Language Modeling

Interpolation, Backoff, and Web-Scale LMs

Backoff and Interpolation

- Sometimes it helps to use **less** context
  - Condition on less context for contexts you haven't learned much about
- **Backoff:**
  - use trigram if you have good evidence,
  - otherwise bigram, otherwise unigram
- **Interpolation:**
  - mix unigram, bigram, trigram

- Interpolation works better

Dan Jurafsky

Linear 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** corpus

Training Data | Held-Out Data | Test Data

- Choose λs to maximize the probability of held-out data:
  - Fix the N-gram probabilities (on the training data)
  - Then search for λs that give largest probability to held-out set:

$$\log P(w_1 \ldots w_n \mid M(\lambda_1 \ldots \lambda_k)) = \sum_i \log P_{M(\lambda_1 \ldots \lambda_k)}(w_i \mid w_{i-1})$$

Dan Jurafsky

Unknown words: Open versus closed vocabulary tasks

- If we know all the words in advanced
  - Vocabulary V is fixed
  - Closed vocabulary task
- Often we don't know this
  - **Out Of Vocabulary** = OOV words
  - Open vocabulary task
- Instead: create an unknown word token <UNK>
  - Training of <UNK> probabilities
    - Create a fixed lexicon L of size V
    - At text normalization phase, any training word not in L changed to <UNK>
    - Now we train its probabilities like a normal word
  - At decoding time
    - If text input: Use UNK probabilities for any word not in training

Dan Jurafsky

Huge web-scale n-grams

- How to deal with, e.g., Google N-gram corpus
- Pruning
  - Only store N-grams with count > threshold.
    - Remove singletons of higher-order n-grams
  - Entropy-based pruning
- Efficiency
  - Efficient data structures like tries
  - Bloom filters: approximate language models
  - Store words as indexes, not strings
    - Use Huffman coding to fit large numbers of words into two bytes
  - Quantize probabilities (4-8 bits instead of 8-byte float)

Smoothing for Web-scale N-grams

- "Stupid backoff" (Brants *et al*. 2007)
- No discounting, just use relative frequencies

$$S(w_i \mid w_{i-k+1}^{i-1}) = \begin{cases} \dfrac{\text{count}(w_{i-k+1}^{i})}{\text{count}(w_{i-k+1}^{i-1})} & \text{if } \text{count}(w_{i-k+1}^{i}) > 0 \\ 0.4 S(w_i \mid w_{i-k+2}^{i-1}) & \text{otherwise} \end{cases}$$

$$S(w_i) = \frac{\text{count}(w_i)}{N}$$

N-gram Smoothing Summary

- Add-1 smoothing:
  - OK for text categorization, not for language modeling
- The most commonly used method:
  - Extended Interpolated Kneser-Ney
- For very large N-grams like the Web:
  - Stupid backoff

Advanced Language Modeling

- Discriminative models:
  - choose n-gram weights to improve a task, not to fit the training set
- Parsing-based models
- Caching Models
  - Recently used words are more likely to appear

$$P_{CACHE}(w \mid history) = \lambda P(w_i \mid w_{i-2} w_{i-1}) + (1-\lambda)\frac{c(w \in history)}{\mid history \mid}$$

  - These perform very poorly for speech recognition (why?)

Language Modeling

Interpolation, Backoff, and Web-Scale LMs

Language Modeling

Advanced: Good Turing Smoothing

Reminder: Add-1 (Laplace) Smoothing

$$P_{Add-1}(w_i \mid w_{i-1}) = \frac{c(w_{i-1}, w_i) + 1}{c(w_{i-1}) + V}$$

Dan Jurafsky

More general formulations: Add-k

$$P_{Add-k}(w_i \mid w_{i-1}) = \frac{c(w_{i-1}, w_i) + k}{c(w_{i-1}) + kV}$$

$$P_{Add-k}(w_i \mid w_{i-1}) = \frac{c(w_{i-1}, w_i) + m(\frac{1}{V})}{c(w_{i-1}) + m}$$

Dan Jurafsky

Unigram prior smoothing

$$P_{Add-k}(w_i \mid w_{i-1}) = \frac{c(w_{i-1}, w_i) + m(\frac{1}{V})}{c(w_{i-1}) + m}$$

$$P_{\mathrm{UnigramPrior}}(w_i \mid w_{i-1}) = \frac{c(w_{i-1}, w_i) + mP(w_i)}{c(w_{i-1}) + m}$$

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**

Dan Jurafsky

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
```

$N_1 = 3$

$N_2 = 2$

$N_3 = 1$

Dan Jurafsky

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?

Dan Jurafsky

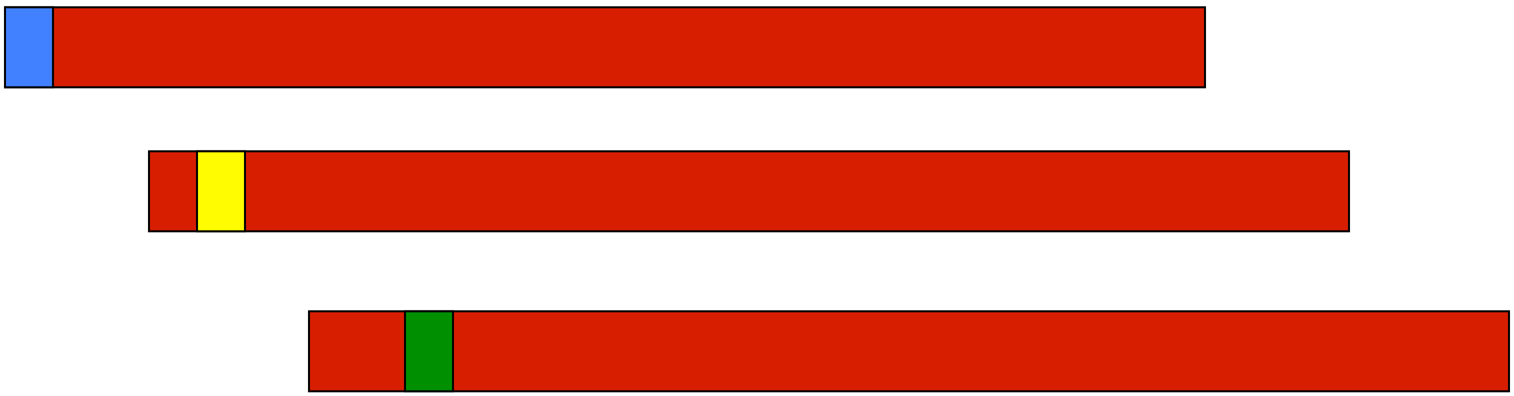
Good Turing calculations

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

- 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) = 2/3 / 18 = 1/27

Ney et al.'s Good Turing Intuition

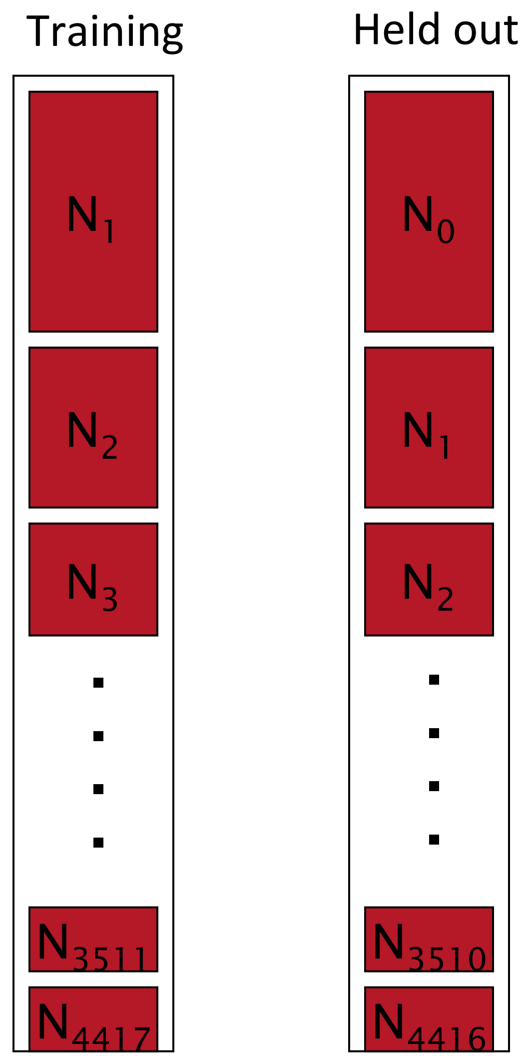
H. Ney, U. Essen, and R. Kneser, 1995. On the estimation of 'small' probabilities by leaving-one-out. IEEE Trans. PAMI. 17:12,1202-1212



Held-out words:

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
  - 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: $$k^* = \frac{(k+1)N_{k+1}}{N_k}$$

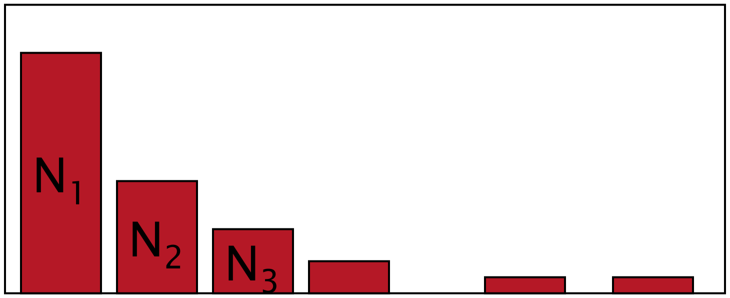


Dan Jurafsky

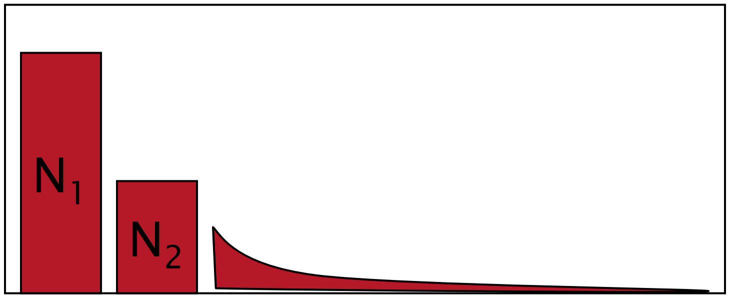
Good-Turing complications

(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



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

Language Modeling

Advanced: Good Turing Smoothing

Language Modeling

Advanced: Kneser-Ney Smoothing

Dan Jurafsky

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

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

- (Maybe keeping a couple extra values of d for counts 1 and 2)
- But should we really just use the regular unigram P(w)?

Kneser-Ney Smoothing I

- Better estimate for probabilities of lower-order unigrams!
  - Shannon game: *I can't see without my reading*___Francisco glasses___?
  - "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

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

Dan Jurafsky

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

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

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

Dan Jurafsky

Kneser-Ney Smoothing III

- Alternative metaphor: The number of # of word types seen to precede w

$$|\{w_{i-1} : c(w_{i-1}, w) > 0\}|$$

- normalized by the # of words preceding all words:

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

- A frequent word (Francisco) occurring in only one context (San) will have a low continuation probability

Dan Jurafsky

Kneser-Ney Smoothing IV

$$P_{KN}(w_i \mid w_{i-1}) = \frac{\max(c(w_{i-1},w_i) - d, 0)}{c(w_{i-1})} + \lambda(w_{i-1}) P_{CONTINUATION}(w_i)$$

λ is a normalizing constant; the probability mass we've discounted

$$\lambda(w_{i-1}) = \frac{d}{c(w_{i-1})} \left| \{ w : c(w_{i-1}, w) > 0 \} \right|$$

the normalized discount

The number of word types that can follow w_{i-1}
= # of word types we discounted
= # of times we applied normalized discount

Dan Jurafsky

Kneser-Ney Smoothing: Recursive formulation

$$P_{KN}(w_i \mid w_{i-n+1}^{i-1}) = \frac{\max(c_{KN}(w_{i-n+1}^{i}) - d, 0)}{c_{KN}(w_{i-n+1}^{i-1})} + \lambda(w_{i-n+1}^{i-1}) P_{KN}(w_i \mid w_{i-n+2}^{i-1})$$

$$c_{KN}(\bullet) = \begin{cases} count(\bullet) & \text{for the highest order} \\ continuationcount(\bullet) & \text{for lower order} \end{cases}$$

Continuation count = Number of unique single word contexts for •

Language Modeling

Advanced:
Kneser-Ney Smoothing