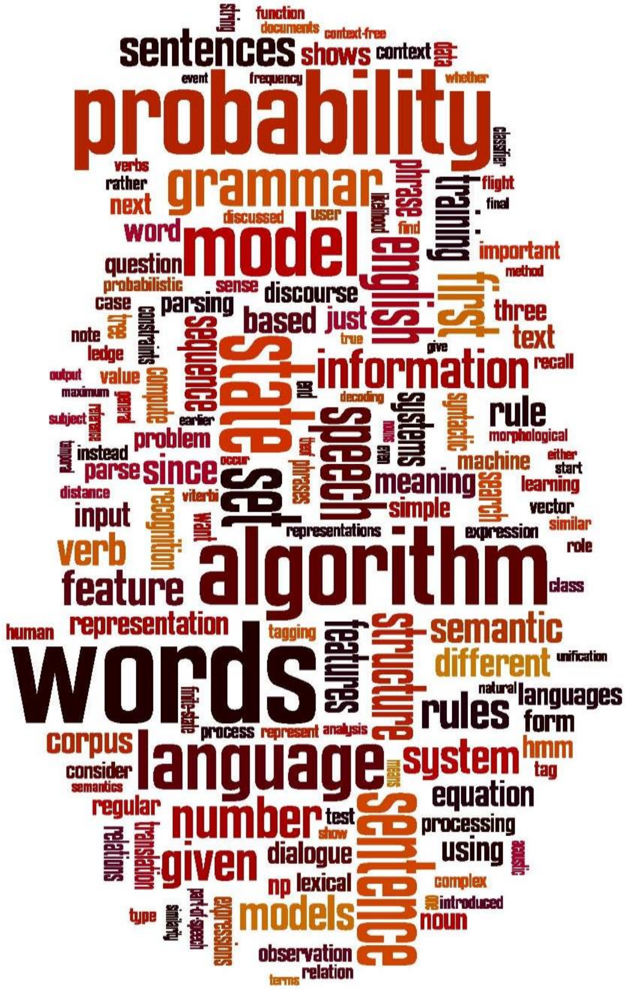


Language Modeling

Introduction to N-grams

Advanced smoothing algorithms

- 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**

Notation: N_c = Frequency of frequency c

- N_c = the count of things we've seen c times

Sam I am I am Sam I do not eat

I	3
sam	2
am	2
do	1
not	1
eat	1

$N_1 = 3$

$N_2 = 2$

$N_3 = 1$

Good-Turing smoothing intuition

- You are fishing (a scenario from Josh Goodman), and caught:
  - 10 carp, 3 perch, 2 whitefish, 1 trout, 1 salmon, 1 eel = 18 fish
- How likely is it that next species is trout?
  - 1/18
- How likely is it that next species is new (i.e. catfish or bass)
  - Let's use our estimate of things-we-saw-once to estimate the new things.
  - 3/18 (because N_1=3)
- Assuming so, how likely is it that next species is trout?
  - Must be less than 1/18
  - How to estimate?

Good Turing calculations

$$P^*_{GT}(\text{things with zero frequency}) = \frac{N_1}{N} \qquad c^* = \frac{(c+1)N_{c+1}}{N_c}$$

- Unseen (bass or catfish)
  - c = 0:
  - MLE p = 0/18 = 0
  - P^*_{GT} (unseen) = N_1/N = 3/18
- Seen once (trout)
  - c = 1
  - MLE p = 1/18
  - C*(trout) = 2 * N2/N1

    = 2 * 1/3

    = 2/3
  - P^*_{GT}(trout) = **c*/N**= 2/3 / 18 = 1/27

Resulting Good-Turing numbers

- Numbers from Church and Gale (1991)
- 22 million words of AP Newswire

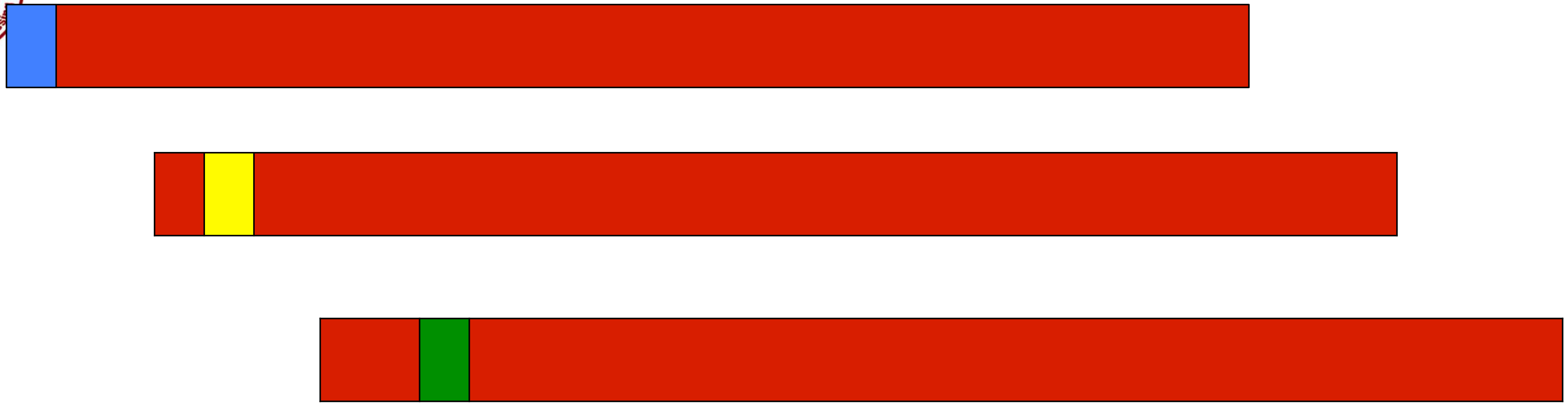
$$c^* = \frac{(c+1)N_{c+1}}{N_c}$$

- It sure looks like c* = (c –.75)

Count c	Good Turing c*
0	.0000270
1	0.446
2	1.26
3	2.24
4	3.24
5	4.22
6	5.19
7	6.21
8	7.24
9	8.25

Ney et al.’s Good Turing Intuition

H. Ney, U. Essen, and R. Kneser, 1995. On the estimation of 'small' probabilities by leaving-one-out. IEEE Trans. PAMI. 17:12,1202--1212

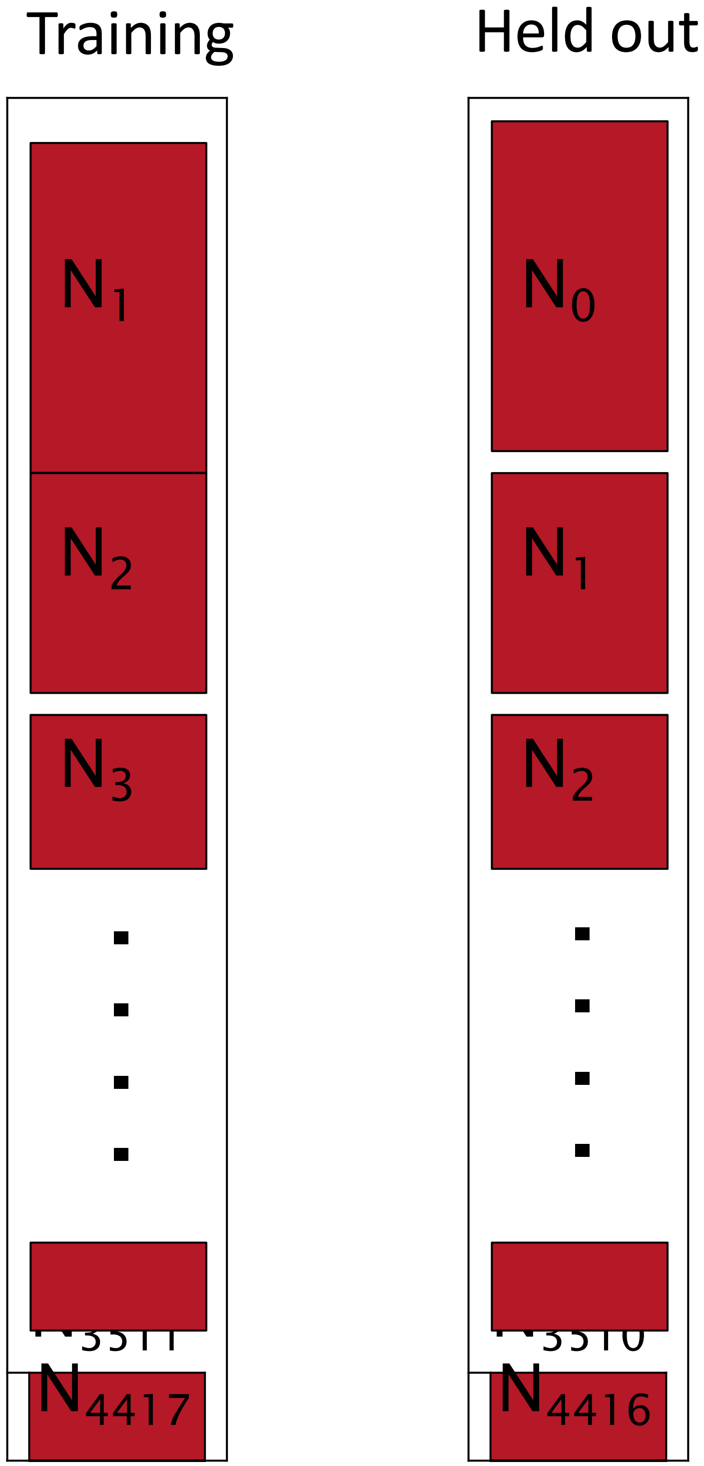


Held-out words:

Dan Jurafsky

Ney *et al.* Good Turing Intuition (slide from Dan Klein)

- Intuition from leave-one-out validation
  - Take each of the *c* training words out in turn
  - *c* training sets of size *c*–1, held-out of size 1
  - What fraction of held-out words are unseen in training?
    - N_1/c *(that occurred once)*
  - What fraction of held-out words are seen *k* times in training?
    - $(k+1)N_{k+1}/c$
  - So in the future we expect $(k+1)N_{k+1}/c$ of the words to be those with training count *k*
  - There are N_k words with training count *k*
  - Each should occur with probability:
    - $(k+1)N_{k+1}/c/N_k$
- …or expected count: $$k^* = \frac{(k+1)N_{k+1}}{N_k}$$

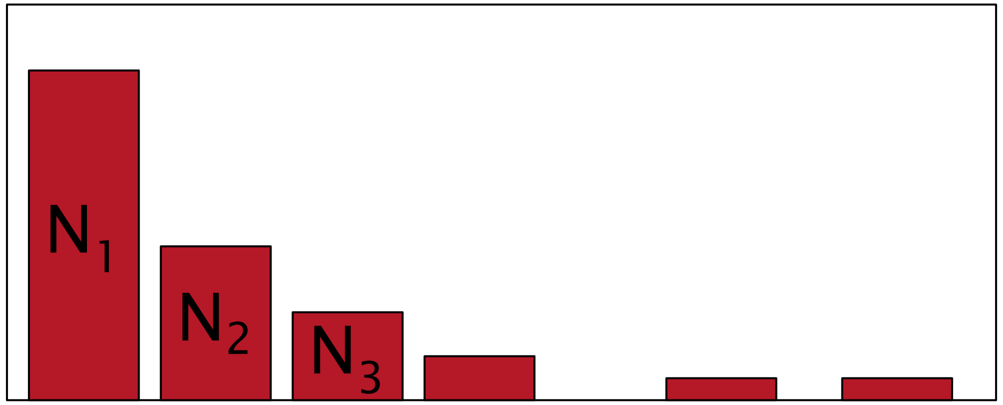


Dan Jurafsky

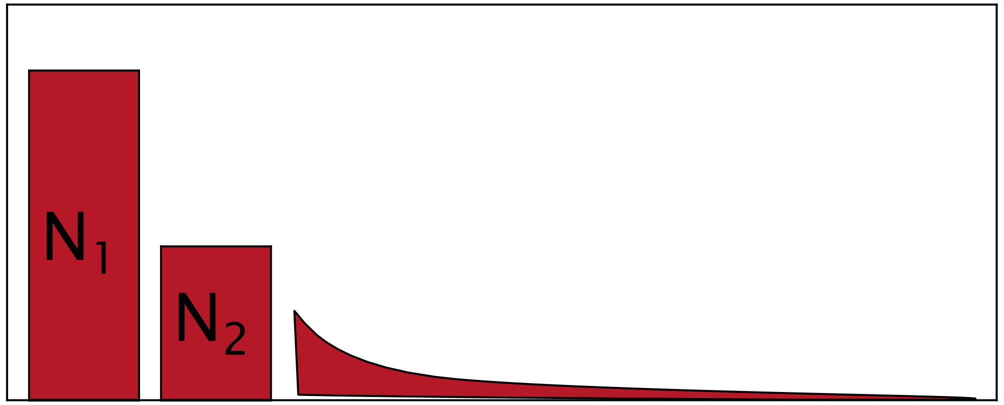
Good-Turing complications

(slide from Dan Klein)

- Problem: what about “the”? (say c=4417)
  - For small k, $N_k > N_{k+1}$
  - For large k, too jumpy, zeros wreck estimates



- Simple Good-Turing [Gale and Sampson]: replace empirical N_k with a best-fit power law once counts get unreliable
- The best-fit power law is a model that predicts the probability of an event based on its frequency, and it is chosen to fit the observed counts as closely as possible.



Resulting Good-Turing numbers

- Numbers from Church and Gale (1991)
- 22 million words of AP Newswire

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

Count c	Good Turing c*
0	.0000270
1	0.446
2	1.26
3	2.24
4	3.24
5	4.22
6	5.19
7	6.21
8	7.24
9	8.25

Absolute Discounting

- Save ourselves some time and just subtract 0.75 (or some d)!

discounted bigram

$$P_{\text{AbsoluteDiscounting}}(w_i \mid w_{i-1}) = \max(\frac{c(w_{i-1}, w_i) - d, 0)}{c(w_{i-1})}$$

- (Maybe keeping a couple extra values of d for counts 1 and 2)

- d=0.75 for c>2, smaller d for c<2

- P(Francisco|san)= (c(san francisco) – 0.75)/c(san)

Absolute Discounting Interpolation

- Save ourselves some time and just subtract 0.75 (or some d)!

discounted bigram

Interpolation weight

$$P_{\text{AbsoluteDiscounting}}(w_i \mid w_{i-1}) = \frac{c(w_{i-1}, w_i) - d}{c(w_{i-1})} + \lambda(w_{i-1})P(w)$$

unigram

- But should we really just use the regular unigram P(w)?
- Use continuation probability
  - Because of discounting you can estimate lambda directly don't need held out dataset for it
  - Lambda (san)= d* num of distinct words following san/c(san)

Kneser-Ney Smoothing I

- Better estimate for probabilities of lower--order unigrams!
  - Shannon game: *I can't see without my reading___Francisco_?*
  - "Francisco" is more common than "glasses"
  - … but "Francisco" always follows "San"
- The unigram is useful exactly when we haven't seen this bigram!
- Instead of P(w): "How likely is w"
- $P_{continuation}$(w): "How likely is w to appear as a novel continuation?
  - For each word, count the number of bigram types it completes
  - Every bigram type was a novel continuation the first time it was seen

$$P_{CONTINUATION}(w) \propto \left|\{w_{i-1} : c(w_{i-1}, w) > 0\}\right|$$

Kneser-Ney Smoothing II

- How many times does w appear as a novel continuation:

$$P_{CONTINUATION}(w) \propto \left|\{w_{i-1} : c(w_{i-1}, w) > 0\}\right|$$

- Normalized by the total number of word bigram types

$$\left|\{(w_{j-1}, w_j) : c(w_{j-1}, w_j) > 0\}\right|$$

$$P_{CONTINUATION}(w) = \frac{\left|\{w_{i-1} : c(w_{i-1}, w) > 0\}\right|}{\left|\{(w_{j-1}, w_j) : c(w_{j-1}, w_j) > 0\}\right|}$$

Francisco is a continuation of how many unique words?

Count of all non-zero bigrams