CS 124/LINGUIST 180: From Language to Information

Dan Jurafsky

Lecture 3: Intro to Probability, Language Modeling

Outline

- Probability
  - Basic probability
  - Conditional probability
- Language Modeling (N-grams)
  - N-gram Intro
  - The Chain Rule
  - The Shannon Visualization Method
  - Evaluation:
    - Perplexity
  - Smoothing:
    - Laplace (Add-1)
    - Add-prior

Language Modeling

- We want to compute
  - $P(w_1,w_2,w_3,w_4,w_5 \ldots w_n) = P(W)$
  - = the probability of a sequence
- Alternatively we want to compute
  - $P(w_5|w_1,w_2,w_3,w_4)$
  - =the probability of a word given some previous words
- The model that computes
  - $P(W)$ or
  - $P(w_n|w_1,w_2 \ldots w_{n-1})$
  - is called the language model.
- A better term for this would be “The Grammar”
- But “Language model” or LM is standard

Computing P(W)

- How to compute this joint probability:
  - P(“the”,”other”,”day”,”I”,”was”,”walking”,”along”,”and”,”saw”,”a”,”lizard”)
- Intuition: let’s rely on the Chain Rule of Probability

The Chain Rule

- Recall the definition of conditional probabilities

$$P(A|B) = \frac{P(A \wedge B)}{P(B)}$$

- Rewriting:

$$P(A \wedge B) = P(A|B)P(B)$$

- More generally
- P(A,B,C,D) = P(A)P(B|A)P(C|A,B)P(D|A,B,C)
- In general
- $P(x_1,x_2,x_3,\ldots x_n) = P(x_1)P(x_2|x_1)P(x_3|x_1,x_2)\ldots P(x_n|x_1\ldots x_{n-1})$

The Chain Rule applied to joint probability of words in sentence

$$\begin{aligned} P(w_1^n) &= P(w_1)P(w_2|w_1)P(w_3|w_1^2)\dots P(w_n|w_1^{n-1}) \\ &= \prod_{k=1}^{n} P(w_k|w_1^{k-1}) \end{aligned}$$

- P("the big red dog was")=

  P(the) * P(big|the) * P(red|the big) * P(dog|the big red) * P(was|the big red dog)

Very easy estimate:

- How to estimate?
  - P(the | its water is so transparent that)

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

Unfortunately

- There are a lot of possible sentences

- We'll never be able to get enough data to compute the statistics for those long prefixes

- P(lizard|the,other,day,I,was,walking,along,and,saw,a)
- Or
- P(the|its water is so transparent that)

Markov Assumption

- Make the simplifying assumption
  - P(lizard|the,other,day,I,was,walking,along,and,saw,a) = P(lizard|a)
- Or maybe
  - P(lizard|the,other,day,I,was,walking,along,and,saw,a) = P(lizard|saw,a)

Markov Assumption

- So for each component in the product replace with the approximation (assuming a prefix of 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

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

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

An example

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

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

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

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

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

Maximum Likelihood Estimates

- The maximum likelihood estimate of some parameter of a model M from a training set T
  - Is the estimate
  - that maximizes the likelihood of the training set T given the model M
- Suppose the word Chinese occurs 400 times in a corpus of a million words (Brown corpus)
- What is the probability that a random word from some other text will be "Chinese"
- MLE estimate is 400/1000000 = .004
  - This may be a bad estimate for some other corpus
- But it is the **estimate** that makes it **most likely** that "Chinese" will occur 400 times in a million word corpus.

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

- 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

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

Bigram estimates of sentence probabilities

- P(<s> I want english food </s>) =

  P(i|<s>) x

  P(want|I) x

  P(english|want) x

  P(food|english) x

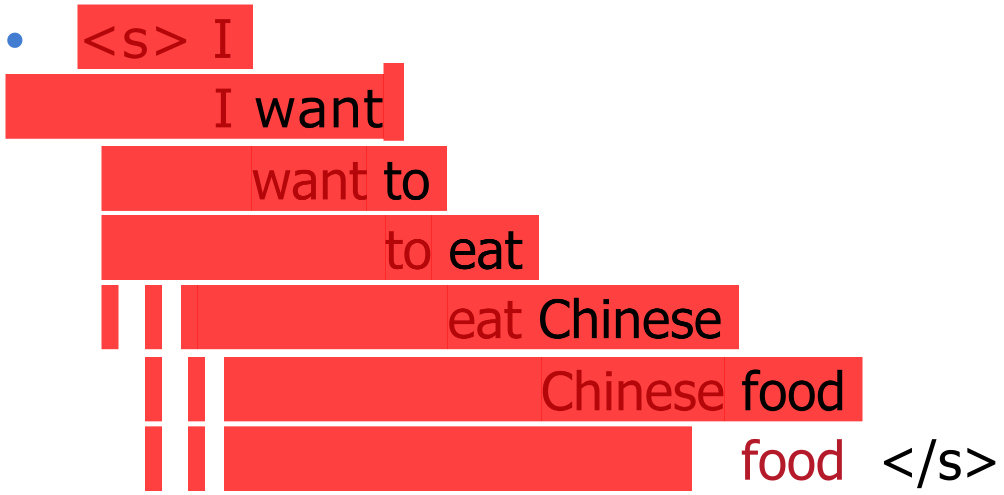
  P(</s>|food)

  =.000031

What kinds of knowledge?

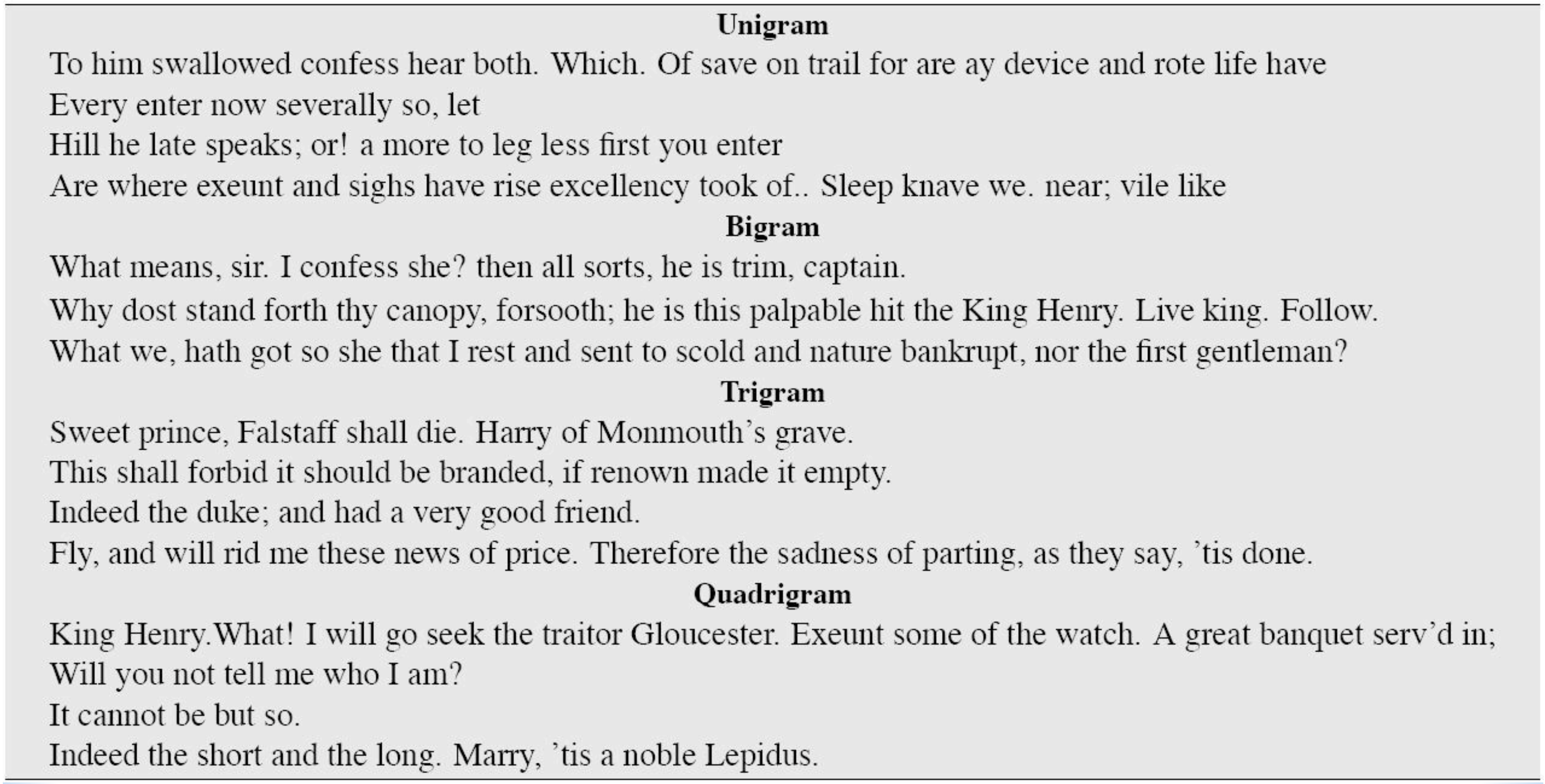
- P(english|want) = .0011
- P(chinese|want) = .0065
- P(to|want) = .66
- P(eat | to) = .28
- P(food | to) = 0
- P(want | spend) = 0
- P (i | <s>) = .25

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
- 

Approximating Shakespeare

-



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?

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

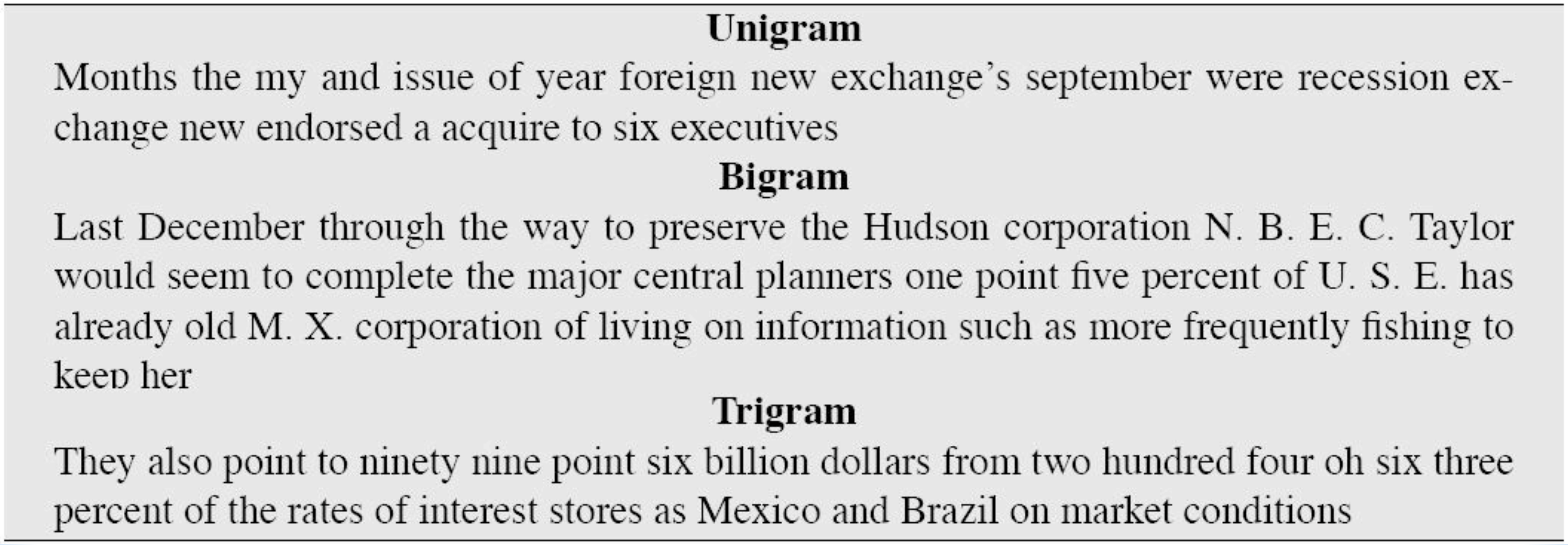
King Henry.What! I will go seek the traitor Gloucester. Exeunt some of the watch. A great banquet serv'd in;
Will you not tell me who I am?
It cannot be but so.
Indeed the short and the long. Marry, 'tis a noble Lepidus.

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 worse: What's coming out looks like Shakespeare because it ***is*** Shakespeare

The wall street journal is not shakespeare (no offense)

-



Unigram

Months the my and issue of year foreign new exchange's september were recession exchange new endorsed a acquire to six executives

Bigram

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

Trigram

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

Lesson 1: 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, adapt to test set, etc

Lesson 2: zeros or not?

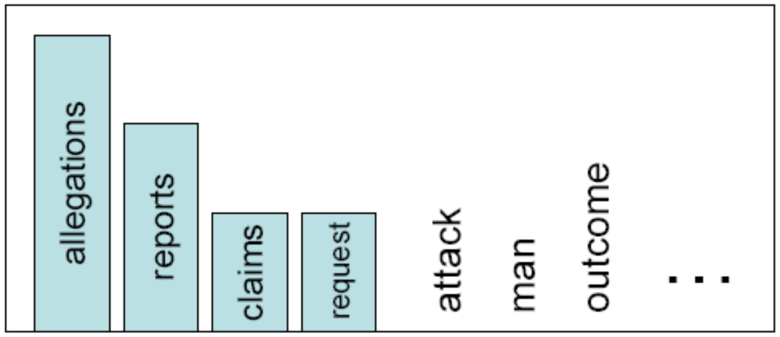
- Some of those zeros are really zeros...
  - Things that really aren't ever going to happen
- On the other hand, some of them are just rare events.
  - If the training corpus had been a little bigger they would have had a count
    - What would that count be in all likelihood?
- Zipf's Law (long tail 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!

Slide adapted from Bonnie Dorr and Julia Hirschberg

Smoothing is like Robin Hood: Steal from the rich and give to the poor (in probability mass)

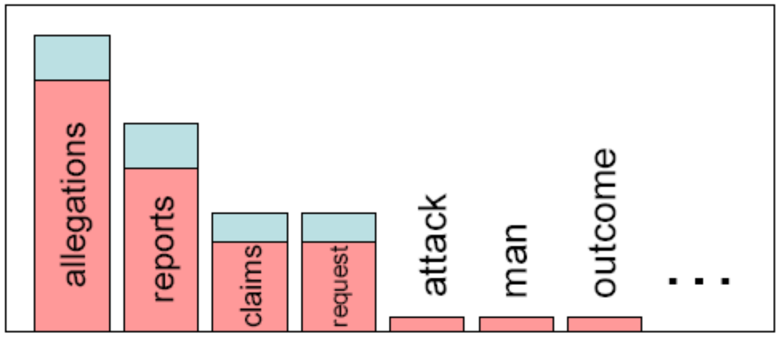
- We often want to make predictions from sparse statistics:

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!

Slide from Dan Klein

Laplace smoothing

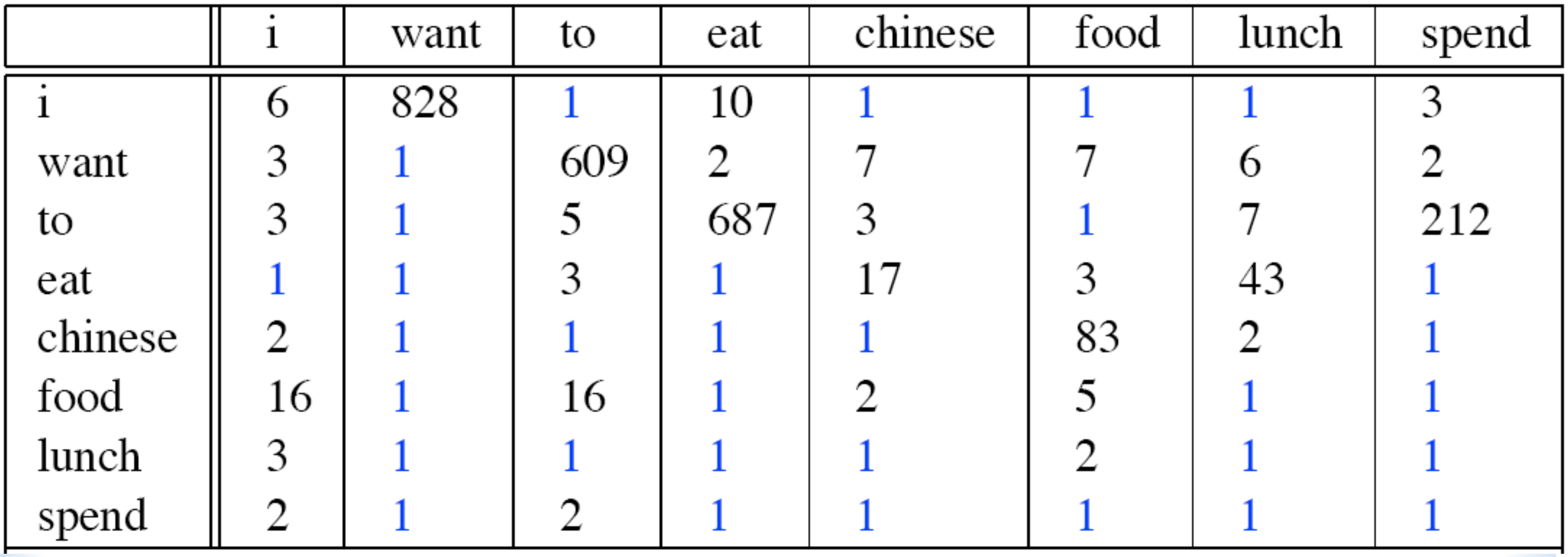
- Also called add-one smoothing
- Just add one to all the counts!
- Very simple

- MLE estimate: $P(w_i) = \frac{c_i}{N}$

- Laplace estimate: $P_{\text{Laplace}}(w_i) = \frac{c_i + 1}{N + V}$

- Reconstructed counts: $c_i^* = (c_i + 1)\frac{N}{N + V}$

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

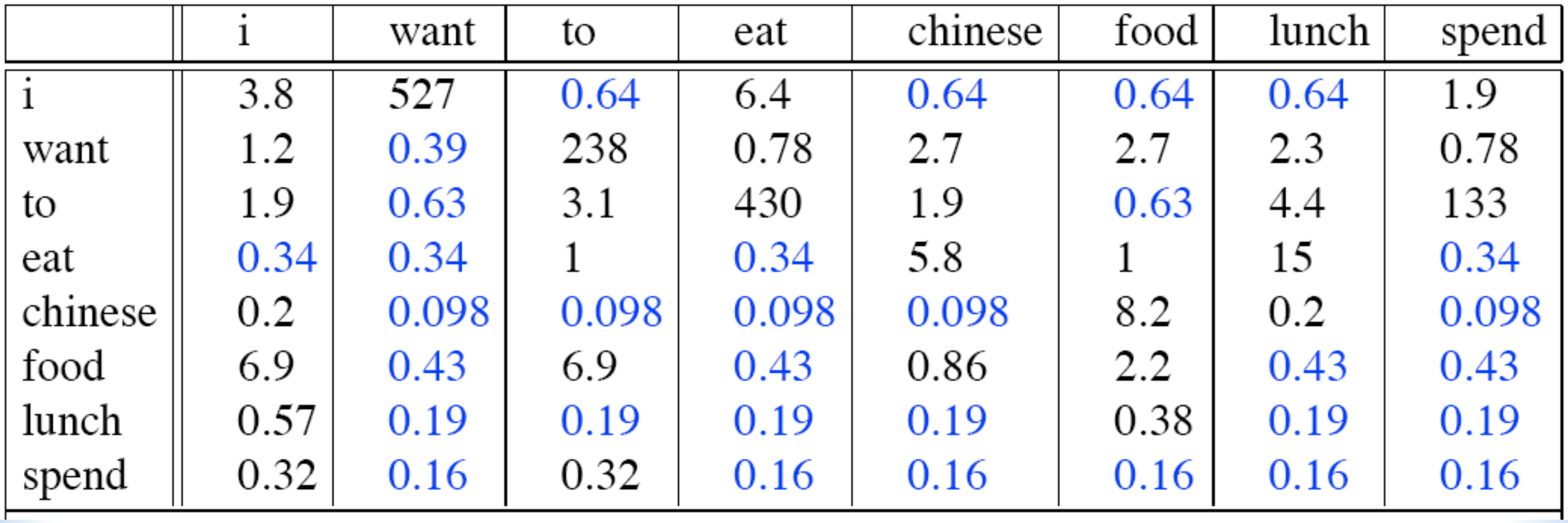
Laplace-smoothed bigrams

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

	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

Note big change to counts

- C(count to) went from 608 to 238!
- P(to|want) from .66 to .26!
- Discount d= c*/c
  - d for "chinese food" =.10!!! A 10x reduction
  - So in general, Laplace is a blunt instrument
- But Laplace smoothing 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.

Add-k

- Add a small fraction instead of 1

Bayesian unigram prior smoothing for bigrams

- Maximum Likelihood Estimation

$$P(w_2 \mid w_1) = \frac{C(w_1, w_2)}{C(w_1)}$$

- Laplace Smoothing

$$P_{Laplace}(w_2 \mid w_1) = \frac{C(w_1, w_2) + 1}{C(w_1) + vocab}$$

- Bayesian prior Smoothing

$$P_{Prior}(w_2 \mid w_1) = \frac{C(w_1, w_2) + P(w_2)}{C(w_1) + 1}$$

Practical Issues

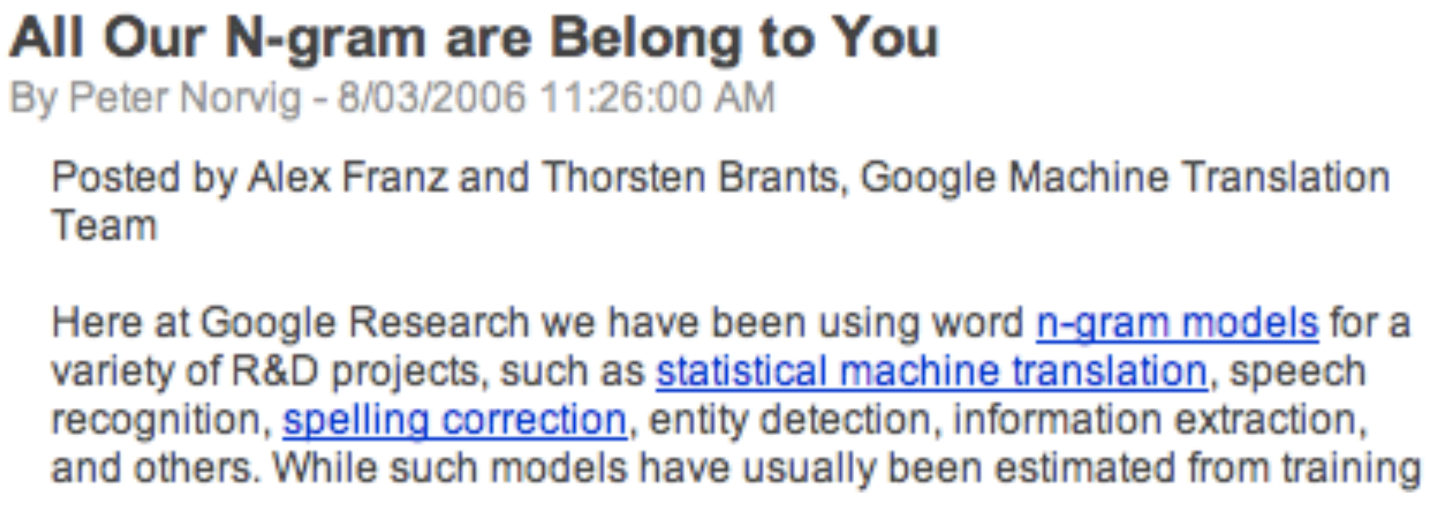
- We do everything in log space
  - Avoid underflow
  - (also adding is faster than multiplying)

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

Language Modeling Toolkit

- SRILM
  - http://www.speech.sri.com/projects/srilm/

Google N-Gram Release

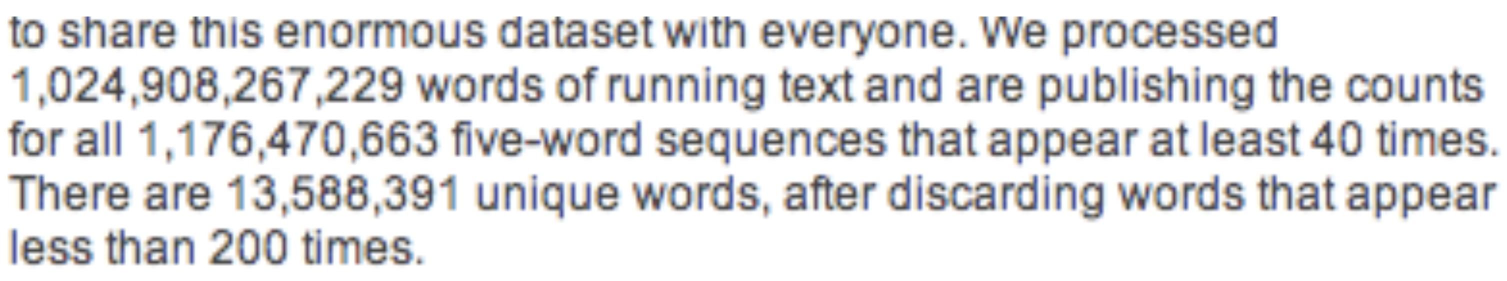


All Our N-gram are Belong to You

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

Evaluating N-gram models

- Best evaluation for an N-gram
  - Put model A in a speech recognizer
  - Run recognition, get word error rate (WER) for A
  - Put model B in speech recognition, get word error rate for B
  - Compare WER for A and B
  - **In-vivo evaluation**

Difficulty of in-vivo evaluation of N-gram models

- In-vivo evaluation
  - This is really time-consuming
  - Can take days to run an experiment
- So
  - As a temporary solution, in order to run experiments
  - To evaluate N-grams we often use an approximation called **perplexity**
    - But perplexity is a poor approximation unless the test data looks **just** like the training data
    - So is **generally only useful in pilot experiments (generally is not sufficient to publish)**
    - But is helpful to think about.

Unknown words: Open versus closed vocabulary tasks

- **If we know all the words in advanced**
  - **Vocabulary V is fixed**
  - **Closed vocabulary task**
- **Often we don't know this**
  - **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
    - At text normalization phase, any training word not in L changed to <UNK>
    - Now we train its probabilities like a normal word
  - At decoding time
    - If text input: Use UNK probabilities for any word not in training

Evaluating N-gram models

- Best evaluation for an N-gram
  - Put model A in a task (language identification, speech recognizer, machine translation system)
  - Run the task, get an accuracy for A (how many lgs identified corrrectly, or Word Error Rate, or etc)
  - Put model B in task, get accuracy for B
  - Compare accuracy for A and B
  - **Extrinsic evaluation**

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

- Extrinsic evaluation
  - This is really time-consuming
  - Can take days to run an experiment
- So
  - As a temporary solution, in order to run experiments
  - To evaluate N-grams we often use an **intrinsic** evaluation, an approximation called **perplexity**
  - But perplexity is a poor approximation unless the test data looks **just** like the training data
  - So is **generally only useful in pilot experiments (generally is not sufficient to publish)**
  - But is helpful to think about.

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**

A totally different perplexity Intuition

- How hard is the task of recognizing digits '0,1,2,3,4,5,6,7,8,9,oh': easy, perplexity 11 (or if we ignore 'oh', perplexity 10)
- How hard is recognizing (30,000) names at Microsoft. Hard: perplexity = 30,000
- If a system has to recognize
  - Operator (1 in 4)
  - Sales (1 in 4)
  - Technical Support (1 in 4)
  - 30,000 names (1 in 120,000 each)
  - Perplexity is 54
- Perplexity is weighted equivalent branching factor

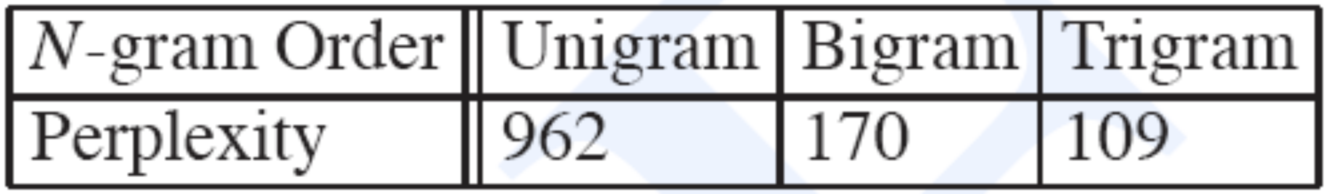
Slide from Josh Goodman

Perplexity as branching factor

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

Lower perplexity = better model

- Training 38 million words, test 1.5 million words, WSJ



N-gram Order	Unigram	Bigram	Trigram
Perplexity	962	170	109

Advanced LM stuff

- Current best smoothing algorithm
  - Kneser-Ney smoothing
- Other stuff
  - Interpolation
  - Backoff
  - Variable-length n-grams
  - Class-based n-grams
    - Clustering
    - Hand-built classes
  - Cache LMs
  - Topic-based LMs
  - Sentence mixture models
  - Skipping LMs
  - Parser-based LMs

Better discounting algorithms

- 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

Types, Tokens and Squirrels

- Much of what's coming up was first studied by field biologists who are often faced with 2 related problems
  - Determining how many species occupy a particular area (types)
  - And determining how many individuals of a given species are living in a given area (tokens)

Good-Turing: Josh Goodman intuition

- Imagine you are fishing
  - There are 8 species: carp, perch, whitefish, trout, salmon, eel, catfish, bass
- You have caught
  - 10 carp, 3 perch, 2 whitefish, 1 trout, 1 salmon, 1 eel = 18 fish
- How likely is it that next fish caught will be an eel?
- How likely is it that next fish caught will be a member of an unseen species (i.e. catfish or bass)
  - 3/18
- Now how likely is it that next fish caught will be an eel?
  - Must be less than 1/18

Slide adapted from Josh Goodman

Good-Turing Intuition

- Notation: N_x is the frequency-of-frequency-x
  - So N_{10}=1, N_1=3, etc
- To estimate total number of unseen species
  - Use number of species (words) we've seen once
  - $c_0^* = c_1$ $p_0 = N_1/N$
- All other estimates are adjusted (down) to give probabilities for unseen

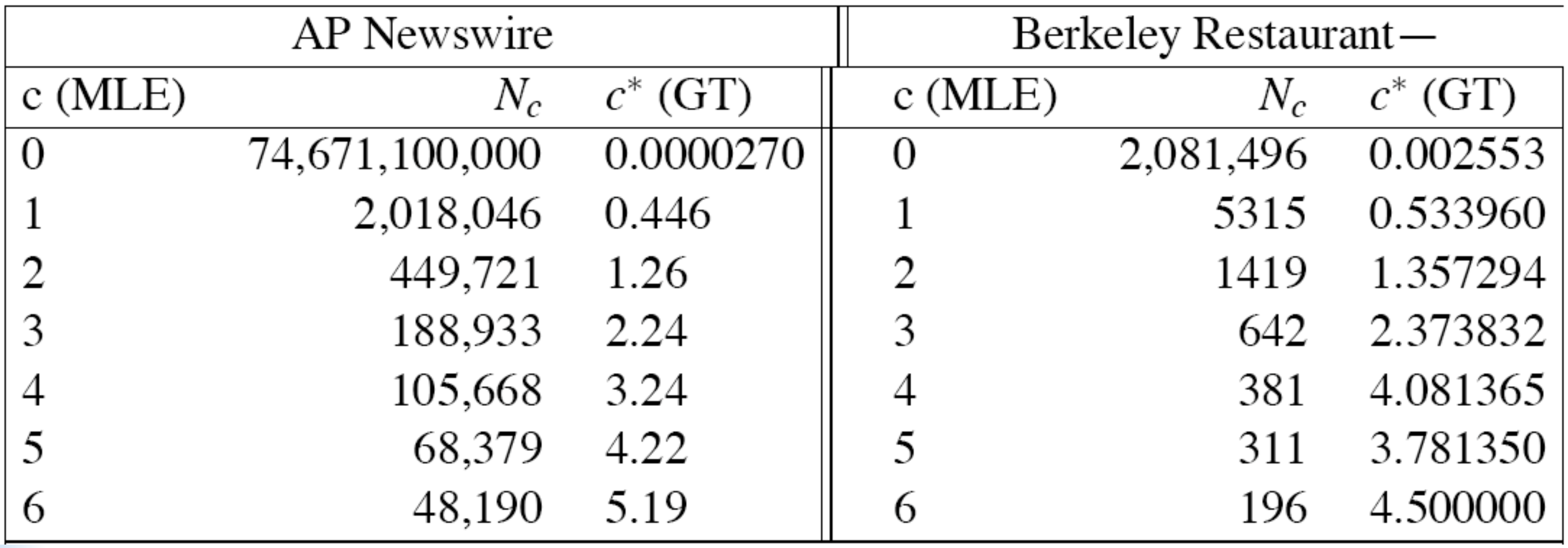
$$c^* = (c+1)\frac{N_{c+1}}{N_c}$$

C*(eel) = c*(1) = (1+1) 1/ 3 = 2/3

Slide from Josh Goodman

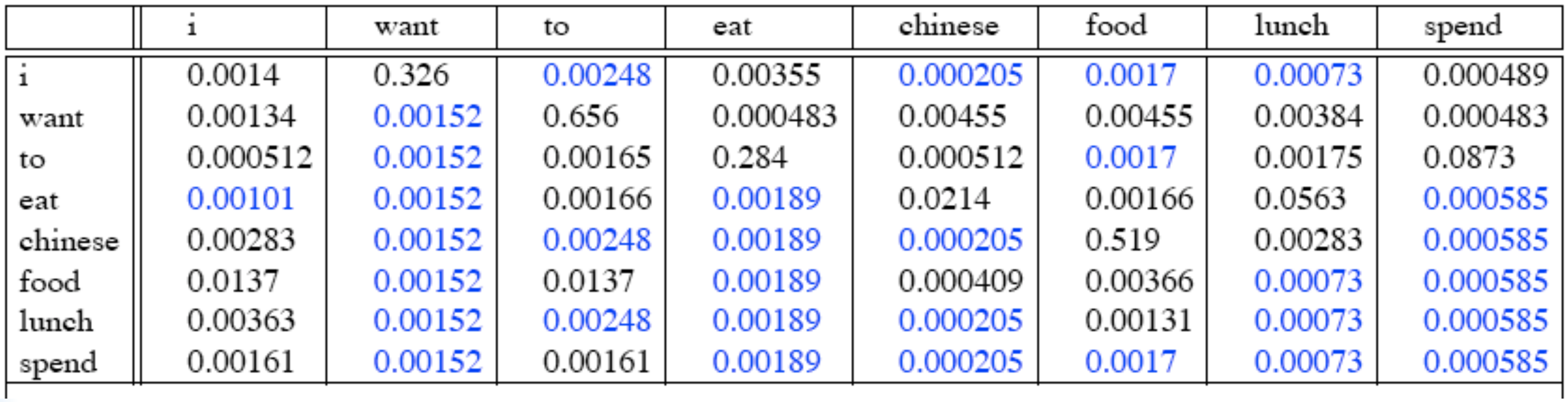
	unseen (bass or catfish)	trout
c	0	1
MLE p	$p = \frac{0}{18} = 0$	$\frac{1}{18}$
c^*		$c^*(\text{trout}) = 2 \times \frac{N_2}{N_1} = 2 \times \frac{1}{3} = .67$
GT p^*_{GT}	$p^*_{\text{GT}}(\text{unseen}) = \frac{N_1}{N} = \frac{3}{18} = .17$	$p^*_{\text{GT}}(\text{trout}) = \frac{.67}{18} = \frac{1}{27}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

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

GT Smoothed Bigram 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	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

Complications

- In practice, assume large counts (c>k for some k) are reliable:

$$c^* = c \ \text{ for } c > k$$

- Also: we assume singleton counts c=1 are unreliable, so treat N-grams with count of 1 as if they were count=0
- Also, need the Nk to be non-zero, so we need to smooth (interpolate) the Nk counts before computing c* from them

Backoff and Interpolation

- Another really useful source of knowledge
- If we are estimating:
  - trigram p(z|xy)
  - but c(xyz) is zero
- Use info from:
  - Bigram p(z|y)
- Or even:
  - Unigram p(z)
- How to combine the trigram/bigram/ unigram info?

Backoff versus 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** 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

Katz Backoff

$$P_{\text{katz}}(w_n|w_{n-N+1}^{n-1}) = \begin{cases} P^*(w_n|w_{n-N+1}^{n-1}), & \text{if } C(w_{n-N+1}^{n}) > 0 \\ \alpha(w_{n-N+1}^{n-1})P_{\text{katz}}(w_n|w_{n-N+2}^{n-1}), & \text{otherwise.} \end{cases}$$

$$P_{\text{katz}}(z|x,y) = \begin{cases} P^*(z|x,y), & \text{if } C(x,y,z) > 0 \\ \alpha(x,y)P_{\text{katz}}(z|y), & \text{else if } C(x,y) > 0 \\ P^*(z), & \text{otherwise.} \end{cases}$$

$$P_{\text{katz}}(z|y) = \begin{cases} P^*(z|y), & \text{if } C(y,z) > 0 \\ \alpha(y)P^*(z), & \text{otherwise.} \end{cases}$$

Why discounts P* and alpha?

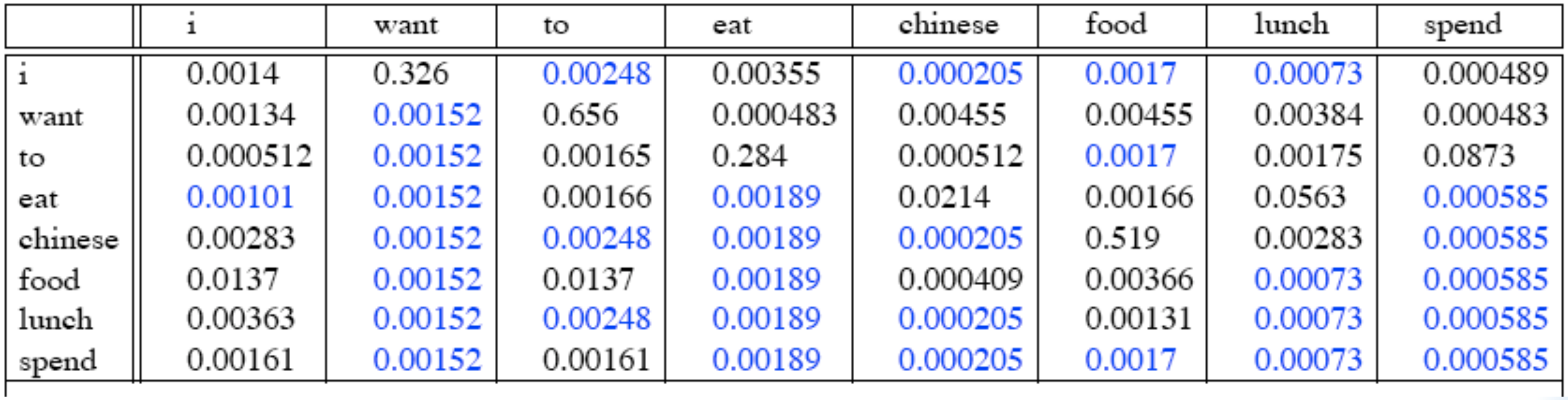
- MLE probabilities sum to 1

$$\sum_i P(w_i|w_j w_k) = 1$$

- So if we used MLE probabilities but backed off to lower order model when MLE prob is zero
- We would be adding extra probability mass
- And total probability would be greater than 1

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

GT smoothed bigram probs



	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	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

Intuition of backoff +discounting

- How much probability to assign to all the zero trigrams?
  - Use GT or other discounting algorithm to tell us
- How to divide that probability mass among different contexts?
  - Use the N-1 gram estimates to tell us
- What do we do for the unigram words not seen in training?
  - **Out Of Vocabulary** = OOV words

ARPA format

unigram:	$\log p^*(w_i)$	w_i	$\log \alpha(w_i)$
bigram:	$\log p^*(w_i \mid w_{i-1})$	$w_{i-1} w_i$	$\log \alpha(w_{i-1} w_i)$
trigram:	$\log p^*(w_i \mid w_{i-2}, w_{i-1})$	$w_{i-2} w_{i-1} w_i$	

```
\data\
ngram 1=1447
ngram 2=9420
ngram 3=5201

\1-grams:
-0.8679678   </s>
-99          <s>                             -1.068532
-4.743076    chow-fun                        -0.1943932
-4.266155    fries                           -0.5432462
-3.175167    thursday                        -0.7510199
-1.776296    want                            -1.04292
...

\2-grams:
-0.6077676   <s>   i                         -0.6257131
-0.4861297   i     want                      0.0425899
-2.832415    to    drink                     -0.06423882
-0.5469525   to    eat                       -0.008193135
-0.09403705  today </s>
...

\3-grams:
-2.579416    <s>   i         prefer
-1.148009    <s>   about     fifteen
-0.4120701   to    go        to
-0.3735807   me    a         list
-0.260361    at    jupiter   </s>
-0.260361    a     malaysian restaurant
...
\end\
```

Summary

- Language Modeling (N-grams)
  - N-gram Intro
  - The Shannon Visualization Method
  - Evaluation:
    - Perplexity
  - Smoothing:
    - Laplace (Add-1)
    - Add-k
    - Good-Turing
    - Backoff
    - Interpolation