Basic Text Analysis

N-Gram Models

Slides adapted from Jurafsky & Martin, Speech and Language Processing

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 fairly simple counts.

Word Prediction

- We can formalize this task using *N*-gram models.
- *N*-grams are token sequences of length *N*.
- Our earlier example contains these 2-grams:

  (I notice)

  (notice three)

  (three guys)

  (guys standing)

  (standing on)

  (on the)
- Given knowledge of N-gram counts, we can guess likely next words in a sequence.

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.
- It lies at the core of the following applications:
  - Automatic speech recognition
  - Handwriting and character recognition
  - Spelling correction
  - Part-of-speech tagging
  - Machine translation
  - And many more

Counting

- Simple counting lies at the core of any probabilistic approach. So let’s first take a look at what we’re counting.

  He stepped out into the hall, was delighted to encounter a water brother.

  - 13 tokens, 15 if we include “,” and “.” as separate tokens.
  - Assuming we include the comma and period, how many bigrams are there?

Counting: Types and Tokens

- How about

  They picnicked by the pool, then lay back on the grass and looked at the stars.

  - 18 tokens (again counting punctuation)
- But we might also note that "the" is used 3 times, so there are only 16 unique types (as opposed to tokens).
- In going forward, we'll have occasion to focus on counting both types and tokens of both words and *N*-grams.

Counting: Wordforms

- Should “cats” and “cat” count as the same when we’re counting?
- How about “geese” and “goose”?
- Remember:
  - Lemma: a set of lexical forms having the base form and major part of speech
  - Wordform: fully inflected surface form
- Again, we’ll have occasion to count both lemmas and wordforms

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 prediction of the conditional probability of a word given previous words in the sequence:

  $P(w_n|w_1,w_2 \ldots w_{n-1})$
- We'll call a statistical model that can do 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.
- Remember:

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

Not That Simple?

- 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.
- Unfortunately, for most sequences and corpora we won't get good estimates from this method.
  - What we're likely to get is 0. Or worse 0/0.
- Clearly, we'll have to be a little more clever.
  - Let's use the chain rule of probability
  - And a particularly useful independence assumption.

The Chain Rule

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

P(its water was so transparent) =

P(its) * P(water|its) * P(was|its water) *
P(so|its water was) * P(transparent|its water was so)

Independence Assumption

- Make the simplifying assumption:

  P(lizard|the,other,day,I,was,walking,along,and,saw,a) = P(lizard|a)

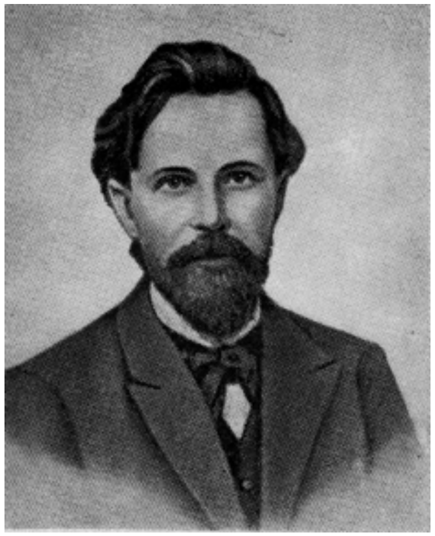
- Or maybe:

  P(lizard|the,other,day,I,was,walking,along,and,saw,a) = P(lizard|saw,a)

- That is, assume the probability in question is independent of its earlier history

Independence Assumption

- This particular kind of independence assumption is called a Markov assumption after the Russian mathematician Andrei Markov.



Markov Assumption

- So for each component in the product use 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)

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

An Example

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

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

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

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

Maximum Likelihood Estimates

- Maximum likelihood estimate:
  - 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 another text from the same distribution is “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 1M word corpus.

Berkeley Restaurant Project

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

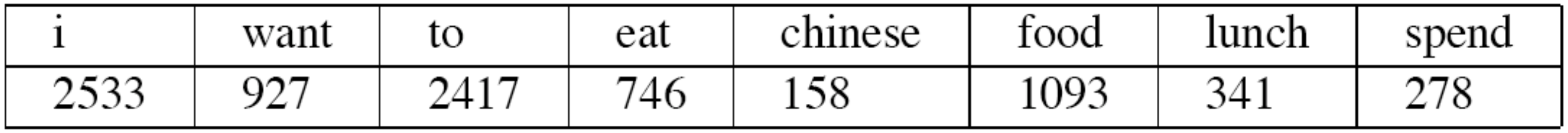
Bigram Counts

- Corpus of 9222 sentences
  - For example: "I want" occurred 827 times

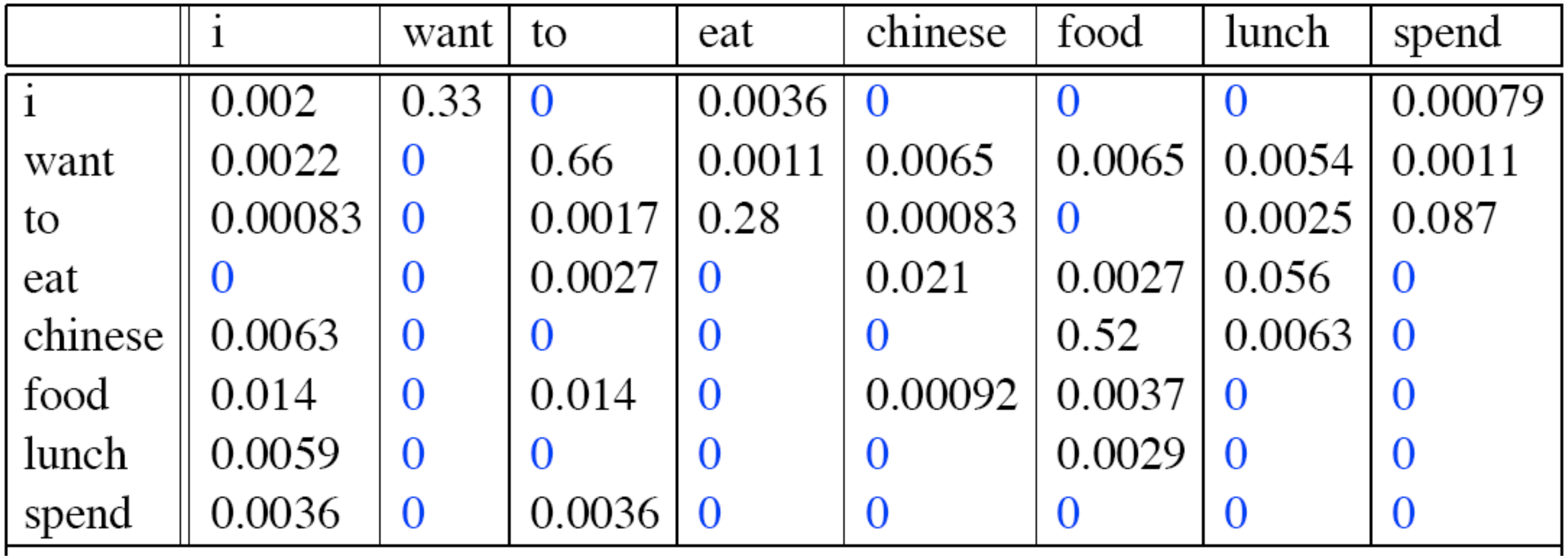
	i	want	to	eat	chinese	food	lunch	spend
i	5	827	0	9	0	0	0	2
want	2	0	608	1	6	6	5	1
to	2	0	4	686	2	0	6	211
eat	0	0	2	0	16	2	42	0
chinese	1	0	0	0	0	82	1	0
food	15	0	15	0	1	4	0	0
lunch	2	0	0	0	0	1	0	0
spend	1	0	1	0	0	0	0	0

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

Sentence Probabilities

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

P(i|<s>) * P(want|I) * P(english|want) *

P(food|english) * P(</s>|food) * = 0.000031

Kinds of Knowledge

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

$$\left.\begin{array}{l} P(\text{english} \mid \text{want}) = .0011 \\ P(\text{chinese} \mid \text{want}) = .0065 \end{array}\right\} \text{World knowledge}$$

$$\left.\begin{array}{l} P(\text{to} \mid \text{want}) = .66 \\ P(\text{eat} \mid \text{to}) = .28 \\ P(\text{food} \mid \text{to}) = 0 \\ P(\text{want} \mid \text{spend}) = 0 \end{array}\right\} \text{Syntax}$$

$$P(\text{i} \mid \text{<s>}) = .25 \quad \text{Discourse}$$

Shannon's Method

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

Shannon’s Method

- Start:

  Sample a random bigram (<s>, w) according P(w | <s>)

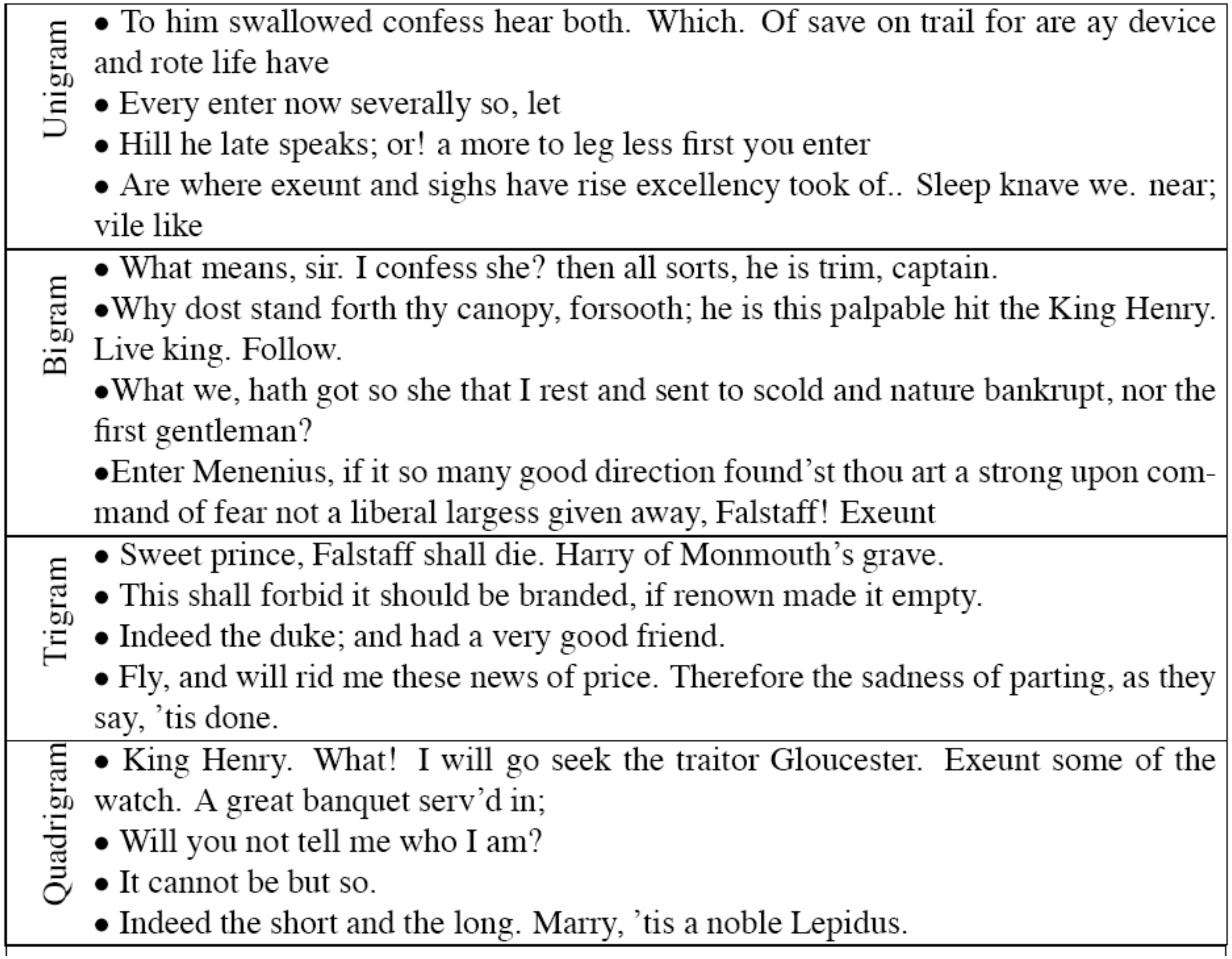
- Repeat:

  Sample a random bigram (w, x) according to P(x | w), the prefix w matches the suffix of the previous bigram

- Until:

  We randomly choose a (y, </s>)

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? •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.

Shakespeare as a Corpus

- Statistics:
  - Tokens N = 884,647
  - Types V = 29,066
  - Possible bigram types = V^2 = 844,832,356
  - Actual bigram types ≈ 300,000
- Note:
  - 99.96% of the possible bigrams were never seen (have zero entries in the table)
  - This is the biggest problem in language modeling.
- 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?
  - In particular, how do we know if one model is better than another?
- Shannon' s game gives us an intuition.
  - The generated texts from the higher order models sure look better. That is, they look more like the text the model was obtained from.
  - But what does that mean? Can we make that notion operational?

Evaluation

- Fit parameters of our model on a training set.
- Look at the 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 different than our training set, but 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 (more later)

Unknown Words

- But once we start looking at test data, we'll run into words that we haven't seen before
- 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 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:

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

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

- Chain rule:

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

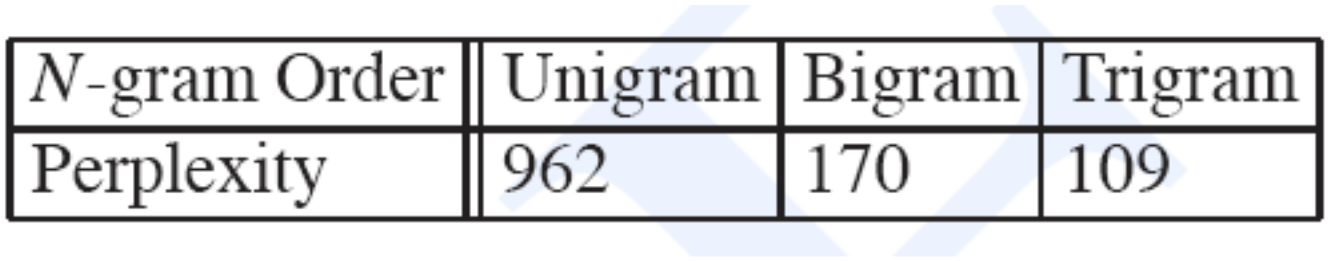
- For bigrams:

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

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

Lower perplexity – better model

- Wall Street Journal:
  - Training 38 million words
  - Test 1.5 million words



N-gram Order	Unigram	Bigram	Trigram
Perplexity	962	170	109

Extrinsic Evaluation

- The best evaluation is often extrinsic:
  - Put the model into an application (for example, ASR)
  - Evaluate performance of the application with model A
  - Put model B into the application and evaluate
  - Compare performance of the application with A and B
- However:
  - This is really time-consuming
  - Can take days to run an experiment
  - In practice, we often fall back on intrinsic evaluation, for example, using perplexity

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
- Data sparseness:
  - Our estimates are sparse! We have no counts at all for the vast bulk of things we want to estimate!

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

Reconstituted Counts

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

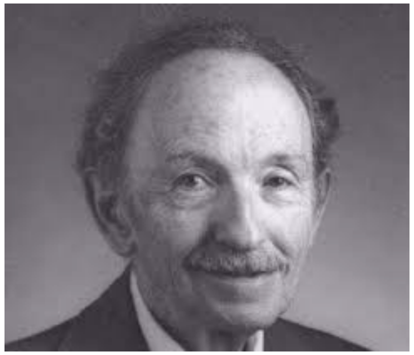
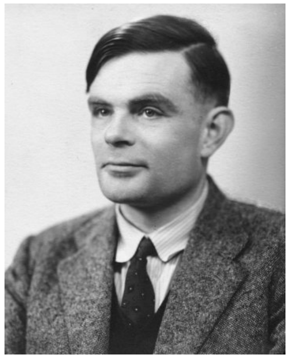
	i	want	to	eat	chinese	food	lunch	spend
i	3.8	527	0.64	6.4	0.64	0.64	0.64	1.9
want	1.2	0.39	238	0.78	2.7	2.7	2.3	0.78
to	1.9	0.63	3.1	430	1.9	0.63	4.4	133
eat	0.34	0.34	1	0.34	5.8	1	15	0.34
chinese	0.2	0.098	0.098	0.098	0.098	8.2	0.2	0.098
food	6.9	0.43	6.9	0.43	0.86	2.2	0.43	0.43
lunch	0.57	0.19	0.19	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

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 is 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
- Use the count of things we've seen once to help estimate the count of things we've never seen

Good-Turing

- Imagine you are fishing
  - There are 8 species: carp, perch, pike, trout, salmon, eel, catfish, bass
- You have caught
  - 10 carp, 3 perch, 2 pike, 1 trout, 1 salmon, 1 eel = 18 fish
- How likely is it that the next catch is a new species?
  - 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

  $N_{10} = 1$ (carp)

  $N_1 = 3$ (trout, salmon, eel)

- To estimate total number of unseen species

  Use number of species (words) we've seen once

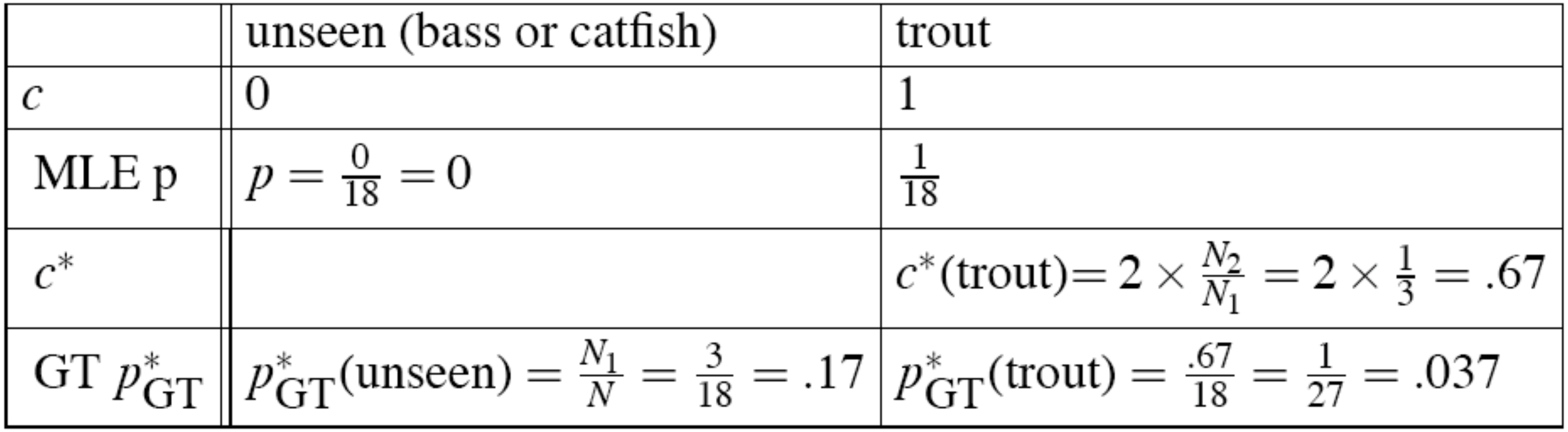
  P(unseen) = $N_1/N = 3/18$

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

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

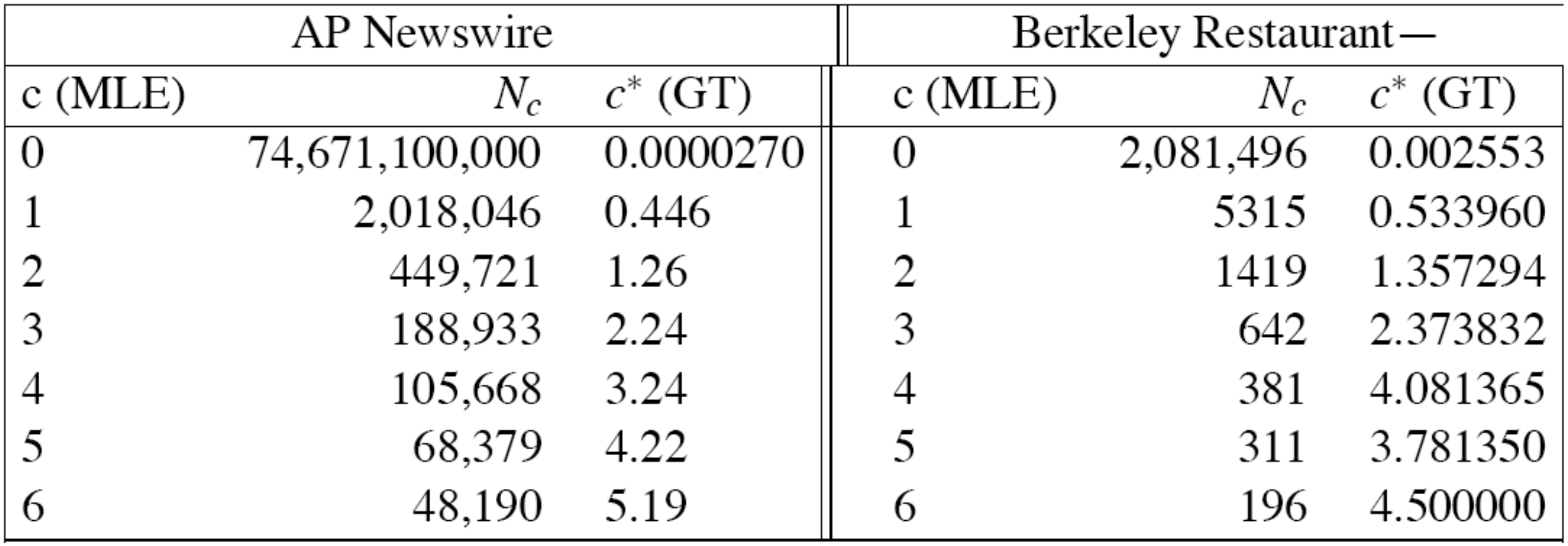
Slide adapted from Josh Goodman

GT Fish Example



	unseen (bass or catfish)	trout
c	0	1
MLE p	$p = \frac{0}{18} = 0$	$\frac{1}{18}$
c^*		c^*(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

Complications

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

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

- That complicates c*, making it:

$$c^* = \frac{(c+1)\frac{N_{c+1}}{N_c} - c\frac{(k+1)N_{k+1}}{N_1}}{1 - \frac{(k+1)N_{k+1}}{N_1}}, \quad \text{for } 1 \leq c \leq k.$$

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

Backoff and Interpolation

- Another really useful source of knowledge
  - If we are estimating trigram P(z|xy)
  - 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

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

$$\sum_i \lambda_i = 1$$

- 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
  - Fix the *N*-gram probabilities
  - Then search for lambda values that when plugged into previous equation give best probability for held-out set
  - Can use EM to do this search (more later)

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?

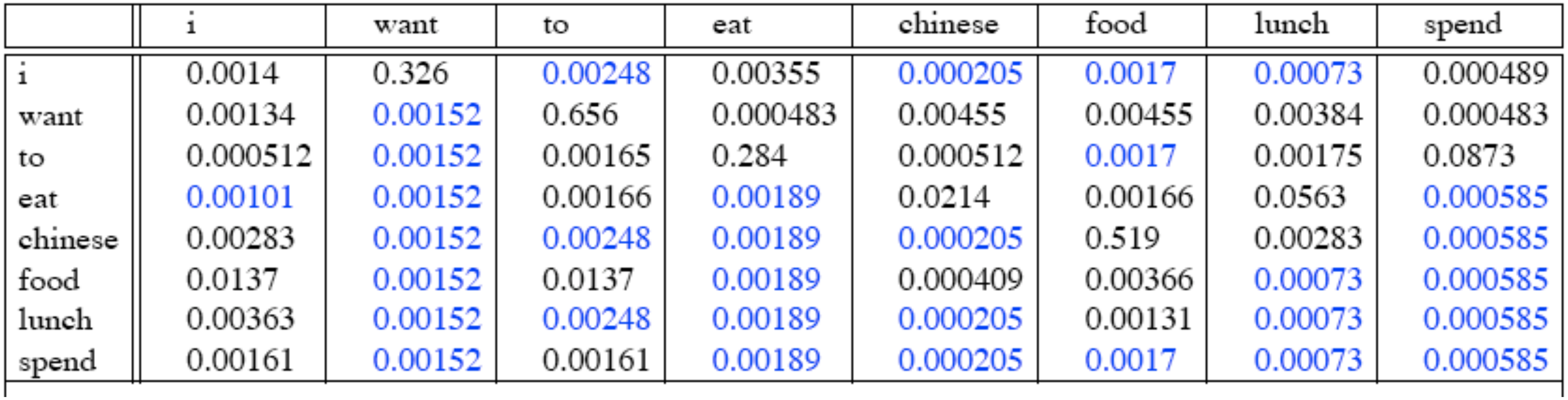
- MLE probabilities sum to 1

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

- What if we use MLE probabilities but back off to lower order model when MLE probability is zero?
- We would then be adding extra probability mass
- And total probability would be greater than 1

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

Bigram Probabilities with Backoff and GT



	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

Backoff + Discounting

- How much probability to assign to all zero trigrams?
  - Use GT or other discounting method to tell us
- How to divide that probability mass among different contexts?
  - Use the N-1-gram estimates to tell us (backoff)
- What do we do with 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
  - 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
  - Adding is faster than multiplying

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