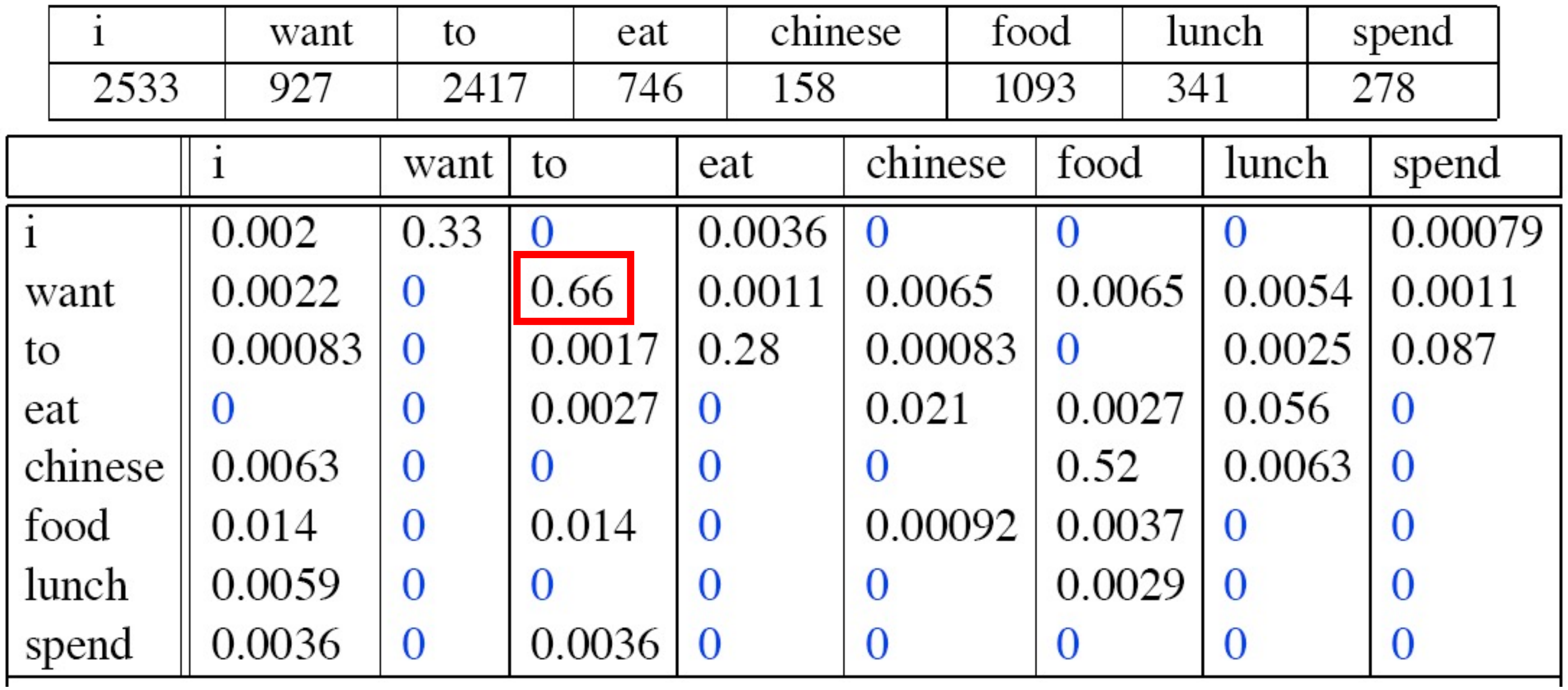
- Much smaller changes to existing probabilities!



	i	want	to	eat	chinese	food	lunch	spend
i	0.0014	0.326	0.00248	0.00355	0.000205	0.0017	0.00073	0.000489
want	0.00134	0.00152	$\boxed{0.656}$	0.000483	0.00455	0.00455	0.00384	0.000483
to	0.000512	0.00152	0.00165	0.284	0.000512	0.0017	0.00175	0.0873
eat	0.00101	0.00152	0.00166	0.00189	0.0214	0.00166	0.0563	0.000585
chinese	0.00283	0.00152	0.00248	0.00189	0.000205	0.519	0.00283	0.000585
food	0.0137	0.00152	0.0137	0.00189	0.000409	0.00366	0.00073	0.000585
lunch	0.00363	0.00152	0.00248	0.00189	0.000205	0.00131	0.00073	0.000585
spend	0.00161	0.00152	0.00161	0.00189	0.000205	0.0017	0.00073	0.000585

Bigram Probabilities

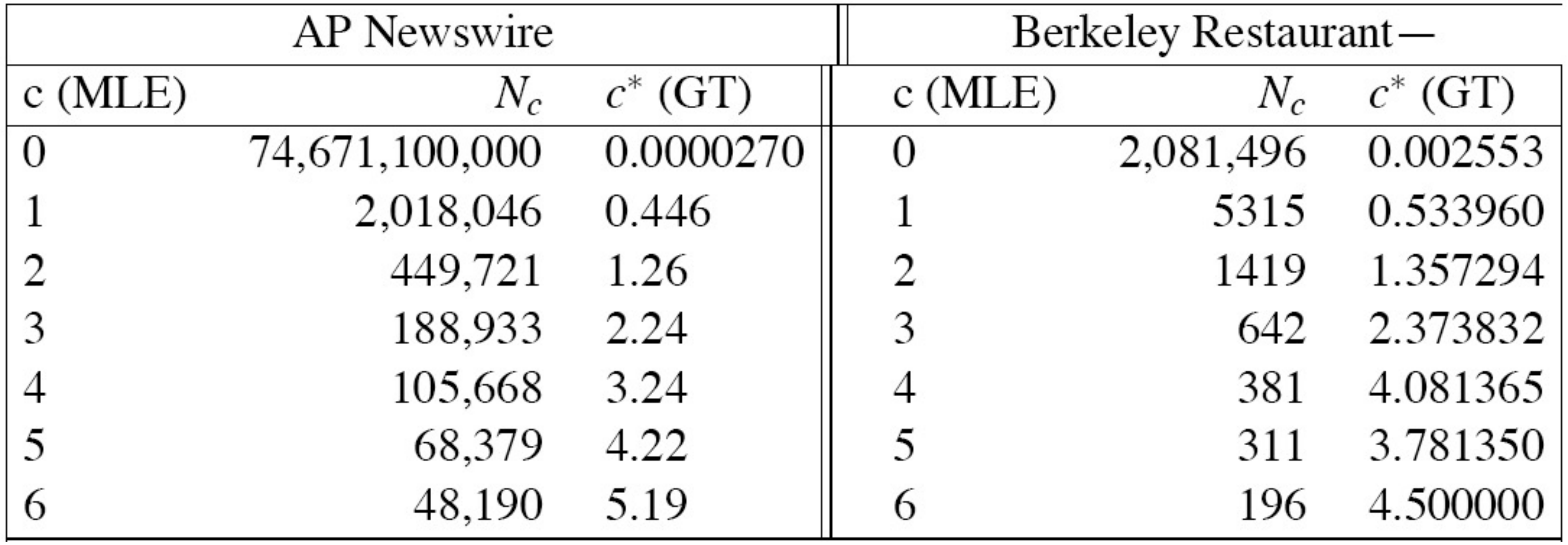
- Divide bigram counts by prefix unigram counts to get probabilities.



i	want	to	eat	chinese	food	lunch	spend
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	$\boxed{0.66}$	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 Frequencies of Frequencies and GT Re-estimates



AP Newswire			Berkeley Restaurant—		
c (MLE)	N_c	c^* (GT)	c (MLE)	N_c	c^* (GT)
0	74,671,100,000	0.0000270	0	2,081,496	0.002553
1	2,018,046	0.446	1	5315	0.533960
2	449,721	1.26	2	1419	1.357294
3	188,933	2.24	3	642	2.373832
4	105,668	3.24	4	381	4.081365
5	68,379	4.22	5	311	3.781350
6	48,190	5.19	6	196	4.500000

Good-Turing: Problems

- Recall the adjusted counts:

$$c^*_{GT}(X_c) = (c+1)\frac{N_{c+1}}{N_c}$$

- What if $N_c = 0$??

N_c	Amount
N_{10}	8
N_{11}	??
N_{12}	6
N_{13}	5

- Usually estimated by (linear) interpolation!

Counting vs. Smoothing

- Counting (MLE) is correct on average, but varies a lot with different data:
  - → Unbiased, but high variance

- Smoothed counts are wrong on average but vary a lot less with different data:
  - → Biased, but low(er) variance

Backoff and Interpolation

- Another useful source of knowledge

- If we are estimating:
  - trigram $P(z|x, y)$
  - but $count(xyz)$ is zero

- Use info from:
  - Bigram $P(z|y)$

- Or even:
  - Unigram $P(z)$

- How to combine this trigram, bigram, unigram info in a valid fashion?

Backoff vs Interpolation

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

Interpolation

- Simple interpolation

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

- Lambdas conditional on context:

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

How to Set the Lambdas?

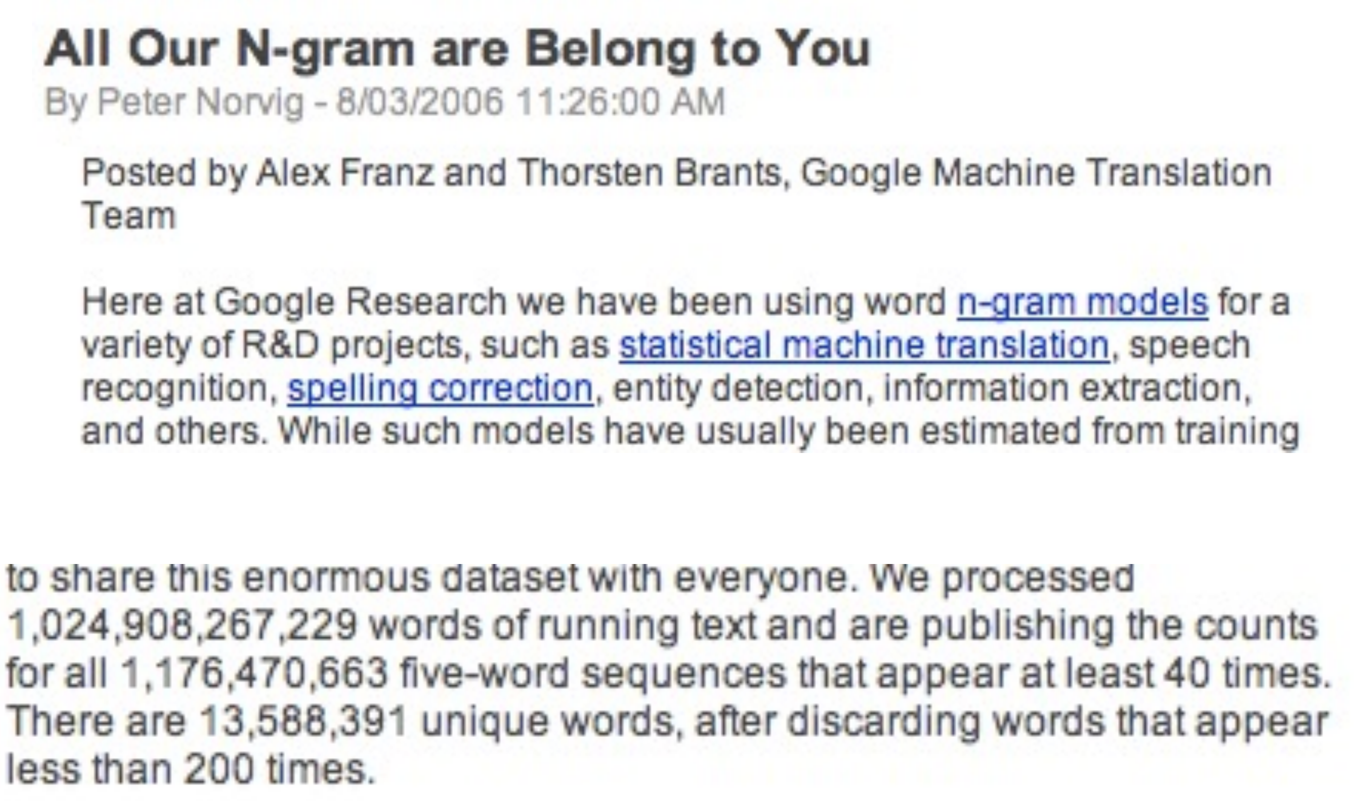
- Use a **held-out, or development**, corpus
- Choose lambdas which maximize the probability of some held-out data
  - I.e., fix the N-gram probabilities
  - Then search for lambda values
  - That when plugged into previous equation
  - Give largest probability for held-out set
  - Can use EM to do this search

Practical Issues

- We do everything in log space
  - Avoid underflow
  - (also adding is faster than multiplying)

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

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 Example:

- 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
- ...

Google Caveat

- Remember the lesson about test sets and training sets:
  Test sets should be similar to the training set (drawn from the same distribution) for the probabilities to be meaningful
- So... The Google corpus is fine if your application deals with arbitrary English text on the Web
- If not, then a smaller domain specific corpus is likely to yield better results