CS4442/9542b Artificial Intelligence II prof. Olga Veksler

Lecture 14

Natural Language Processing

Language Models

Many slides from: Joshua Goodman, L. Kosseim, D. Klein, D. Jurafsky

Outline

- Why we need to model language
- Probability background
  - Basic probability axioms
  - Conditional probability
  - Chain Rule
- n-gram model
- Parameter Estimation Techniques
  - MLE
  - Smoothing
    - Add-One, add-Delta
    - Good-Turing
  - Interpolation

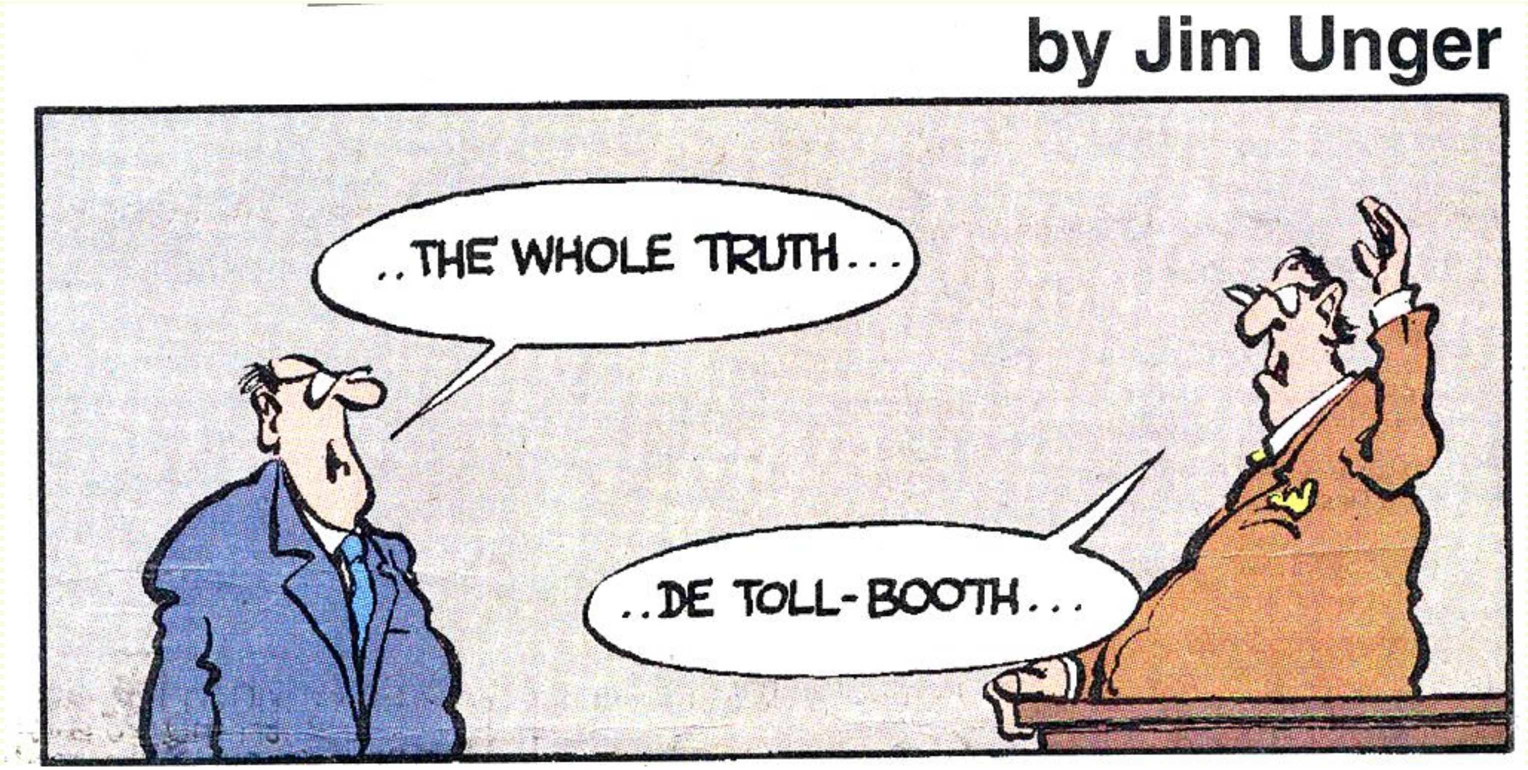
Why Model Language?

- Design probability model **P**() such that
- Spell checker:
  - **P**(I think there are OK) < **P**(I think they are OK)
- Speech recognition:
  - **P**(lie cured mother) < **P**(like your mother)
- Optical character recognition
  - **P**(thl cat) < **P**(the cat)
- Machine translation: "On voit Jon à la télévision"
  - **P**(In Jon appeared TV) < **P**(Jon appeared on TV)
- Many other applications

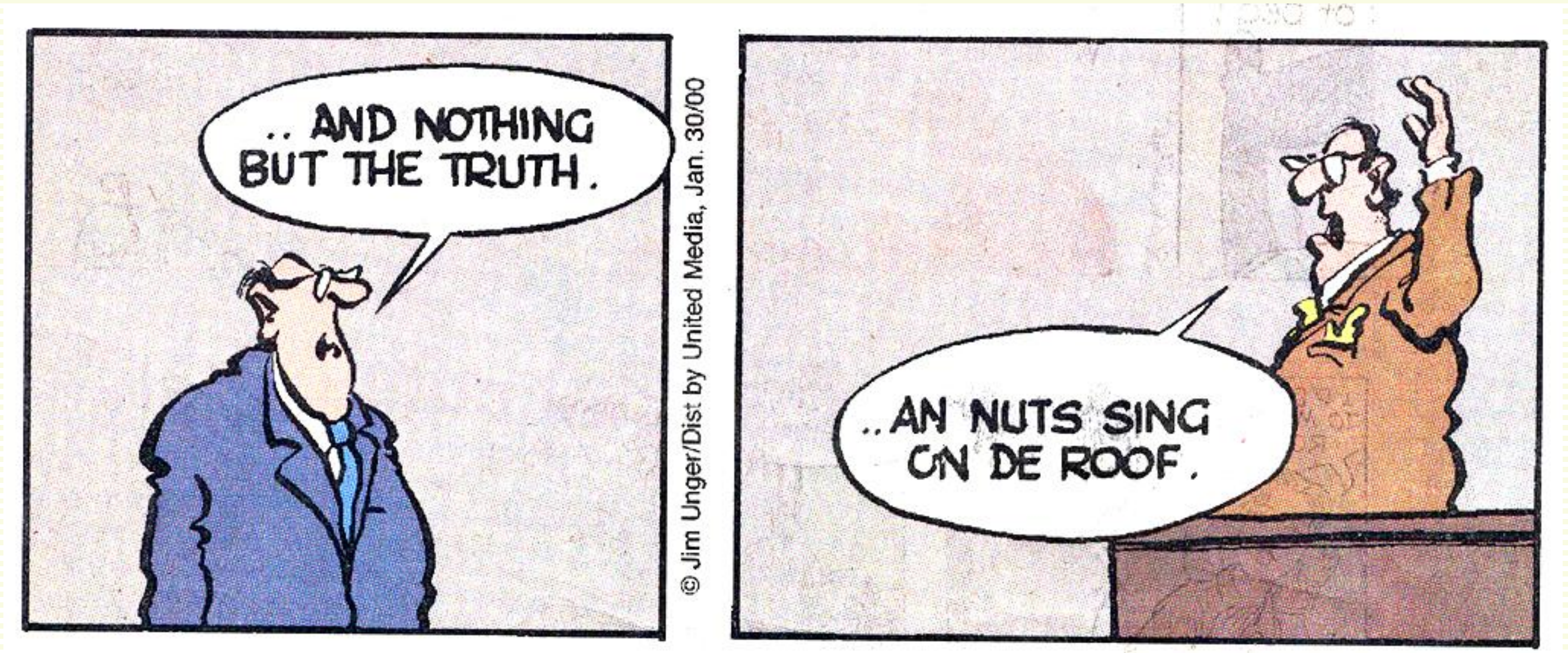
Language Model for Speech Recognition

HERMAN
REPEAT AFTER ME... I SWEAR TO TELL THE TRUTH...
I SWERVE TO SMELL DE SOUP...

Language Model for Speech Recognition



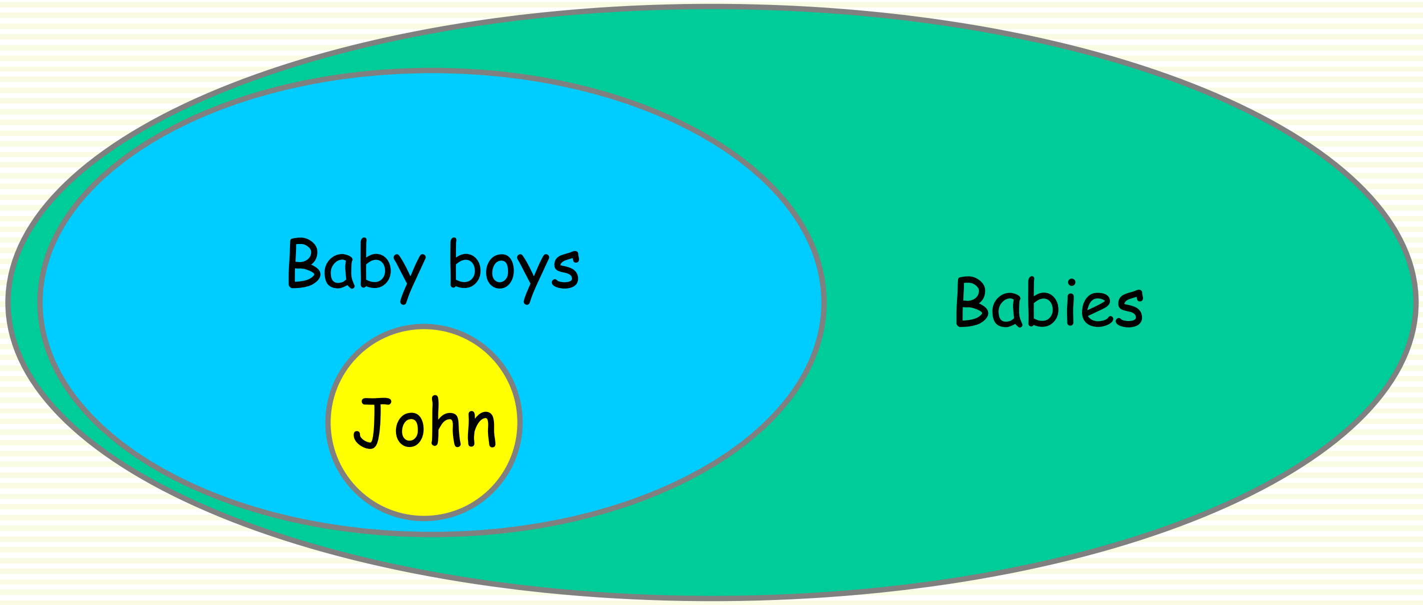
Language Model for Speech Recognition


.. AND NOTHING BUT THE TRUTH.
.. AN NUTS SING ON DE ROOF.
© Jim Unger/Dist by United Media, Jan. 30/00

Language Model for Speech Recognition

Basic Probability

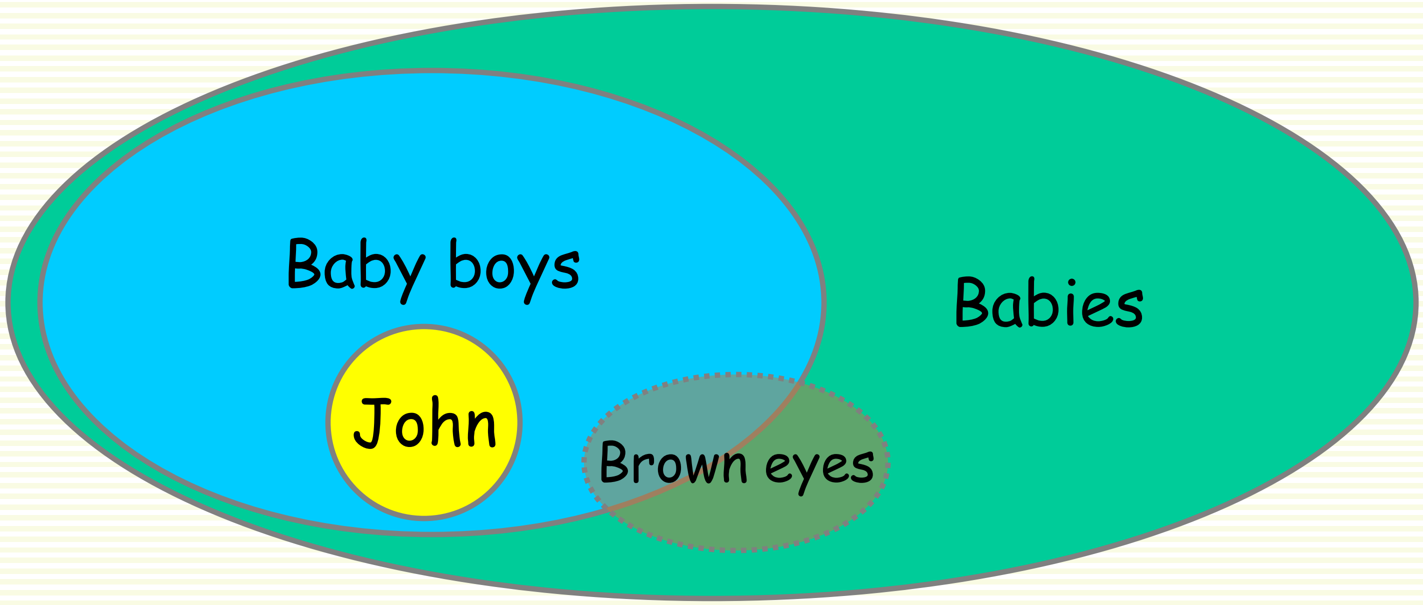
- **P**(X) is probability that X is true
  - **P**(baby is a boy) = 0.5 (1/2 of all babies are boys)
  - **P**(baby is named John) = 0.001 (1 in1000 babies named John)



Joint probabilities

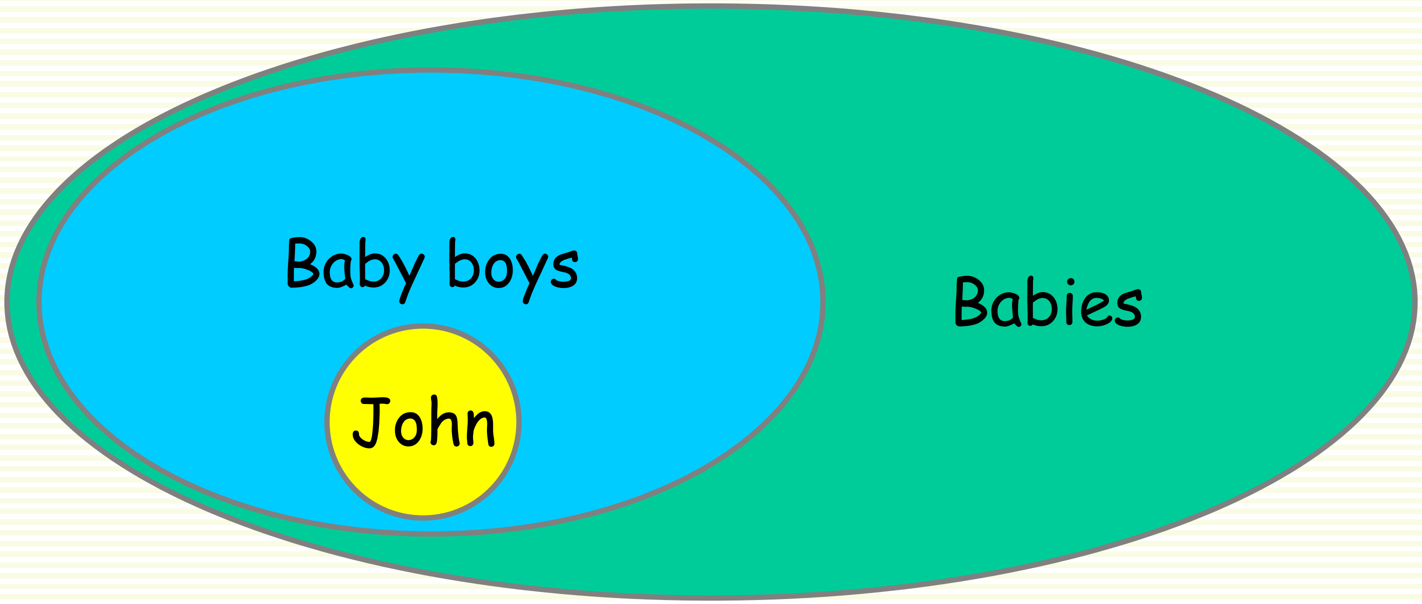
- **P**(X,Y) is probability that X and Y are both true

  P(brown eyes, boy) = (number of all baby boys with brown eyes)/(total number of babies)



Conditional probability

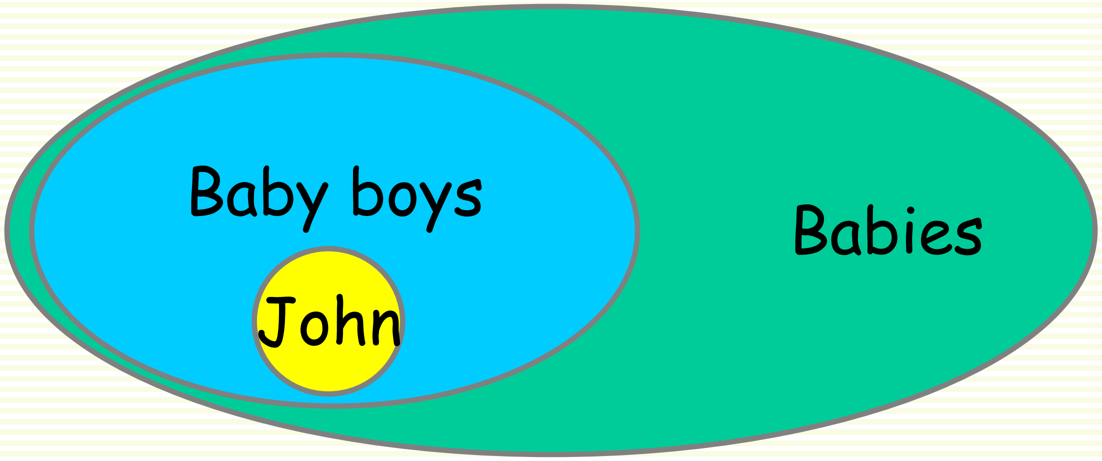
- **P**(X|Y) is probability that X is true when we already know Y is true



Conditional Probability

- **P**(X|Y) = **P**(X, Y) / **P**(Y)
- **P**(baby is named John | baby is a boy) =

$$\frac{\mathbf{P}(\text{baby is named John, baby is a boy})}{\mathbf{P}(\text{baby is a boy})} = \frac{0.001}{0.5} = 0.002$$



- **P**(baby is a boy | baby is named John) = 1

Chain Rule

- From Conditional Probability:

$$\mathbf{P}(X,Y) = \mathbf{P}(Y|X)\ \mathbf{P}(X)$$

- Extend to three events:

$$\mathbf{P}(X,Y,Z) = \mathbf{P}(Y,Z|X)\mathbf{P}(X) = \mathbf{P}(Z|X,Y)\mathbf{P}(Y|X)\mathbf{P}(X)$$

- Extend to multiple events:

$$\mathbf{P}(X_1,X_2,\ldots,X_n) = \mathbf{P}(X_1)\mathbf{P}(X_2|X_1)\mathbf{P}(X_3|X_1X_2)\ldots\mathbf{P}(X_n|X_1,\ldots,X_{n-1})$$

Language Modeling

- Start with vocabulary
  - words vocabulary V = {a, an, apple,…, zombie}
  - or character vocabulary V = {a, A,…., z, Z,*,…, -}
- In LM, events are sequences of words (or characters)
- Example “an apple fell” or “abracadabra!!!+”
- **P**(an apple fell) is the probability of the joint event that
  - the first word in a sequence is “an”
  - the second word in a sequence is “apple”
  - the third word in a sequence is “fell”
- **P**(fell | an apple) is probability that the third word in a sequence is “fell” given that the previous 2 words are “an apple”

Probabilistic Language Modeling

- A language model is a probability distribution over word or character sequences

$$\mathbf{P}(W) = P(w_1 w_2 w_3 w_4 w_5 \ldots w_k)$$

- Want:
  - **P**("And nothing but the truth") ≈ 0.001
  - **P**("And nuts sing on the roof") ≈ 0.000000001
- Related task: probability of an upcoming word:

$$\mathbf{P}(w_5 | w_1, w_2, w_3, w_4)$$

- A model that computes either of these:

$$\mathbf{P}(W) \quad \text{or} \quad \mathbf{P}(w_k | w_1, w_2 \ldots w_{k-1})$$

  is called a **language model**

- Build model **P** from observed texts (corpora)

Probabilistic Language Modeling

- Get lots of training text (corpora)
- Use it to estimate $P(w_1w_2w_3w_4w_5\dots w_k)$:
- Naïve idea: $\mathbf{P}\left(\mathbf{w}_1\mathbf{w}_2\dots\mathbf{w}_\mathbf{k}\right)=\dfrac{\mathbf{C}\left(\mathbf{w}_1\mathbf{w}_2\dots\mathbf{w}_\mathbf{k}\right)}{\mathbf{N}}$
    - where $C(w_1w_2\dots w_k)$ is the number of times (count) sentence $w_1w_2\dots w_k$ appears in training data
    - N is number of sentences in training data
- Problem: language is infinite, many reasonable English sentences do not appear in training data
    - "A happy new mother first put on a purple polka-dot dress on her baby daughter, and then kissed her tiny left toe. "
    - Do not want any sentence to have probability 0

Markov Assumption

- Can we make some simplifying assumptions?
- Consider

  P(computer | instead of listening to this boring lecture, I would like to play on my)

- Probability that "computer" follows "Instead of listening to this boring lecture, I would like to play on my" is intuitively almost the same as probability that "computer" follows words "play on my"
- Probability of the next word depends most strongly on just a few previous words

Shannon Game (1951)

Context	0	1	2	3
Entropy (H)	4.76	4.03	3.21	3.1

"I am going to make a collect …"

- Predict next word/character given *n-1* previous words/characters
- Human subjects were shown a few characters of text and were asked to guess the next character
- As context increases, entropy decreases
    - the smaller the entropy => the larger probability of predicting next letter
- But only a few words is enough to make a good prediction on the next word, in most cases
- Evidence that we only need to look back at n-1 previous words

n-grams

- n-gram model: probability of a word depends only on the n-1 previous words (the history)

$$\mathbf{P}(w_k \mid \cancel{w_1 w_2 \ldots w}_{k-n} w_{k+1-n} \ldots w_{k-1}) \approx \mathbf{P}(w_k \mid w_{k+1-n} \ldots w_{k-1})$$

- This called **Markov Assumption**: only the closest n words are relevant
- Special cases:
  - Unigram (n=1): previous words do not matter
  - Bigram (n=2): only the previous one word matters
  - Trigram (n=3): only the previous two words matter

Example: Trigram Approximation (n = 3)

- Each word depends only on previous two words
  - three words total with the current one
  - tri means three
  - gram means writing

- **P**(the|… whole truth and nothing but) ≈
  P(the|nothing but)

- **P**(truth|… whole truth and nothing but the) ≈
  P(truth|but the)

Chain Rule

- First Decompose using the chain rule:

P(and nothing but the truth) =

P(and)

x **P**(nothing|and)

x **P**(but|and nothing)

x **P**(the|~~and~~ nothing but)

x **P**(truth|~~and nothing~~ but the)

- **P**(and nothing but the truth) ≈

  P(and)**P**(nothing|and) **P**(but|and nothing)
  P(the|nothing but) **P**(truth|but the)

How Compute Trigram Probabilities?

- $\mathbf{P}(w_3 \mid w_1 \, w_2) \approx ?$
  - these probabilities are usually called "parameters"

- First rewrite: $\mathbf{P}(w_3 \mid w_1 \, w_2) = \dfrac{\mathbf{P}(w_1 \, w_2 \, w_3)}{\mathbf{P}(w_1 \, w_2)}$

- Need to estimate $\mathbf{P}(w_1 \, w_2 \, w_3)$, $\mathbf{P}(w_1 \, w_2)$, $\mathbf{P}(w_1)$, etc.
  - will call these trigram, bigram, unigram, etc
- Get lots of real text, and approximate based on counts

$$\mathbf{P}(w_1 \, w_2 w_3) = \frac{\mathbf{C}(\mathbf{w_1} \, \mathbf{w_2} \, \mathbf{w_3})}{\text{number of trigrams in text}}$$

- where $\mathbf{C}(w_1 \, w_2 \, w_3)$ is the number of times we saw trigram $\mathbf{w_1} \, \mathbf{w_2}$ $\mathbf{w_3}$ in the training text

How Compute Trigram Probabilities?

- Suppose our text is

  "and nothing but the truth when nuts and nothing on the roof"

- 12 unigrams, 11 bigrams, 10 trigrams
- Estimate $\mathbf{P}(\text{but} \mid \text{and nothing}) = \dfrac{\mathbf{P}(\text{and nothing but})}{\mathbf{P}(\text{and nothing})}$

$$\mathbf{P}(\text{and nothing but}) = \frac{\mathbf{C}(\text{and nothing but})}{10} = \frac{1}{10}$$

$$\mathbf{P}(\text{and nothing}) = \frac{\mathbf{C}(\text{and nothing})}{11} = \frac{2}{11}$$

$$\mathbf{P}(\text{but} \mid \text{and nothing}) = \frac{1/10}{2/11} = \frac{11}{20}$$

How Compute Trigram Probabilities?

- In practice in a file with **N** words, we have
  - **N** unigrams
  - **N**-1 bigrams
  - **N**-2 trigrams, etc.
- **N** is so large, that dividing by **N**, or **N**-1, or **N**-2 makes no difference in practice

$$5/10{,}000{,}006 \;=_{\text{almost}}\; 5/10{,}000{,}005 =_{\text{almost}} 5/10{,}000{,}004$$

- Previous example becomes:

$$\mathbf{P}(\text{but} \mid \text{and nothing}) = \frac{1/12}{2/12} = \frac{1}{2}$$

How Compute Trigram Probabilities?

- Calculations simplify:

$$\mathbf{P}(w_3 \mid w_1\, w_2) = \frac{\mathbf{C}(w_1\, w_2\, w_3)/\mathbf{N}}{\mathbf{C}(w_1\, w_2)/\mathbf{N}} = \frac{\mathbf{C}(w_1\, w_2\, w_3)}{\mathbf{C}(w_1\, w_2)}$$

- Side note: this also avoids **P** > 1
- Consider training text again

“and nothing but the truth when nuts and nothing on the roof”

- If we used exact arithmetic, i.e. **N**-2, **N**-1

$$\mathbf{P}(\text{truth} \mid \text{but the}) = \frac{\mathbf{C}(\text{but the truth})\ /10}{\mathbf{C}(\text{but the})\ /11} = \frac{1/10}{1/11} = \frac{11}{10}$$

Computing Trigrams

- From now on

$$\mathbf{P}(w_3 \mid w_1 w_2) = \frac{\mathbf{C}(w_1 w_2 w_3)}{\mathbf{C}(w_1 w_2)}$$

$$\mathbf{P}(w_1 w_2 w_3) = \frac{\mathbf{C}(w_1 w_2 w_3)}{\mathbf{N}}$$

- where **N** is number of words in the training text

Trigrams, continued

- **P**(and nothing but the truth) ≈

  P(and)**P**(nothing|and) **P**(but|and nothing)

  P(the|nothing but) **P**(truth|but the)

$$= \frac{C(\text{and})}{N}\;\frac{C(\text{and nothing})}{C(\text{and})}\;\frac{C(\text{and nothing but})}{C(\text{and nothing})}\;\frac{C(\text{nothing but the})}{C(\text{nothing but})}\;\frac{C(\text{but the truth})}{C(\text{but the})}$$

- where **N** is the number of words in our training text

Text Generation with n-grams

- Trained on 40 million words from WSJ (wall street journal)
- Generate next word according to the n-gram model
- 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 planner 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 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.*

Example with Sentence Start/End

- Training text:

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

- Bigram model: $\mathbf{P}(w_i | w_{i-1}) = \dfrac{\mathbf{C}(w_{i-1}w_i)}{\mathbf{C}(w_{i-1})}$

- Some bigram probabilities:

P(I|<s>) = 2/3 = 0.67	**P**(Sam|<s>) = 1/3 = 0.33
P(</s>|Sam) = 1/2 =0.5	**P**(Sam|am) = 1/2 = 0.5
P(am|I) = 2/3 = 0.67	**P**(do|I) = 1/3 = 0.33

Raw Bigram Counts

- Can construct **V**-by-**V** matrix of probabilities/frequencies
- **V** = size of the vocabulary we are modeling
- Used 922 sentences

2nd word

1st word

	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

- Normalize by unigrams to get conditional P(second|first) :

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

P(want|I)

P(<s> I want chinese food </s>) = P(I|<s>)
× P(want|I)
× P(chinese|want)
× P(food|chinese)
× P(</s>|food)
= .000031

Practical Issue

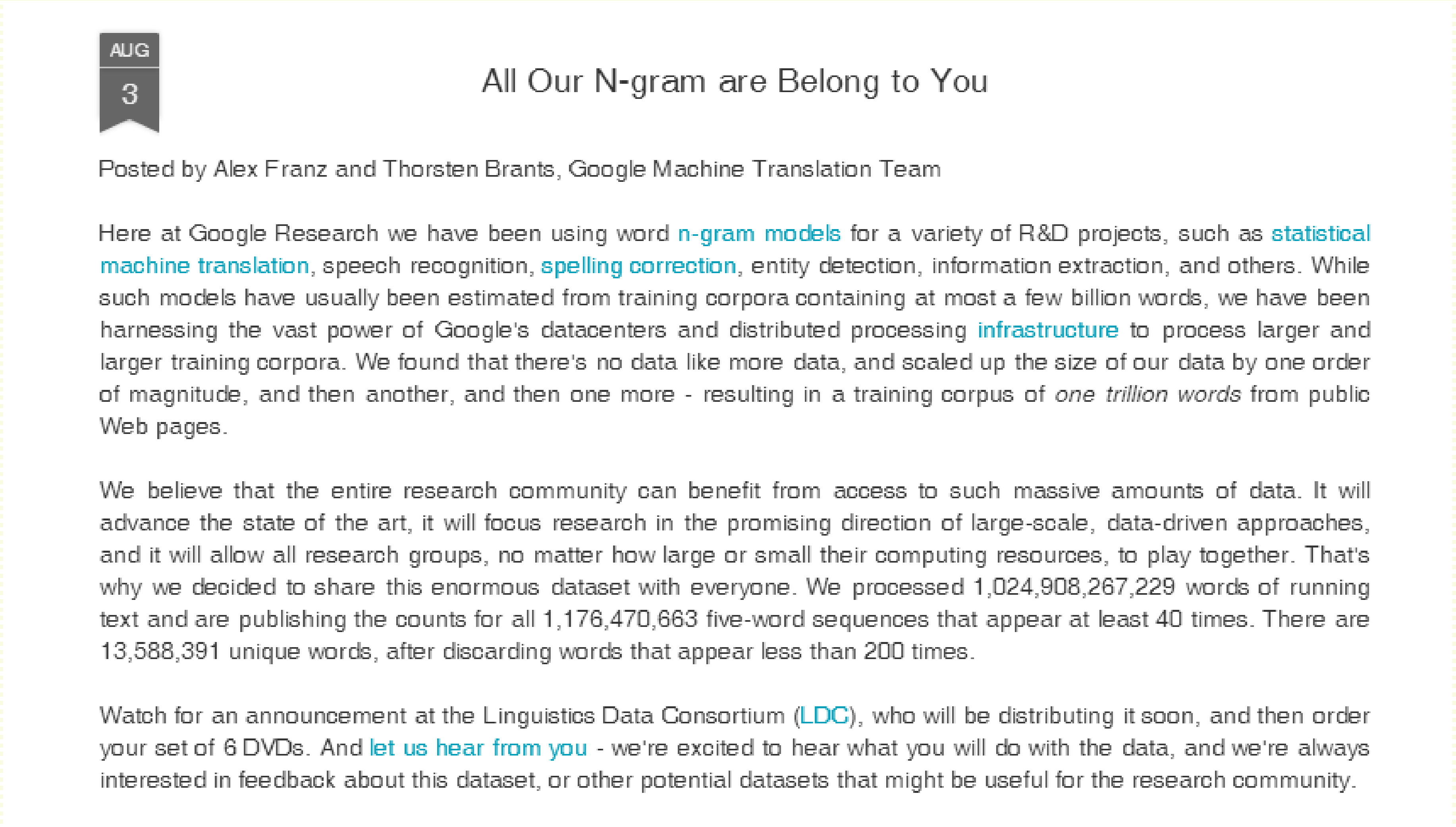
- We do everything in log space
  - to avoid underflow
  - also adding is faster than multiplying
  - instead of **P**(a)×**P**(b)×**P**(c) compute log[**P**(a)]+ log[**P**(a)] +log[**P**(a)]
- Example, instead of:

P(<s> I want chinese food </s>) =**P**(I|<s>) × **P**(want|I) × **P**(chinese|want)
× **P**(food|chinese) × **P**(</s>|food)
= .000031

- we compute:

log[**P**(<s> I want chinese food </s>)] = log[**P**(I|<s>)] + log[**P**(want|I)] +
log[**P**(chinese|want)] + **P**(food|chinese)
+ log[**P**(</s>|food)] = -4.501

Google N-Gram Release, August 2006



AUG
3

All Our N-gram are Belong to You

Posted by Alex Franz and Thorsten Brants, Google Machine Translation Team

Here at Google Research we have been using word n-gram models for a variety of R&D projects, such as statistical machine translation, speech recognition, spelling correction, entity detection, information extraction, and others. While such models have usually been estimated from training corpora containing at most a few billion words, we have been harnessing the vast power of Google's datacenters and distributed processing infrastructure to process larger and larger training corpora. We found that there's no data like more data, and scaled up the size of our data by one order of magnitude, and then another, and then one more - resulting in a training corpus of *one trillion words* from public Web pages.

We believe that the entire research community can benefit from access to such massive amounts of data. It will advance the state of the art, it will focus research in the promising direction of large-scale, data-driven approaches, and it will allow all research groups, no matter how large or small their computing resources, to play together. That's why we decided to share this enormous dataset with everyone. We processed 1,024,908,267,229 words of running text and are publishing the counts for all 1,176,470,663 five-word sequences that appear at least 40 times. There are 13,588,391 unique words, after discarding words that appear less than 200 times.

Watch for an announcement at the Linguistics Data Consortium (LDC), who will be distributing it soon, and then order your set of 6 DVDs. And let us hear from you - we're excited to hear what you will do with the data, and we're always interested in feedback about this dataset, or other potential datasets that might be useful for the research community.

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

Google Book N-grams

- http://ngrams.googlelabs.com/

Which n-gram to Use?

- *“the large green _______ .”*
  - *“mountain”? “tree”? “pill”? “broccoli”?* …
- *“Sue swallowed the large green _______ .”*
  - *“pill”? “broccoli”?*
- Knowing that Sue “*swallowed*” helps narrow down possibilities
- Larger ***n*** corresponds to more context, i.e. looking back further
- But we need more data to estimate larger **n-**grams reliably

Which n-gram to use?

- example: for a vocabulary of 20,000 words
  - number of bigrams = 400 million ($20\,000^2$)
  - number of trigrams = 8 trillion ($20\,000^3$)
  - number of four-grams = 1.6×10^{17} ($20\,000^4$)
- number of **n**-grams is exactly the number of parameters to learn
- Going from **n**-gram to (**n**+1)-gram, number of parameters to learn grows by a factor of **n**
- For reliable estimates, the more parameters we need to learn, the more training samples we need
- But usually training data has fixed size of **N** words, it does not change when we go from **n**-gram to (**n**+1)-gram

Unigram vs. Bigram Illustration

- For reliable estimates, the more parameters we need to learn, the more training samples we need
- Suppose Vocabulary size is 10,000=10^4 and we have a text with 100,000 = 10^5 words, i.e. 10^5 training samples
- For Unigrams
  - Need to estimate unigram counts for each vocabulary word, i.e. C('place'), C('apple'), etc.
  - Number of parameters (unigram counts) to estimate is 10^4
  - On average, $10^5/ 10^4$ = 10 training samples per parameter, reasonable
- For Bigrams
  - Need to estimate bigram counts for $10^4 * 10^4 = 10^8$ possible bigrams
  - Number of parameters (bigram counts) to estimate is 10^8
  - Number of training samples is still 10^5 (or $10^5 - 1$ to be exact)
  - On average, have $10^5/10^8 = 10^{-3}$ training samples to fit per parameter
  - Highly insufficient, need much more data

Reliability vs. Discrimination

- larger n:
  - **greater discrimination**: more information about the context of the specific instance
  - but **less reliability**:
    - model is too complex, that is has too many parameters
    - cannot estimate parameters reliably from limited data (data sparseness)
      - too many chances that the history has never been seen before
      - parameter estimates are not reliable because we have not seen enough examples
- smaller n:
  - **less discrimination**, not enough history to predict next word very well, our model is not so good
  - but **more reliability**:
    - more instances in training data, better statistical estimates of parameters
- Bigrams or trigrams are most often used in practice
  - works well, although there are longer-range dependencies not captured

Reducing number of Parameters

- with a 20 000 word vocabulary:
  - bigram needs to store 400 million parameters
  - trigram needs to store 8 trillion parameters
  - using a language model > trigram is impractical
- to reduce the number of parameters, we can:
  - do stemming (use stems instead of word types)
    - help = helps = helped
  - group words into semantic classes
    - {Monday,Tuesday,Wednesday,Thursday,Friday} = one word
  - seen once --> same as unseen

Statistical Estimators

- How do we estimate parameters (probabilities of unigrams, bigrams, trigrams)?
- Maximum Likelihood Estimation (MLE)
  - already saw this, major problems due to data sparseness
- Smoothing
  - Add-one -- Laplace
  - Add-delta -- Lidstone's & Jeffreys-Perks' Laws (ELE)
  - Good-Turing
- Combining Estimators
  - Simple Linear Interpolation

Maximum Likelihood Estimation

- Already saw this
- Let $C(w_1 \ldots w_n)$ be the frequency of n-gram $w_1 \ldots w_n$

$$P_{MLE}(w_n \mid w_1 \ldots w_{n-1}) = \frac{C(w_1 \ldots w_n)}{C(w_1 \ldots w_{n-1})}$$

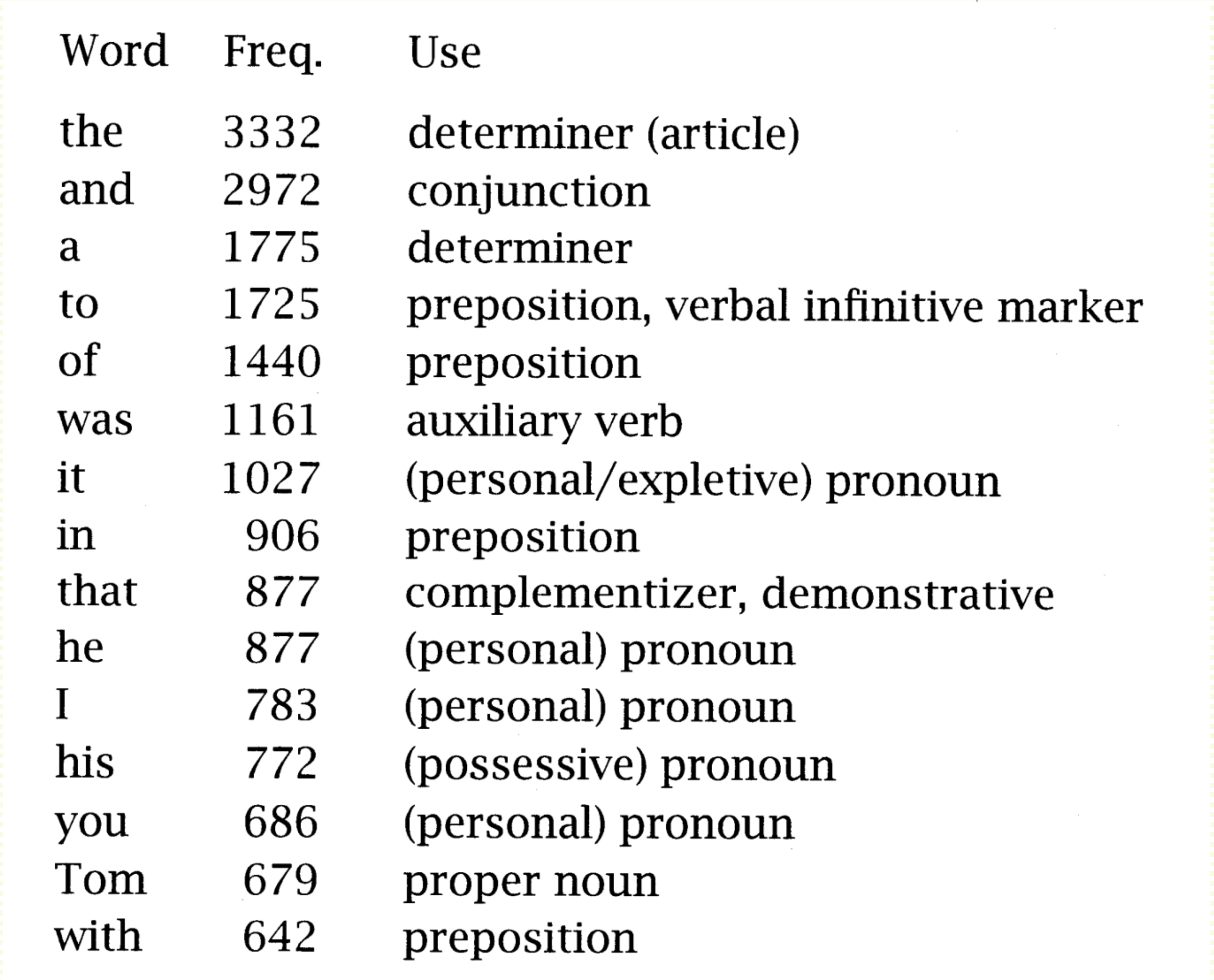
- “Maximum Likelihood” because the parameter values it gives lead to highest probability of the **training** corpus
- However, interested in good performance on **test** data

Data Sparseness Example

- in a training corpus, we have 10 instances of "*come across*"
  - 8 times, followed by "*as*"
  - 1 time, followed by "*more*"
  - 1 time, followed by "*a*"

- so we have:
  - $P_{MLE}(\text{as} \mid \text{come across}) = \dfrac{C(\text{come across as})}{C(\text{come across})} = \dfrac{8}{10}$
  - $P_{MLE}(\text{more} \mid \text{come across}) = 0.1$
  - $P_{MLE}(\text{a} \mid \text{come across}) = 0.1$
  - $P_{MLE}(X \mid \text{come across}) = 0$ where $X \neq$ "*as*", "*more*", "*a*"

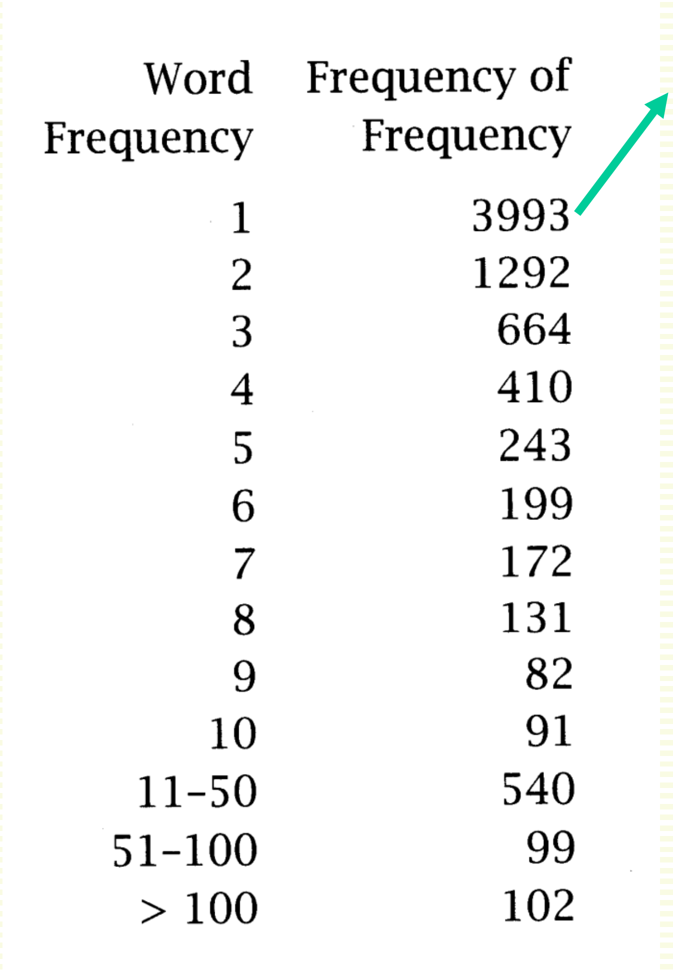
Common words in Tom Sawyer



Word	Freq.	Use
the	3332	determiner (article)
and	2972	conjunction
a	1775	determiner
to	1725	preposition, verbal infinitive marker
of	1440	preposition
was	1161	auxiliary verb
it	1027	(personal/expletive) pronoun
in	906	preposition
that	877	complementizer, demonstrative
he	877	(personal) pronoun
I	783	(personal) pronoun
his	772	(possessive) pronoun
you	686	(personal) pronoun
Tom	679	proper noun
with	642	preposition

but words in NL have an uneven distribution…

Most Words are Rare



Word Frequency	Frequency of Frequency
1	3993
2	1292
3	664
4	410
5	243
6	199
7	172
8	131
9	82
10	91
11-50	540
51-100	99
> 100	102

- most words are rare
  - 3993 (50%) word types appear only once
  - they are called **happax legomena** (*read only once*)
- common words are **very** common
  - 100 words account for 51% of all tokens (of all text)

Problem with MLE: Data Sparseness

- Got trigram "nothing but the" in training corpus, but not trigram "and nuts sing"
- Therefore estimate P(and nuts sing) = 0
- Any sentence which has "and nuts sing" has probability 0
  - We want P("and nuts sing") to be small, but not 0!
- If a trigram never appears in training corpus, probability of sentence containing this trigram is 0
- MLE assigns a probability of zero to unseen events …
- Probability of an n-gram involving unseen words will be zero
- but … most words are rare
- n-grams involving rare words are even more rare… data sparseness

Problem with MLE: Data Sparseness

- From [Balh et al 83]
  - training with 1.5 million words
  - 23% of the trigrams from another part of the same corpus were previously unseen
- in Shakespeare's work
  - out of all possible bigrams, 99.96% were never used
- So MLE alone is not good enough estimator

Discounting or Smoothing

- MLE alone is unsuitable for NLP because of the sparseness of the data
- We need to allow for possibility of seeing events not seen in training
- Must use a **Discounting** or **Smoothing** technique
- Decrease the probability of previously seen events to give a little bit of probability for previously unseen events

Smoothing

- Increase **P**(unseen event) → decrease **P**(seen event)
- **P**(w|denied the)

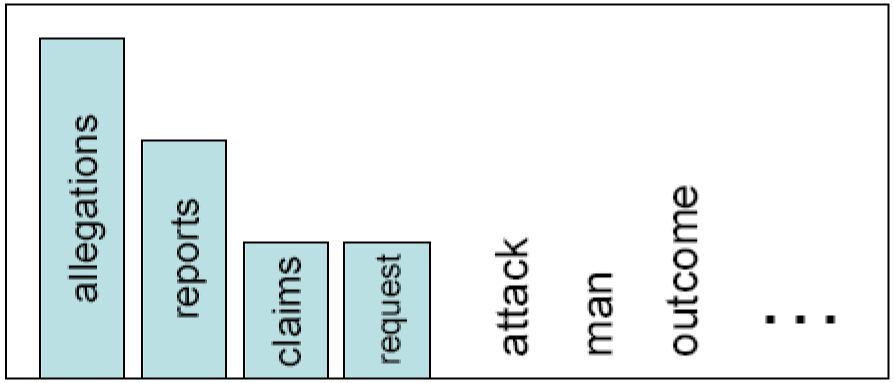
  3 allegations

  2 reports

  1 claims

  1 request

  7 total



- Smoothing flattens spiky distributions so they generalize better

P(w|denied the)

  2.5 allegations

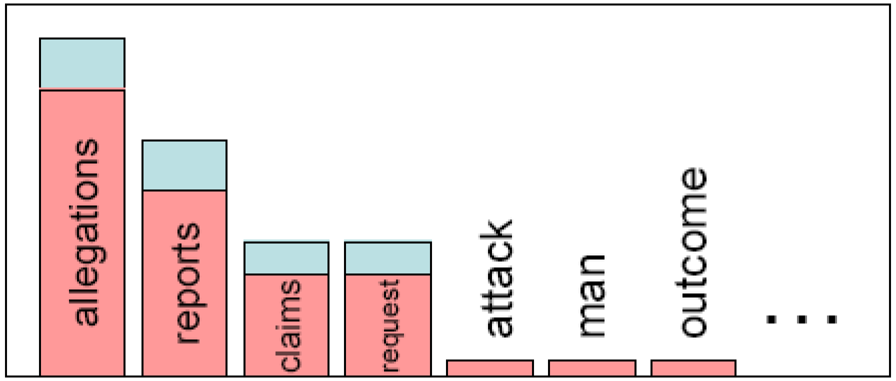
  1.5 reports

  0.5 claims

  0.5 request

  2 other

  7 total

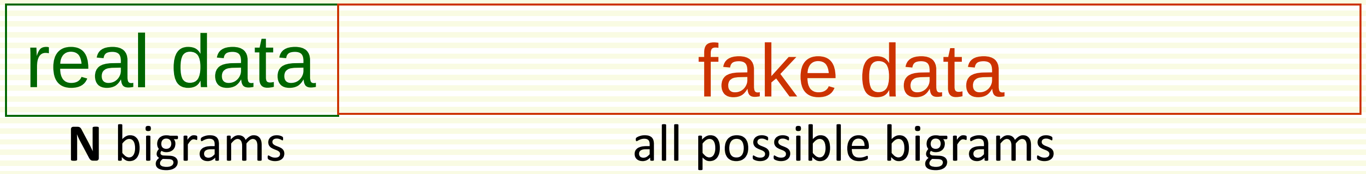


Many smoothing techniques

- Add-one
- Add-delta
- Good-Turing smoothing
- Many other methods we will not study...

Add-one Smoothing (Laplace's Law 1814)

- Give a little bit of probability space to unseen events
- Pretend we have seen every n-gram at least once
- Intuitively appended all possible n-grams to training data



- Training data has **N** n-grams
- The "new" size is **N+B**,where **B** is # of all possible n-grams
- If **V** words in vocabulary, then:
  - **B= V*V** for bigrams
  - **B=V*V*V** for trigrams
  - etc.
- We get: $P_{Add1}(w_1\, w_2 \ldots w_n) = \dfrac{C(w_1\, w_2 \ldots w_n) + 1}{N + B}$

Add-One Example

$$P_{Add1}(w_1 w_2 \ldots w_n) = \frac{C(w_1 w_2 \ldots w_n) + 1}{N + B}$$

- Let us use character model
- Training data = “abraabr”
  - **N** = 7
  - bigrams = ab,br,ra,aa,ab,br
- Let **V** = 256
- With bigram approximation (**n** = 2), B = 256^2

$$P_{Add1}(ab) = \frac{C(ab) + 1}{7 + 256^2} = \frac{3}{7 + 256^2} \approx 4.6 \times 10^{-5}$$

Add-One Example

- How well does it work in practice?
- Works ok if sparsity problem is mild
    - not a lot of missing nGrams

Add-One: Example

Unsmoothed bigram counts

	I	want	to	eat	Chinese	food	lunch	…	Total
I	8	1087	0	13	0	0	0		N(I)=3437
want	3	0	786	0	6	8	6		N(want)=1215
to	3	0	10	860	3	0	12		N(to)=3256
eat	0	0	2	0	19	2	52		N(eat)=938
Chinese	2	0	0	0	0	120	1		N(Chinese)=213
food	19	0	17	0	0	0	0		N(food)=1506
lunch	4	0	0	0	0	1	0		N(lunch)=459
…									N=10,000

Unsmoothed bigram probabilities

	I	want	to	eat	Chinese	food	lunch	…	Total
I	.0008	.1087	0	.0013	0	0	0		
want	.0003	0	.0786	0	.0006	.0008	.0006		
to	.0003	0	.001	.086	.0003	0	.0012		
eat	0	0	.0002	0	.0019	.0002	.0052		
Chinese	.0002	0	0	0	0	.012	.0001		
food	.0019	0	.0017	0	0	0	0		
lunch	.0004	0	0	0	0	.0001	0		
…									N=10,000

Add-one: Example

add-one smoothed bigram counts

	I	*want*	*to*	*eat*	*Chinese*	*food*	…	*Total*
I	~~8~~ 9	~~1087~~ 1088	1	14	1	1		~~3437~~ N(I) + V = 5053
want	~~3~~ 4	1	787	1	7	9		N(want) + V = 2831
to	4	1	11	861	4	1		N(to) + V = 4872
eat	1	1	23	1	20	3		N(eat) + V = 2554
Chinese	3	1	1	1	1	121		N(Chinese) + V =1829
food	20	1	18	1	1	1		N(food) + V = 3122
…								~~N= 10,000~~ $N+V^2$ = 10,000 +1616^2 = 2,621,456

add-one bigram probabilities

	I	*want*	*to*	*eat*	*Chinese*	…
I	.0000034 (9/2621456)	.00041	.00000038	.0000053	.00000038	
want	.0000015	.00000038	.0003	.00000038	.0000027	
to	.0000015	.00000038	.000004	.0046	.0000015	
eat	.00000038	.00000038	.0000088	.00000038	.0000076	
…						

Example Allocation to Unseen Bigrams

- Data from the AP from (Church and Gale, 1991)
- **N** = 22,000,000
- **V** = 273,266
- **B** = **V**^{2} = 74,674,306,756
- 74,671,100,000 unseen bigrams
- Add One probability of unseen bigram:

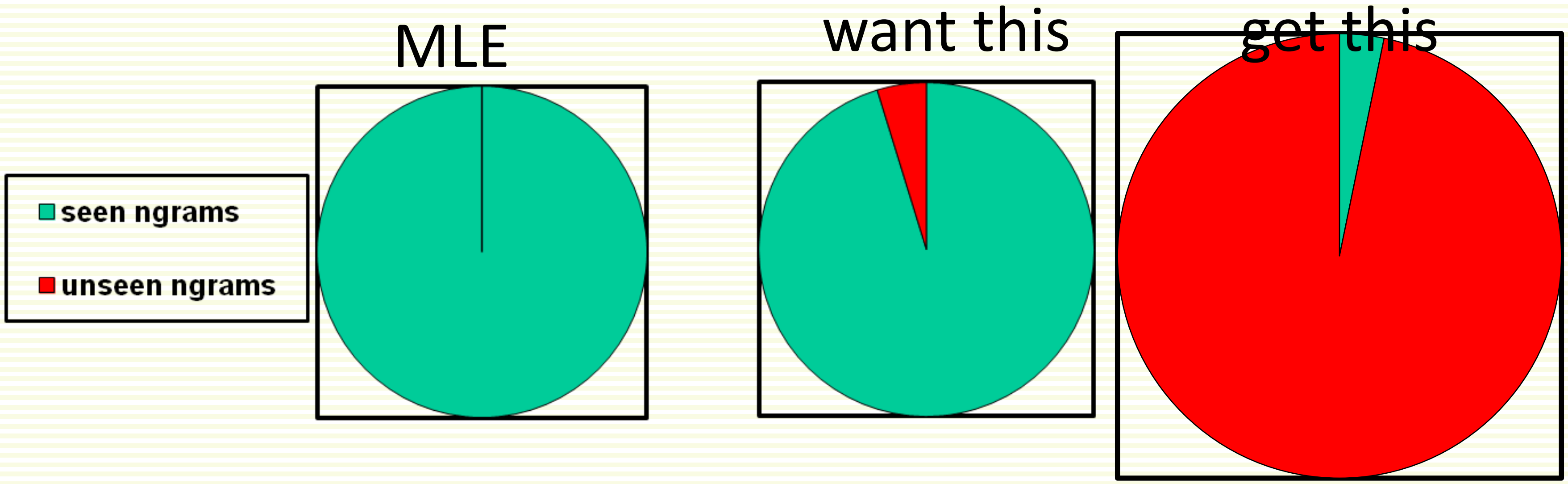
$$\frac{1}{N+B} = \left(\frac{1}{22{,}000{,}000 + 74{,}674{,}306{,}756} \right) = 1.33875\text{x}10^{-11}$$

- Portion of probability mass given to unseen bigrams:

number of unseen bigrams x **P**(unseen bigram) =

$$74{,}671{,}100{,}000 \times \left(1.33875 \times 10^{-11}\right) \approx 99.96$$

Problem with add-one smoothing



- each individual unseen n-gram is given a low probability
- but there is a huge number of unseen n-grams
- Instead of giving small portion of probability to unseen events, most of the probability space is given to unseen events
- But how do we know we gave too much space to unseen bigrams? Maybe they should have 99% of all probability space?

Evaluation: How good is our model?

- Train parameters of our model on a **training set**
- Test model performance on data we haven't seen
  - A **test set** is an unseen dataset that is different from our training set, totally unused
  - An **evaluation metric** tells us how well our model does on the test set
    - compare estimated counts (probability) with actual counts (empirical counts) on test data
    - recall that count/**N** is the probability
    - it's easy to switch between the two
    - but count gives an easier number to look at, probability is usually tiny

Evaluation: How good is our model?

- Compare the actual counts on the test set (empirical counts) to counts predicted by the model
- $\mathbf{C}_{train}(w_1 \ldots w_n)$ = count of $w_1 \ldots w_n$ in the training data
- $\mathbf{C}_{test}(w_1 \ldots w_n)$ = count of $w_1 \ldots w_n$ in the test data
- $\mathbf{N}_r$ = number of n-grams with count **r** in training data
- Let $\mathbf{T}_r$ be total number of times all n-grams that appeared r-times in training data appear in test data

$$\mathbf{T_r} = \sum_{\mathbf{w_1 \ldots w_n : C_{train}(w_1 \ldots w_n) = r}} \mathbf{C_{test}}(\mathbf{w_1 \ldots w_n})$$

- Empirical count (averaged) of these n-grams is $\mathbf{T}_r/\mathbf{N}_r$
- Want predicted count close to empirical count $\mathbf{T}_r/\mathbf{N}_r$

Evaluation: Bigrams Example

- V = {a,b,r}
- Training data = “abraabr”
  - Training bigrams = ab, br, ra, aa, ab, br
  - $\mathbf{C}_{train}$(ab) = 2, $\mathbf{C}_{train}$(br)=2, $\mathbf{C}_{train}$(ra)=1, $\mathbf{C}_{train}$(aa)=1
  - $\mathbf{N}_0$ = 5 (ar, ba, bb, rb, rr), $\mathbf{N}_1$ = 2, $\mathbf{N}_2$ = 2
- Test data = “raraabr”
  - Test bigrams = ra, ar, ra, aa, ab, br
  - $\mathbf{C}_{test}$(ra) = 2, $\mathbf{C}_{test}$(ar) = 1, $\mathbf{C}_{test}$(aa) = 1, $\mathbf{C}_{test}$(ab) = 1,$\mathbf{C}_{test}$(br) = 1
- $\mathbf{T}_0$ = $\mathbf{C}_{test}$(ar)= 1, $\mathbf{T}_1$ = $\mathbf{C}_{test}$(aa)+$\mathbf{C}_{test}$(ra)= 3, $\mathbf{T}_2$ = $\mathbf{C}_{test}$(ab) +$\mathbf{C}_{test}$(br)= 2
- Empirical counts: $\mathbf{T}_0/\mathbf{N}_0$ =1/5, $\mathbf{T}_1/\mathbf{N}_1$ =3/2 and $\mathbf{T}_2/\mathbf{N}_2$ =2/2
  - bigrams that never occur in training, occur, on average, 1/5 times in test
  - bigrams that occurred once in training, occur, on average, 1.5 times in test
  - bigrams that occurred twice in training , occur, on average, once in test
- Predicted counts are good if close to the empirical counts

Counts on Test Data

- Corpus of 44,000,000 bigram tokens, 22,000,000 for training, 22,000,000 for testing
  - Data from the AP from (Church and Gale, 1991)
- To get probability, divide count by 22,000,000
- Each unseen bigram was given a count of 0.000295

num. of times appeared in training corpus

observed count in testing corpus

Add-one count on testing corpus

C_{MLE}	$C_{empirical}$	$C_{add\text{-}one}$
0	**0.000027**	**0.000295**
1	**0.448**	**0.000589**
2	**1.25**	**0.000884**
3	**2.24**	**0.001180**
4	**3.23**	**0.001470**
5	**4.21**	**0.001770**

too high

too low

Add-delta smoothing (Lidstone's law)

- instead of adding 1, add some smaller positive value δ

$$P_{AddD}(w_1 w_2 \ldots w_n) = \frac{C(w_1 w_2 \ldots w_n) + \delta}{N + \delta B}$$

- This is called Lidstone's law
- most widely used value for δ = 0.5, in this case it's called
  - the **Expected Likelihood Estimation** (ELE)
  - or the **Jeffreys-Perks** Law

$$P_{ELE}(w_1 w_2 \ldots w_n) = \frac{C(w_1 w_2 \ldots w_n) + 0.5}{N + 0.5B}$$

- better than add-one, but still not very good

Add-Delta Example

$$P_{AddD}(w_1 w_2 \ldots w_n) = \frac{C(w_1 w_2 \ldots w_n) + \delta}{N + \delta B}$$

- Let us use character model
- Training data = "abraabr"
  - **N** = 7
  - trigrams = abr, bra, raa, aab, abr
- Let **V** = 256
- With trigram approximation (**n** = 3), B = 256^3

$$P_{AddD}(aba) = \frac{C(aba) + 0.1}{7 + 0.1 \cdot 256^3} = \frac{0.1}{7 + 0.1 \cdot 256^3} \approx 5.9 \times 10^{-8}$$

Smoothing: Good Turing (1953)

- Suppose we are fishing, fish species in the sea are:
  - carp, cod, tuna, trout, salmon, eel, shark, tilapia, etc …
- We caught 10 carp, 3 cod, 2 tuna, 1 trout, 1 salmon, 1 eel
- How likely is it that the next species is new?
  - roughly 3/18, since 18 fish total, 3 unique species (trout, salmon, eel)
- Say that there are 20 species of fish that *we have not seen* yet (bass, shark, tilapia,….)
- Probability of any individual *unseen* species is

$$\frac{3}{18 \cdot 20}$$

- **P**(bass) = **P**(shark)= **P**(tilapia) = $\frac{3}{18 \cdot 20}$

Smoothing: Good Turing

- Let $\mathbf{N_1}$ be the number species (n-grams) seen once
- Use it to estimate for probability of unseen species
  - probability of new species (unseen n-gram) is $\mathbf{N_1/N}$
- Let $\mathbf{N_0}$ be the number of unseen species (unseen n-grams)
- Spreading around the mass equally for unseen n-grams, the probability of seeing any individual unseen species (unseen n-gram) is

$$\frac{\mathbf{N_1}}{\mathbf{N \cdot N_0}}$$

Smoothing: Good Turing

- We caught 10 Carp, 3 Cod, 2 tuna, 1 trout, 1 salmon, 1 eel
- 20 species unseen so far
- How likely is it that next species is new? 3/18
  - The probability of any individual unseen fish is $\frac{3}{18 \cdot 20}$
- What is the new probability of catching a trout?
  - should be smaller than 1/18 to make room for unseen fish
  - continue the in the same direction as with unseen species
  - if we catch another trout, trout will occur with the rate of 2
  - according to our data, what is the probability of fish with rate = count = 2?
  - tuna occurs 2 times, so probability is 2/18
  - now spread the probability of 2/18 over all species seen once
    - 3 species (trout, salmon, eel)
  - probability of catching a fish which occurred 1 time: $\frac{2}{18 \cdot 3}$

Smoothing: Good Turing: General Case

- Let **r** be rate (count) with which n-gram occurs in training data
- If an n-gram occurs with rate **r**, computed its probability as
  - **r/N**, where **N** is the size of the training data
  - need to lower all these rates to make room for unseen n-grams
- The number of n-grams which occur with rate **r**+1 is smaller than the number of grams which occur with rate **r**
- Good-Turing Idea: take the portion of probability space occupied by n-grams which occur with rate **r**+1 and divide it among the n-grams which occur with rate **r**

Good Turing Formula

- **N_r** number of different n-grams in training occuring exactly **r** times
  - training data ="catch a cow, make a cow sing"
  - bigrams = "catch a", "a cow", "cow make", "make a", "cow sing"
  - $N_1 = 4$ and $N_2 = 1$
- Probability for any n-gram with rate **r** is estimated from the space occupied by n-grams with rate **r**+1
- **N** is the size of the training data. Space occupied by n-grams with rate **r**+1 is:

$$\frac{(\mathbf{r}+1)\mathbf{N}_{\mathbf{r}+1}}{\mathbf{N}}$$

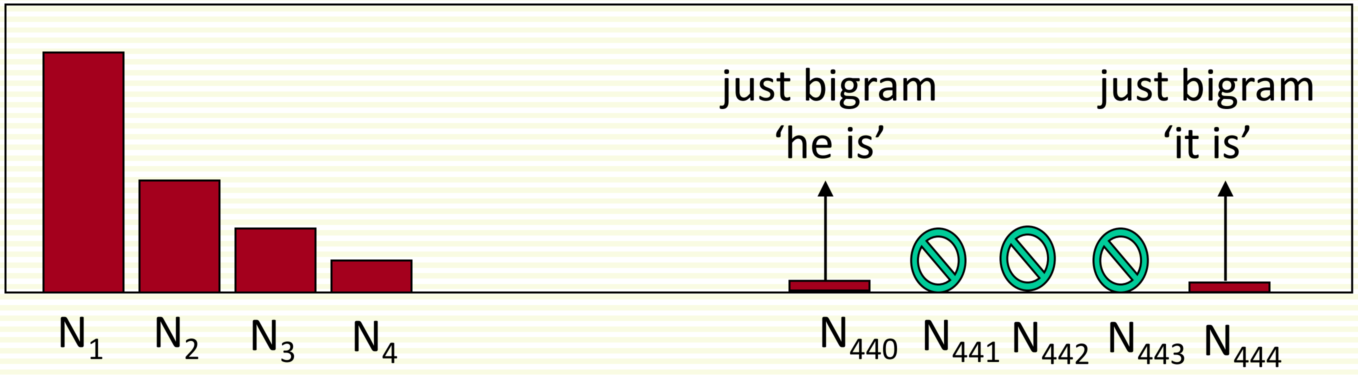
- Spread it evenly among n-grams with rate **r**, there are **N_r** of them:

$$\frac{(\mathbf{r}+1)\mathbf{N}_{\mathbf{r}+1}}{\mathbf{N}\cdot\mathbf{N}_\mathbf{r}}$$

- If n-gram x has rate **r**, Good Turing estimate is: $\mathbf{P}_{\mathbf{GT}}(\mathbf{x}) = (\mathbf{r}+1)\frac{\mathbf{N}_{\mathbf{r}+1}}{\mathbf{N}\cdot\mathbf{N}_\mathbf{r}}$

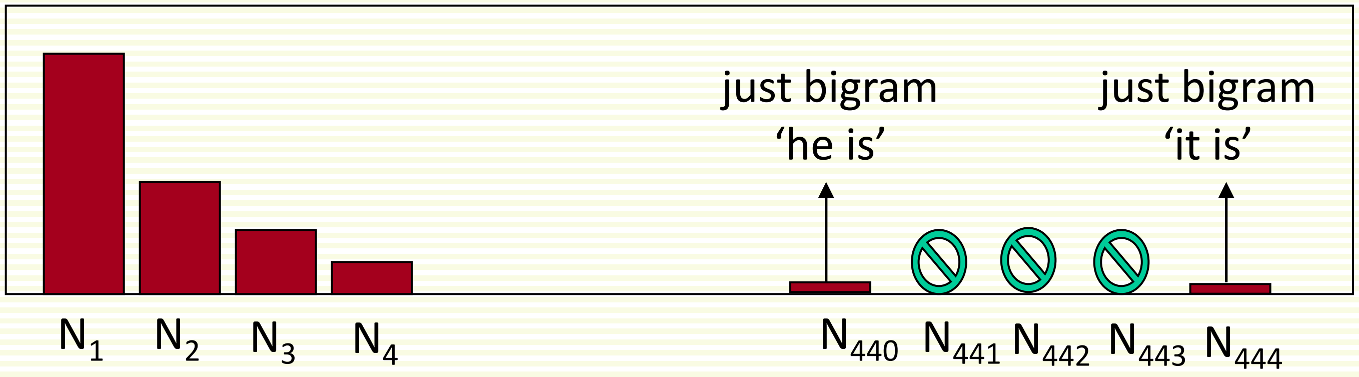
Fixing Good Turing

- If n-gram x that occurs **r** times: $\mathbf{P_{GT}}(\mathbf{x}) = (\mathbf{r}+1)\dfrac{\mathbf{N_{r+1}}}{\mathbf{N}\cdot\mathbf{N_r}}$
- Does not work well for high values of **r**
- $\mathbf{N_r}$ is not reliable estimate of the number of n-grams that occur with rate **r**
- In particular, fails for the most frequent **r** since $\mathbf{N_{r+1}}$=0



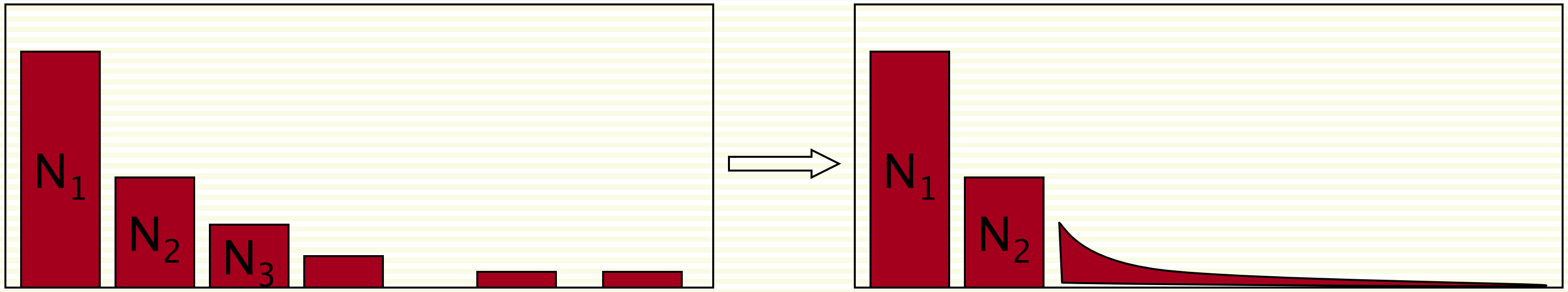
Fixing Good Turing: Solution 1

- MLE is reliable for higher values of **r**
- choose threshold **t**, say **t** = 6
  - best threshold depends on data
- for **r** $\geq$ t, use $\mathbf{P}_{MLE}(w_1 \ldots w_n) = \mathbf{C}(w_1 \ldots w_n)/\mathbf{N}$
- for **r** < t, use P_{GT}
  - for this range of **r**, make sure $\mathbf{N}_{\mathbf{r+1}} > 0$
  - also make sure $\frac{\mathbf{N}_{r+1}}{\mathbf{N}_r} < \frac{\mathbf{r}}{(\mathbf{r}+1)}$
  - otherwise the new rate $r^* = \frac{(r+1)N_{r+1}}{N_r}$ is larger than old rate **r**



Fixing Good Turing: Solution 2

- Smooth out $\mathbf{N_r}$ by fitting a power law function $\mathbf{F(r) = ar^b}$
  - with $b < -1$
- Search for the best **a** and **b** < -1 to fit observed $\mathbf{N_r}$
  - one line in Matlab
- Used smoothed $\mathbf{N_r}$ instead of actual $\mathbf{N_r}$



Smoothing: Fixing Good Turing

- Probabilities will not add up to 1, whether using Solution 1 or Solution 2 from the previous slide
- Have to renormalize all probabilities so that they add up to 1
  - could renormalize all n-grams
  - or renormalize only the n-grams with observed rates higher than 0
    - suppose the total space for unseen n-grams is 1/20
    - renormalize the weight of the seen n-grams so that the total is 19/20

Good Turing vs. Add-One

c_{MLE}	$c_{empirical}$	$c_{add-one}$	c_{GT}
0	0.000027	0.000295	0.000027
1	0.448	0.000589	0.446
2	1.25	0.000884	1.26
3	2.24	0.001180	2.24
4	3.23	0.001470	3.24
5	4.21	0.001770	4.22

Good-Turing (GT) Example

- $\mathbf{P}_{GT}$(n-gram occuring r times) = $\mathbf{(r+1)}\frac{\mathbf{N}_{r+1}}{\mathbf{N}\mathbf{N}_r}$
- Vocabulary is {a,b,c}
- Possible bigrams: {aa,ab,ba,bb,ac,bc,ca,cb,cc}
- Corpus: babaacbcacac
  - observed bigrams are {ba, ab, ba, aa, ac, cb, bc, ca, ac, ca, ac}
  - unobserved bigrams: bb,cc
- Observed bigram counts
  - ab: 1, aa: 1,cb: 1, bc: 1, ba: 2, ca: 2, ac: 3
- $\mathbf{N}_0=2$, $\mathbf{N}_1=4$, $\mathbf{N}_2=2$, $\mathbf{N}_3=1$, $\mathbf{N} = 12$
- Use Good Turing (GT) probabilities up to and including **r** = 2
- GT: **P**(bb) = **P**(cc)= $(0+1)\times(\mathbf{N}_1/(\mathbf{N}\times\mathbf{N}_0))=4/(12\times2) = 1/6$
- GT: **P**(ab) = **P**(aa)=**P**(cb)=**P**(bc)= $(1+1)\times(\mathbf{N}_2/(\mathbf{N}\times\mathbf{N}_1)) = 1/12$
- GT: **P**(ba) = **P**(ca)= $(2+1)\times(\mathbf{N}_3/(\mathbf{N}\times\mathbf{N}_2)) = 1/8$
- MLE: **P**(ac) = 3/12 = 1/4

Good-Turing (GT) Example

- Now renormalize. Before renormalization:
  - **P**’(bb) = **P**’(cc) = 1/6
  - **P**’(ab) = **P**’(aa) = **P**’(cb) = **P**’(bc) = 1/12
  - **P**’(ba) = **P**’(ca) = 1/8
  - **P**’(ac) = 1/4
  - **P**’(·) to indicate that the above are not true probabilities, they don’t add up to 1
- Renormalization 1
  - unseen bigrams should occupy **P**’(bb) + **P**’(cc) = 1/3 of space after normalization
- Weight of seen bigrams ab,aa,cb,bc,ba,ca,ac should be 1 – 1/3 = 2/3
  - **P**’(ab) + **P**’(aa) + **P**’(cb) + **P**’(bc) + **P**’(ba) + **P**’(ca) + **P**’(ac) = 10/12 = 5/6
  - Solve for **y** equation:
    - (5/6) × **y** = 2/3
    - **y** = 4/5
  - Multiply the above **P**’(·) by 4/5, except for the unseen bigrams:
    - **P**(bb) = **P**(cc)= 1/6, did not want to change these
    - **P**(ab) = **P**(aa) = **P**(cb) = **P**(bc)= (1/12)×(4/5) = 1/15
    - **P**(ba) = **P**(ca) = (1/8)×(4/5) = 1/10
    - **P**(ac) = (1/4)×(4/5) =1/5

Good-Turing (GT) Example

- Renormalization 2:
- Before renormalization:
  - **P**'(bb) = **P**'(cc)= 1/6 = $\mathbf{P}'_0$
  - **P**'(ab) = **P**'(aa) = **P**'(cb) = **P**'(bc) = 1/12 = $\mathbf{P}'_1$
  - **P**'(ba) = **P**'(ca) = 1/8 = $\mathbf{P}'_2$
  - **P**'(ac) = 1/4 = $\mathbf{P}'_3$
- Simply renormalize all **P**' to add to 1
  - (1) find their sum; (2) Divide each by the sum
- Add up based on rates, since ngrams with the same rate have equal probability
  - Let $\mathbf{S}_r$ contain all nGrams that were observed **r** times, $\mathbf{N}_r$ is the number of items in $\mathbf{S}_r$
  - $\mathbf{S}_0$ = {bb,cc}, $\mathbf{S}_1$ = {ab,aa,cb,bc}, $\mathbf{S}_2$ = {ba,ca}, $\mathbf{S}_3$ = {ac}
  - sum = $\mathbf{P}'_0\mathbf{N}_0+\mathbf{P}'_1\mathbf{N}_1+\mathbf{P}'_2\mathbf{N}_2+\mathbf{P}'_3\mathbf{N}_3 = (1/6)\times 2 + (1/12)\times 4 + (1/8)\times 2 + (1/4) = 7/6$
- New probabilities are:
  - **P**(bb) = **P**(cc) = (1/6)/(7/6) = 1/7 = $\mathbf{P}_0$
  - **P**(ab) = **P**(aa) = **P**(cb) = **P**(bc) = (1/12)/(7/6) = 1/14 = $\mathbf{P}_1$
  - **P**(ba) = **P**(ca) = (1/8)/(7/6) = 3/28 = $\mathbf{P}_2$
  - **P**(ac) = (1/4)/(7/6) = 3/14 = $\mathbf{P}_3$

Good-Turing (GT) Example

- Let's calculate **P**(abcab) using our model
- Probabilities, using the first case of normalization:
  - **P**(bb) = **P**(cc)= 1/6
  - **P**(ab) = **P**(aa) = **P**(cb) = **P**(bc)= 1/15
  - **P**(ba) = **P**(ca) = 1/10
  - **P**(ac) = 1/5
- Also need probabilities for unigrams a,b,c, compute with MLE
  - Corpus = "babaacbcacac"
  - **P**(a) = 5/12, **P**(b) = 3/12, **P**(c)=4/12
- Recall bigram approximation:

$$\mathbf{P}(abcab) \approx \mathbf{P}(a)\,\mathbf{P}(b|a)\,\mathbf{P}(c|b)\,\mathbf{P}(a|c)\,\mathbf{P}(b|a)$$

$$= \mathbf{P}(a)\frac{\mathbf{P}(ab)}{\mathbf{P}(a)}\,\frac{\mathbf{P}(bc)}{\mathbf{P}(b)}\,\frac{\mathbf{P}(ca)}{\mathbf{P}(c)}\,\frac{\mathbf{P}(ab)}{\mathbf{P}(a)}$$

$$= \frac{5}{12}\,\frac{1/15}{5/12}\,\frac{1/15}{3/12}\,\frac{1/10}{4/12}\,\frac{1/15}{5/12} \approx 0.0008533$$

Combining Estimators

- Assume we have never seen the bigrams
  - *"journal of"* $\Rightarrow$ $\mathbf{P}_{unsmoothed}(of \mid journal) = 0$
  - *"journal from"* $\Rightarrow$ $\mathbf{P}_{unsmoothed}(from \mid journal) = 0$
  - *"journal never"* $\Rightarrow$ $\mathbf{P}_{unsmoothed}(never \mid journal) = 0$
- all models we looked at so far will give the same probability to 3 bigrams above
- But intuitively, *"journal of"* is more probable because
  - *"of"* is more frequent than *"from"* & *"never"*
  - unigram probability **P**(*of*) > **P**(*from*) > **P**(*never*)

Combining Estimators

- observation:
  - unigram model suffers less from data sparseness than bigram model
  - bigram model suffers less from data sparseness than trigram model
  - …
- if we have several models of how the history predicts what comes next, we can combine them in the hope of producing an even better model

Simple Linear Interpolation

- Solve the sparseness in a trigram model by mixing with bigram and unigram models
- Also called:
  - linear interpolation
  - finite mixture models
  - deleted interpolation
- Combine linearly

$$\mathbf{P}_{li}(w_n \mid w_{n-2}, w_{n-1}) = \lambda_1 \mathbf{P}(w_n) + \lambda_2 \mathbf{P}(w_n \mid w_{n-1}) + \lambda_3 \mathbf{P}(w_n \mid w_{n-2} w_{n-1})$$

  - where $0 \leq \lambda_i \leq 1$ and $\Sigma_i \lambda_i = 1$
  - λ_i can be learned on validation data
  - search for λ_i's which maximize probability of validation data

Applications of LM

- Author / Language identification
- Hypothesis: texts that resemble each other (same author, same language) share similar characteristics
  - In English character sequence "*ing*" is more probable than in French
- Training phase:
  - pre-classified documents (known language/author)
  - construct the language model for each document class separately
- Testing phase:
  - evaluation of unknown text (comparison with language model)

Example: Language identification

- bigram of characters
  - characters = 26 letters (case insensitive)
  - possible variations: case sensitivity, punctuation, beginning/end of sentence marker, …

Example: Language Identification

	A	B	C	D	…	Y	Z
A	0.0014	0.0014	0.0014	0.0014	…	0.0014	0.0014
B	0.0014	0.0014	0.0014	0.0014	…	0.0014	0.0014
C	0.0014	0.0014	0.0014	0.0014	…	0.0014	0.0014
D	0.0042	0.0014	0.0014	0.0014	…	0.0014	0.0014
E	0.0097	0.0014	0.0014	0.0014	…	0.0014	0.0014
…	…	…	…	…	…	…	0.0014
Y	0.0014	0.0014	0.0014	0.0014	…	0.0014	0.0014
Z	0.0014	0.0014	0.0014	0.0014	0.0014	0.0014	0.0014

1. Train a language model for English
2. Train a language model for French
3. Evaluate probability of a sentence with LM-English and LM-French
4. Higher probability $\Rightarrow$ language of the sentence

Spam/Ham Classification

- Can do the same thing for ham/spam emails
- Construct character based model for ham/spam separately
  - use all 256 characters
  - punctuation is important
- For new email, evaluate its character sequence using spam character model and ham character model
- Highest probability model wins
- This is approach was the best one on our assignment 1 data, as presented in a workshop where the data comes from