CS114 Lecture 5

Ngrams

January 29, 2014

Professor Meteer

Thanks for Jurafsky & Martin & Prof. Pustejovksy for slides

Probabilities of two events

- Joint probability
  - If two events A and B are **independent** then
    - P(A and B) = P(A) x P(B)
  - What is the probability that a card is red and a king?
    - P(king) = 4 / 52 = 1/13
    - P(red) = 26 / 52 = ½

$$1/13 \times 1/2 = 1/26$$

- Conditional probability
  - the probability of event B occurring given event A occurs
  - What is the probability that a card is a king if I know that it is red?
    - P(king | red) =

$$\frac{P(king \cap red)}{P(red)} = \frac{1/26}{26/52} = \frac{1}{13}$$

The Chain Rule

- Recall the definition of conditional probabilities

- Rewriting:

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

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

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

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(*the* | *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.

The Chain Rule

- Recall the definition of conditional probabilities

- Rewriting:

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

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

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

Markov Independence Assumption

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 (MLE)

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

Shannon's Method

- 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 first.
- 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>

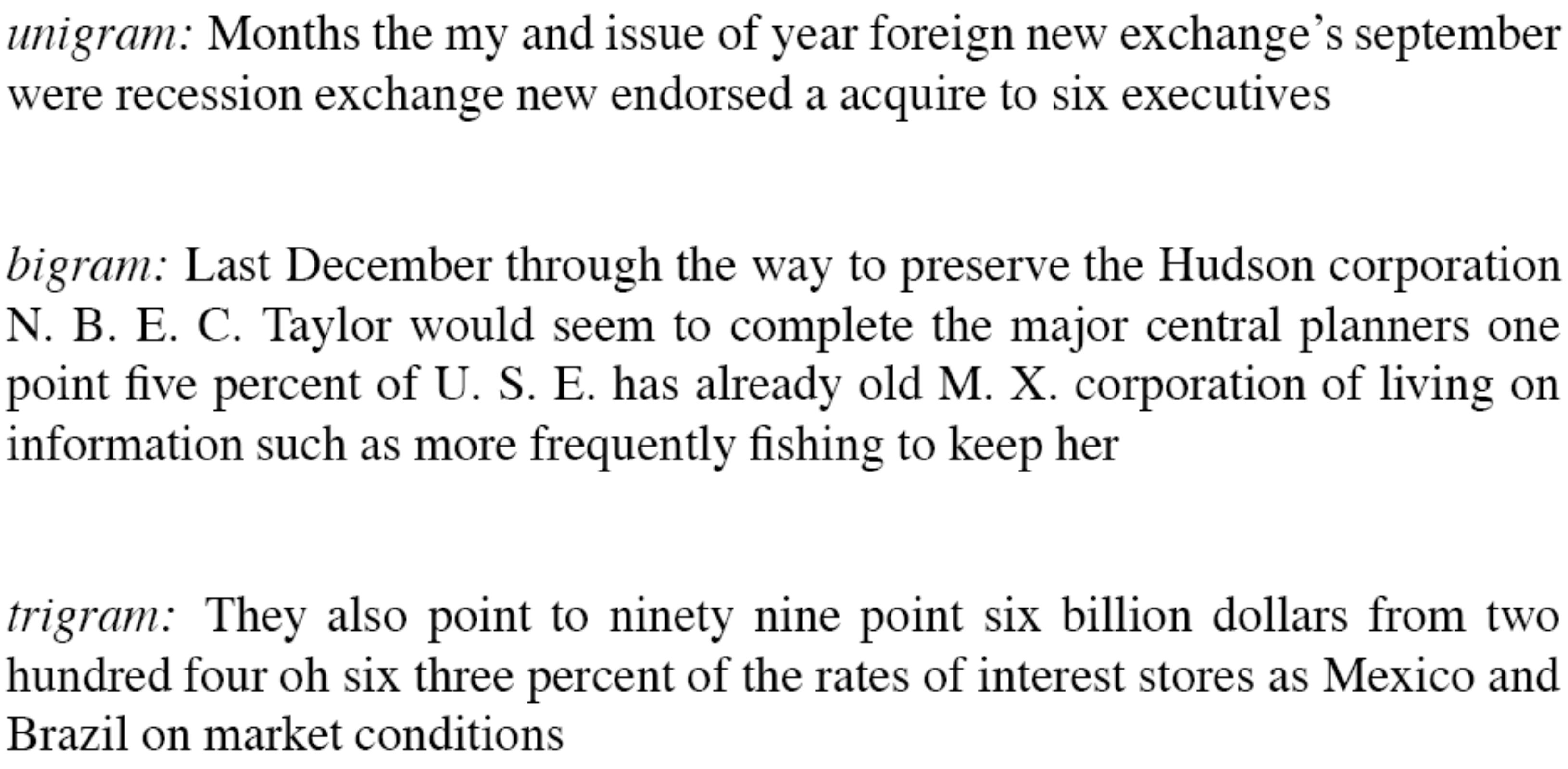
Shakespeare

- Unigrams
  - To him swallowed confess hear both. Which. Of save on trial for are ay device and rote life have c
  - Hill he late speaks; or! A more or legless first you enter
- Bigrams
  - What means, sir. I confess she? Then all sorts, he is trim, captain.
  - Why doest stand forth they canopy, forsooth he is this palpable hit the King Henry. Live king. Follow.
- Trigrams
  - Sweet prince, Falstaff shall die. Harry of Monmouths grave
  - This shall forbid it should be branded, if renown made it empty
- Quadrigrams
  - King Henry. What! I will go seek the traitor Gloucester. Exeunt some of the watch. A great banquet serv'd in;
  - Willyou not tell me who I am?
  - It cannot be but so.

Shakespeare as a Corpus

- N=884,647 tokens, V=29,066 types (vocabulary)
- 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)
  - This is the biggest problem in language modeling; we'll come back to it.
- Quadrigrams are worse: What's coming out looks like Shakespeare because it ***is*** Shakespeare

The Wall Street Journal is Not Shakespeare



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

Evaluation

- How do we know if our models are any good?
  - And in particular, 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 the models 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)

Unknown Words

- But once we start looking at test data, we'll run into words that we haven't seen before (pretty much regardless of how much training data you have.
- With an *Open Vocabulary* task
  - Create an unknown word token <UNK>
  - Training of <UNK> probabilities
    - Create a fixed lexicon L, of size V
      - From a dictionary or
      - A subset of terms from the training set
    - At text normalization phase, any training word not in L changed to <UNK>
    - Now we count that like a normal word
  - At test time
    - Use UNK counts for any word not in training

Perplexity

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

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

- Minimizing perplexity is the same as maximizing probability
  - **The best language model is one that best predicts an unseen test set**

Perplexity

- Chain rule:

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

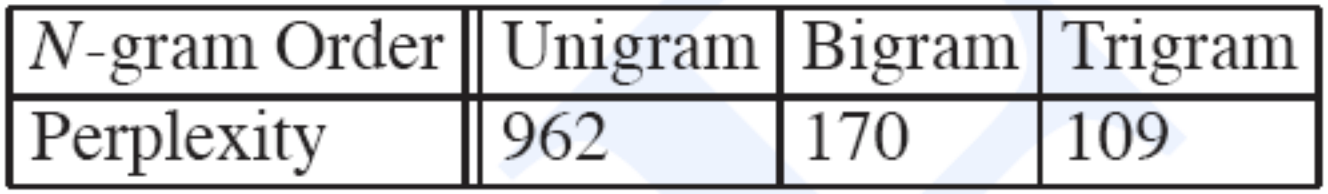
- For bigrams:

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

- 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, WSJ



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***

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.

Example: Linguistic Segmentation

- Acoustic segmentations
  - I'm not sure how many active volcanoes there are now and and what the amount of material that they do
  - <s> uh <s> put into the atmosphere
  - <s> ! think probably the greatest cause is uh
  - <s> vehicles
  - <s> especially around cities <s>
- Linguistic segmentations
  - I'm not sure how many active volcanoes there are now and and what the amount of material that they do uh put into the atmosphere
  - <s> I think probably the greatest cause is uh vehicles especially around cities

Compare perplexity

- Build three models

Test		**Training**	
	Acoustic Seg	Ling Seg	No Seg
Acoustic Seg	105	111	
Ling Seg	89	78	
No Seg	163	174	130

Zero Counts

- Back to Shakespeare
  - Recall that 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)
  - Does that mean that any sentence that contains one of those bigrams should have a probability of 0?

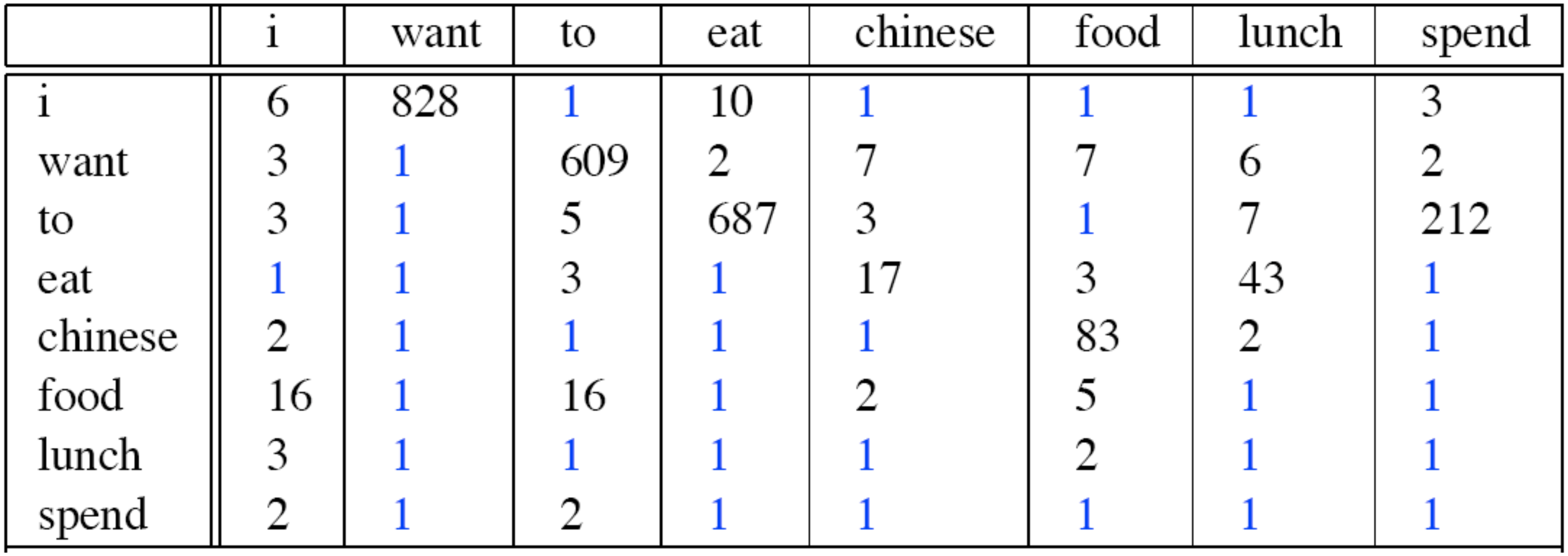
Zero Counts

- 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
- 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



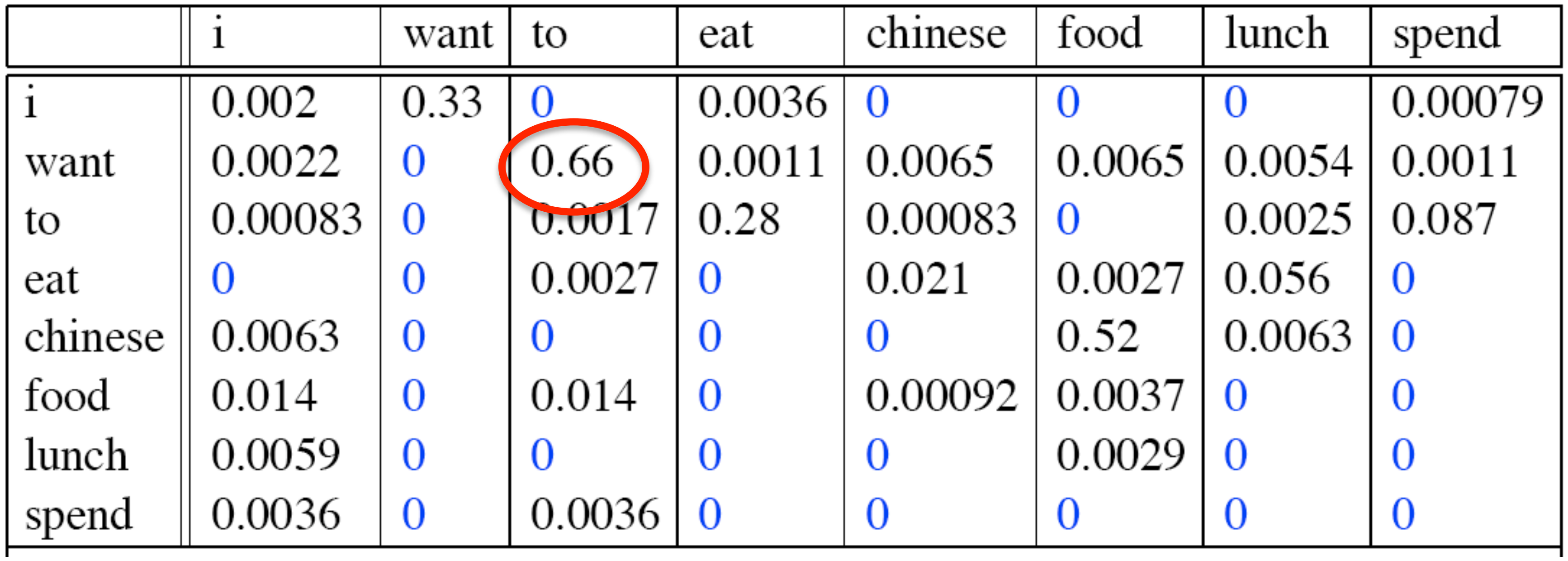
	i	want	to	eat	chinese	food	lunch	spend
i	6	828	1	10	1	1	1	3
want	3	1	609	2	7	7	6	2
to	3	1	5	687	3	1	7	212
eat	1	1	3	1	17	3	43	1
chinese	2	1	1	1	1	83	2	1
food	16	1	16	1	2	5	1	1
lunch	3	1	1	1	1	2	1	1
spend	2	1	2	1	1	1	1	1

Laplace-Smoothed 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	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

Original 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

Reconstituted Counts

[illegible]

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

	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

Big Change to the Counts!

- C(want 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
  - Could use more fine-grained method (add-k)
- 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.

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

- 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 the next fish caught is from a new species (one not seen in our previous catch)?
  - 3/18
- Assuming so, how likely is it that next species is trout?
  - Must be less than 1/18

Slide adapted from Josh Goodman

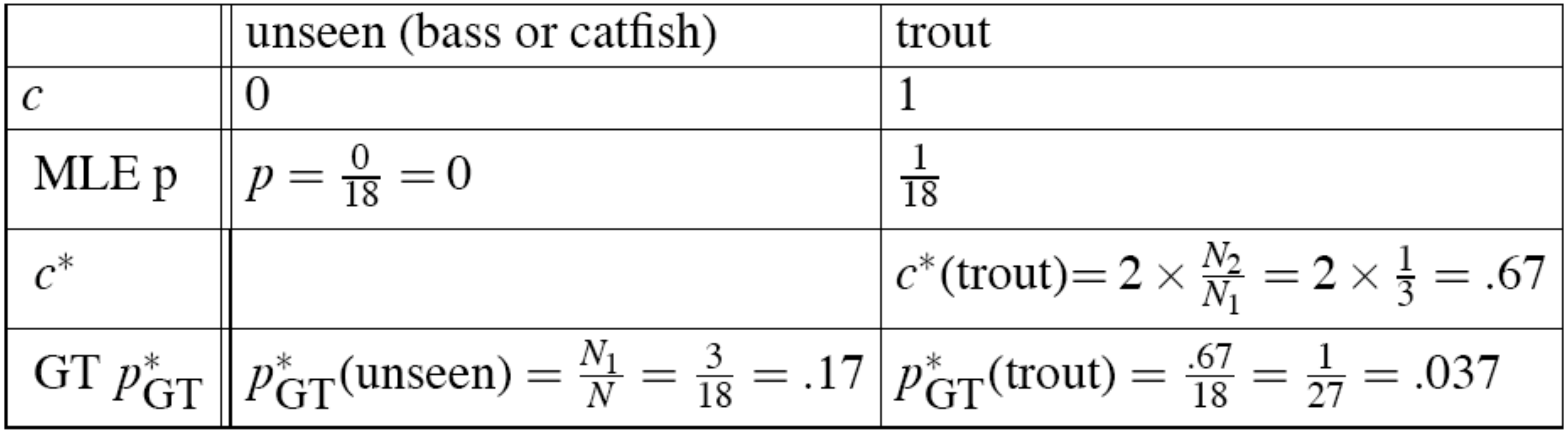
Good-Turing

- Notation: N_x is the frequency-of-frequency-x
  - So $N_{10}=1$
    - Number of fish species seen 10 times is 1 (carp)
  - $N_1=3$
    - Number of fish species seen 1 is 3 (trout, salmon, eel)
- To estimate total number of unseen species
  - Use number of species (words) we've seen once
  - $c_0^* = c_1$ $p_0 = N_1/N$

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

- All other estimates are adjusted (down) to give probabilities for unseen

GT Fish Example



	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 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
  - $c_0^* = c_1$ $p_0 = N_1/N$ $p_0=N_1/N=3/18$

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

- All other estimates are adjusted (down) to give probabilities for unseen

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

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

GT Fish Example

- OR use the 1s for 0s (3/18 spread over 2 species)
- AND Look at the things that happened 2s to share with 1s
  - C(whitefish) = 2 happened once
  - Discount 1s by 2/3
- LOTS OF ALTERNATIVES! Just estimates

	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}	$p^*_{\text{GT}}(\text{unseen}) = \frac{N_1}{N} = \frac{3}{18} = .17$	$p^*_{\text{GT}}(\text{trout}) = \frac{.67}{18} = \frac{1}{27} = .037$

Could just spread 1s over 0s

Carp	10	10
Perch	3	3
WF	2	2
Trout	1	1
Salmon	1	1
Eel	1	1
Catfish	0	1
Bass	0	1
TOTAL	18	

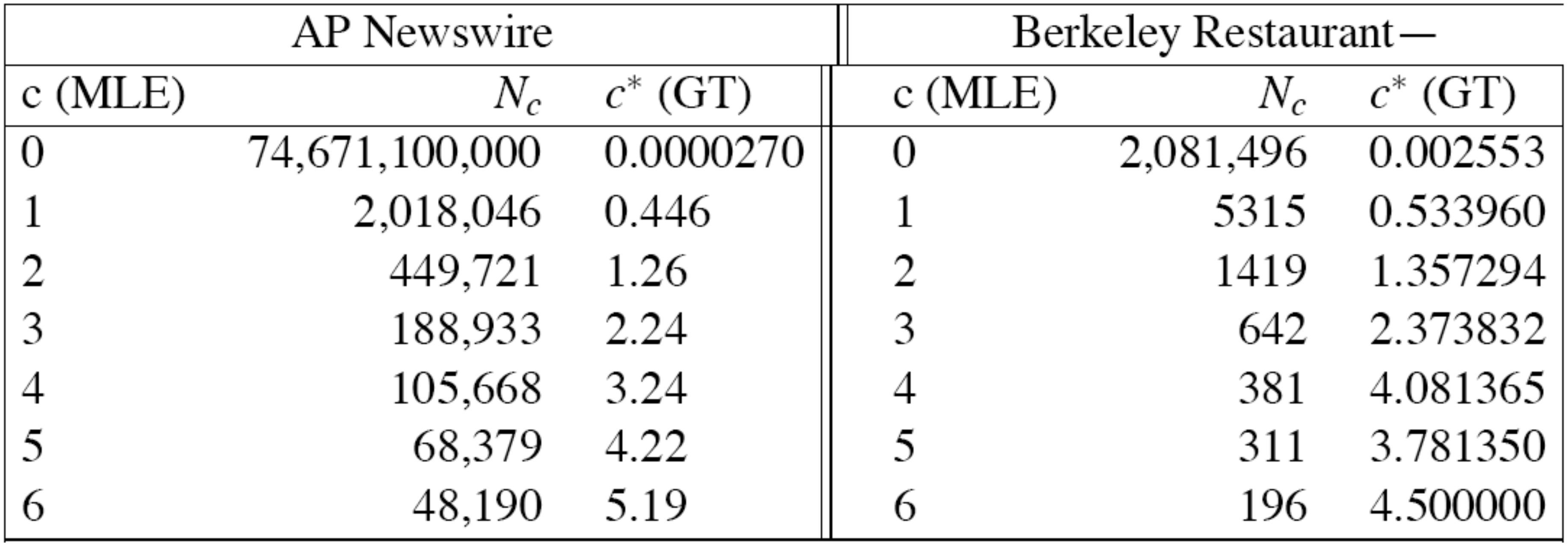
- Prob of things that occurred once

  1\18 + 1\18 + 1\18 = 3\18
- Add one to zero counts
- Spread probability over 1s and 0s
- (3/18) / 5 = .066

Practical considerations for Good- Turing

- The new estimations of c (c*) are dependent on the counts of c+1
  - But some can be 0
    - First use linear regression to smooth counts
  - Higher counts are more likely to be right
    - Only apply this to counts <=5
  - Low counts can be noise
    - Treat counts of 1 as if they were 0
- Good-Turing is never used alone—always combined with back-off and interpolation

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

Backoff and Interpolation

- Another really 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

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?

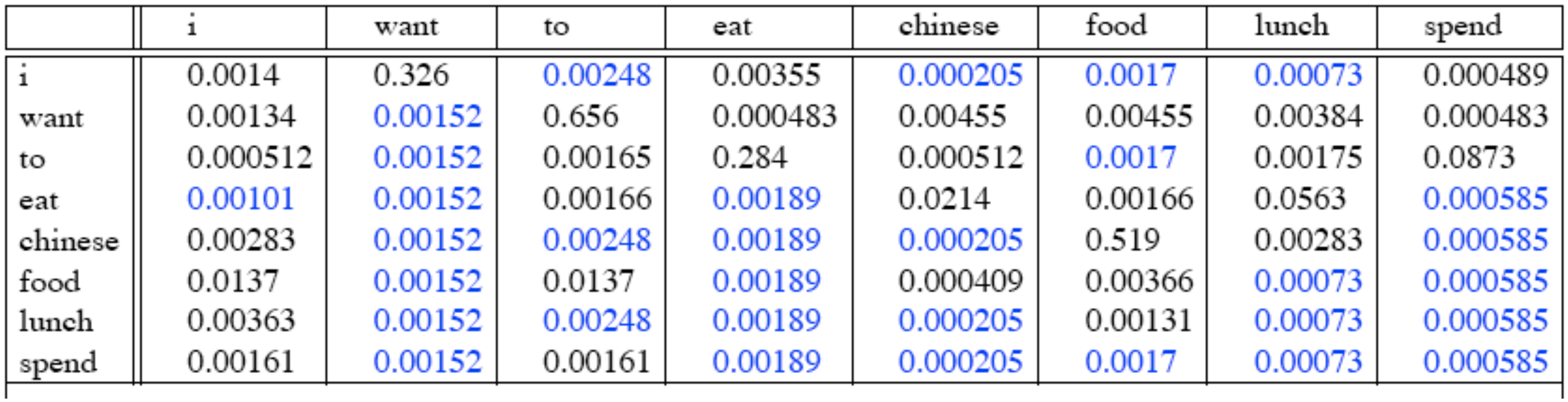
- MLE probabilities sum to 1

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

- 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 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

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

OOV words: <UNK> word

- **Out Of Vocabulary** = OOV words
- We don’t use GT smoothing for these
  - Because GT assumes we know the number of unseen events
- 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

Practical Issues

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

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