

# Language Modeling with Smoothing

Instructor: Wei Xu

Many slides from Dan Jurafsky

# Language Modeling

## Evaluation and Perplexity

# Perplexity

The best language model is one that best predicts an unseen test set

- Gives the highest  $P(\text{sentence})$

Perplexity is the inverse probability of the test set, normalized by the number of word:

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

*equivalently:*

$$PP(W) = 2^{-l}$$

$$\text{where } l = \frac{1}{N} \log P(w_1 w_2 \dots w_N)$$

per word log likelihood

Minimizing perplexity is the same as maximizing probability

Lower perplexity = better model

- Training 38 million words, test 1.5 million words, Wall Street Journal

N-gram Order	Unigram	Bigram	Trigram
Perplexity	962	170	109

# Perplexity as branching factor

- Let's suppose a sentence consisting of random digits [0,1, ..., 9]
- What is the perplexity of this sentence according to a model that assign  $P=1/10$  to each digit?

$$\begin{aligned} \text{PP}(W) &= P(w_1 w_2 \dots w_N)^{-\frac{1}{N}} \\ &= \left(\frac{1}{10}\right)^N^{-\frac{1}{N}} \\ &= \frac{1}{10}^{-1} \\ &= 10 \end{aligned}$$

# Perplexity as branching factor

- If one could report a model perplexity of 247 ( $2^{7.95}$ ) per word
- In other words, the model is as confused on test data as if it had to choose uniformly and independently among 247 possibilities for each word.
- But,
  - a trigram language model can get perplexity of 247 on the Brown Corpus
  - simply guessing that the next word in the Brown Corpus is “the” have an accuracy of 7%, not 1/247

Q: Why?

# Language Modeling

Smoothing: Add-one  
(Laplace) smoothing

# Zeros

- Training set:
  - ... denied the allegations
  - ... denied the reports
  - ... denied the claims
  - ... denied the request
- Test set
  - ... denied the offer
  - ... denied the loan

$$P(\text{"offer"} \mid \text{denied the}) = 0$$

# The intuition of smoothing (from Dan Klein)

- When we have sparse statistics:

$P(w | \text{denied the})$

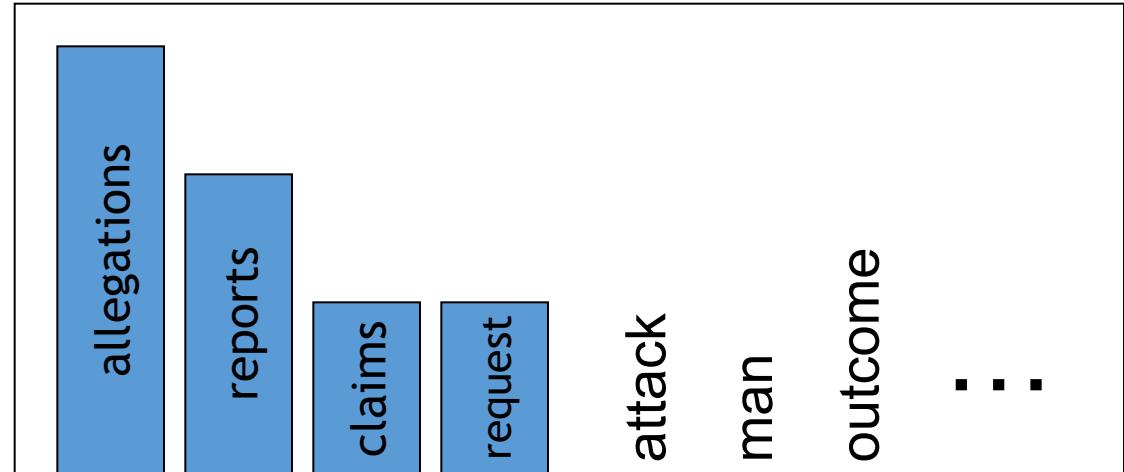
3 allegations

2 reports

1 claims

1 request

7 total



- Steal probability mass to generalize better

$P(w | \text{denied the})$

2.5 allegations

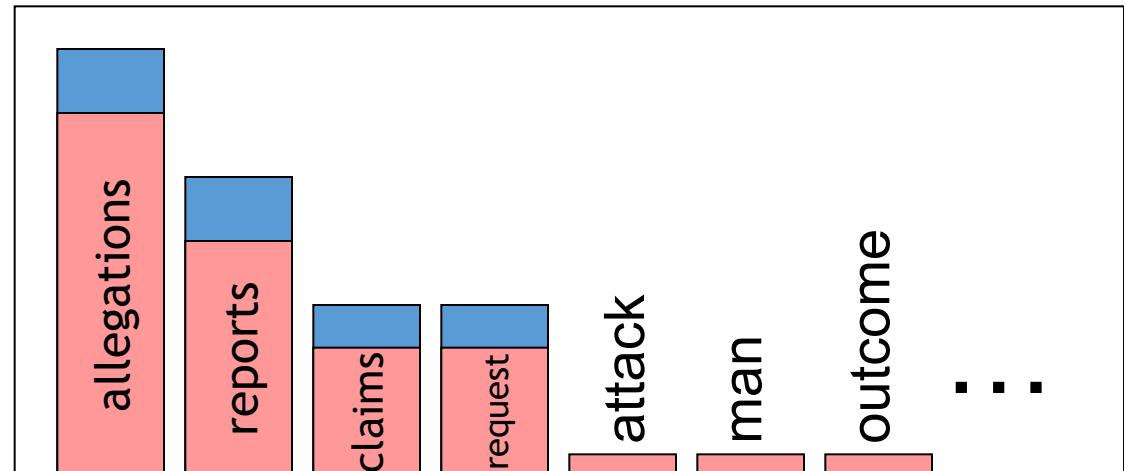
1.5 reports

0.5 claims

0.5 request

**2 other**

7 total



# Add-one estimation

- Also called Laplace smoothing
- Pretend we saw each word one more time than we did
- Just add one to all the counts!
- MLE estimate:
- Add-1 estimate:

$$P_{MLE}(w_i | w_{i-1}) = \frac{c(w_{i-1}, w_i)}{c(w_{i-1})}$$

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

# Berkeley Restaurant Corpus: 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 bigrams

	i	want	to	eat	chinese	food	lunch	spend
i	0.0015	0.21	<b>0.00025</b>	0.0025	<b>0.00025</b>	<b>0.00025</b>	<b>0.00025</b>	0.00075
want	0.0013	<b>0.00042</b>	0.26	0.00084	0.0029	0.0029	0.0025	0.00084
to	0.00078	<b>0.00026</b>	0.0013	0.18	0.00078	<b>0.00026</b>	0.0018	0.055
eat	<b>0.00046</b>	<b>0.00046</b>	0.0014	<b>0.00046</b>	0.0078	0.0014	0.02	<b>0.00046</b>
chinese	0.0012	<b>0.00062</b>	<b>0.00062</b>	<b>0.00062</b>	<b>0.00062</b>	0.052	0.0012	<b>0.00062</b>
food	0.0063	<b>0.00039</b>	0.0063	<b>0.00039</b>	0.00079	0.002	<b>0.00039</b>	0.00039
lunch	0.0017	<b>0.00056</b>	<b>0.00056</b>	<b>0.00056</b>	<b>0.00056</b>	0.0011	<b>0.00056</b>	<b>0.00056</b>
spend	0.0012	<b>0.00058</b>	0.0012	<b>0.00058</b>	<b>0.00058</b>	<b>0.00058</b>	<b>0.00058</b>	<b>0.00058</b>

# Exercise

# Language Modeling

Smoothing: Add-one  
(Laplace) smoothing

# The intuition of smoothing (from Dan Klein)

- When we have sparse statistics:

$P(w \mid \text{denied the})$

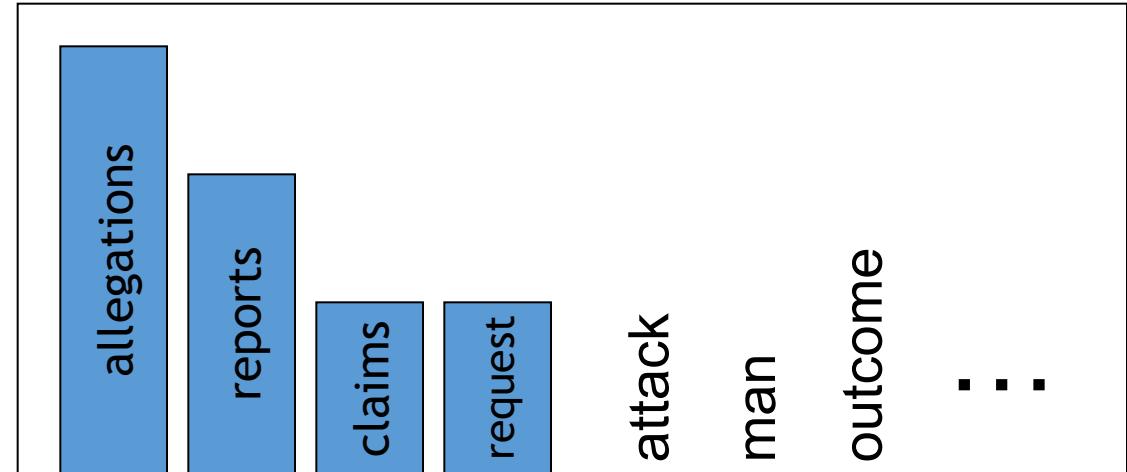
3 allegations

2 reports

1 claims

1 request

7 total



- Steal probability mass to generalize better

$P(w \mid \text{denied the})$

2.5 allegations

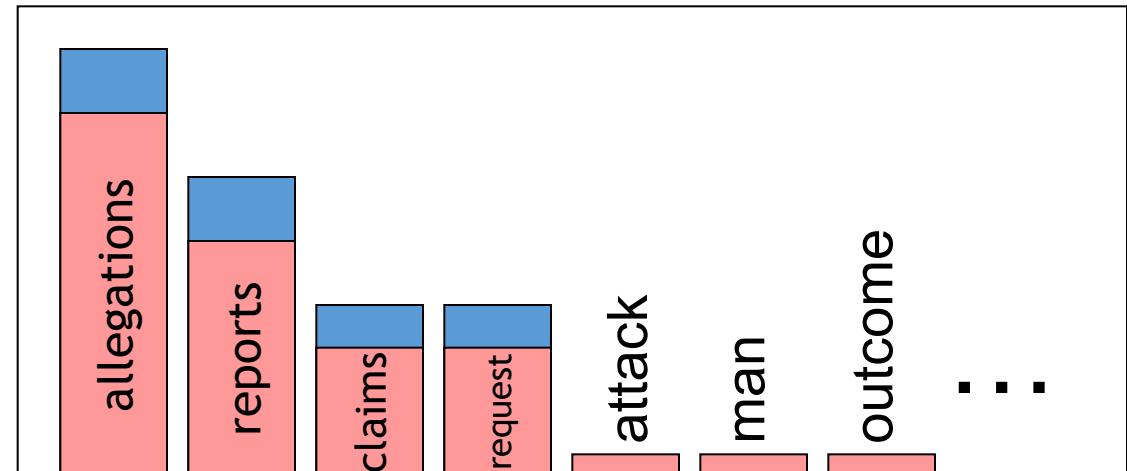
1.5 reports

0.5 claims

0.5 request

**2 other**

7 total



# Add-one estimation

- Also called Laplace smoothing
- Pretend we saw each word one more time than we did
- Just add one to all the counts!
- MLE estimate:
- Add-1 estimate:

$$P_{MLE}(w_i | w_{i-1}) = \frac{c(w_{i-1}, w_i)}{c(w_{i-1})}$$

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

# Berkeley Restaurant Corpus: 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 bigrams

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

	i	want	to	eat	chinese	food	lunch	spend
i	0.0015	0.21	<b>0.00025</b>	0.0025	<b>0.00025</b>	<b>0.00025</b>	<b>0.00025</b>	0.00075
want	0.0013	<b>0.00042</b>	0.26	0.00084	0.0029	0.0029	0.0025	0.00084
to	0.00078	<b>0.00026</b>	0.0013	0.18	0.00078	<b>0.00026</b>	0.0018	0.055
eat	<b>0.00046</b>	<b>0.00046</b>	0.0014	<b>0.00046</b>	0.0078	0.0014	0.02	<b>0.00046</b>
chinese	0.0012	<b>0.00062</b>	<b>0.00062</b>	<b>0.00062</b>	<b>0.00062</b>	0.052	0.0012	<b>0.00062</b>
food	0.0063	<b>0.00039</b>	0.0063	<b>0.00039</b>	0.00079	0.002	<b>0.00039</b>	0.00039
lunch	0.0017	<b>0.00056</b>	<b>0.00056</b>	<b>0.00056</b>	<b>0.00056</b>	0.0011	<b>0.00056</b>	<b>0.00056</b>
spend	0.0012	<b>0.00058</b>	0.0012	<b>0.00058</b>	<b>0.00058</b>	<b>0.00058</b>	<b>0.00058</b>	<b>0.00058</b>

# 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	<b>0.64</b>	6.4	<b>0.64</b>	<b>0.64</b>	<b>0.64</b>	1.9
want	1.2	<b>0.39</b>	238	0.78	2.7	2.7	2.3	0.78
to	1.9	<b>0.63</b>	3.1	430	1.9	<b>0.63</b>	4.4	133
eat	<b>0.34</b>	0.34	1	<b>0.34</b>	5.8	1	15	<b>0.34</b>
chinese	<b>0.2</b>	<b>0.098</b>	<b>0.098</b>	<b>0.098</b>	<b>0.098</b>	8.2	<b>0.2</b>	<b>0.098</b>
food	6.9	0.43	6.9	<b>0.43</b>	0.86	2.2	<b>0.43</b>	0.43
lunch	0.57	<b>0.19</b>	<b>0.19</b>	<b>0.19</b>	<b>0.19</b>	0.38	<b>0.19</b>	<b>0.19</b>
spend	0.32	<b>0.16</b>	0.32	<b>0.16</b>	<b>0.16</b>	<b>0.16</b>	<b>0.16</b>	<b>0.16</b>

Language Modeling

Generalization and  
zeros

# The Shannon Visualization Method

- Choose a random bigram ( $\langle s \rangle$ , w) according to its probability
- Now choose a random bigram (w, x) according to its probability
- And so on until we choose  $\langle /s \rangle$
- Then string the words together

$\langle s \rangle$  I  
I want  
want to  
to eat  
eat Chinese  
Chinese food  
food  $\langle /s \rangle$

I want to eat Chinese food

Q: How do you write a program and do it?

# Approximating Shakespeare

- |           |   |
|-----------|---|
| 1<br>gram | -To him swallowed confess hear both. Which. Of save on trail for are ay device and<br>rote life have              |
|           | -Hill he late speaks; or! a more to leg less first you enter  |
| 2<br>gram | -Why dost stand forth thy canopy, forsooth; he is this palpable hit the King Henry. Live<br>king. Follow.         |
|           | -What means, sir. I confess she? then all sorts, he is trim, captain.   |
| 3<br>gram | -Fly, and will rid me these news of price. Therefore the sadness of parting, as they say,<br>'tis done.           |
|           | -This shall forbid it should be branded, if renown made it empty.   |
| 4<br>gram | -King Henry. What! I will go seek the traitor Gloucester. Exeunt some of the watch. A<br>great banquet serv'd in; |
|           | -It cannot be but so.   |

# Shakespeare as corpus

- $N=884,647$  tokens,  $V=29,066$
- 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)
- Quadrigrams (4-gram) worse:
  - What's coming out looks like Shakespeare because it *is* Shakespeare

# The wall street journal is not Shakespeare (no offense)

1  
gram

Months the my and issue of year foreign new exchange's september were recession exchange new endorsed a acquire to six executives

2  
gram

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

3  
gram

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

# The perils of overfitting

- N-grams only work well for word prediction if the test corpus looks like the training corpus
  - In real life, it often doesn't
  - We need to train robust models that generalize!
  - One kind of generalization: Zeros!
    - Things that don't ever occur in the training set
    - but occur in the test set

# (Recall) Raw bigram counts

- Out of 9222 sentences

	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

# Zero probability bigrams

- Bigrams with zero probability
  - mean that we will assign 0 probability to the test set!
- And hence we cannot compute perplexity (can't divide by 0)!

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

for bigram

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

Q: How do we deal with bigrams with zero probability?

# Language Modeling

Smoothing: Add-one  
(Laplace) smoothing

# The intuition of smoothing (from Dan Klein)

- When we have sparse statistics:

$P(w | \text{denied the})$

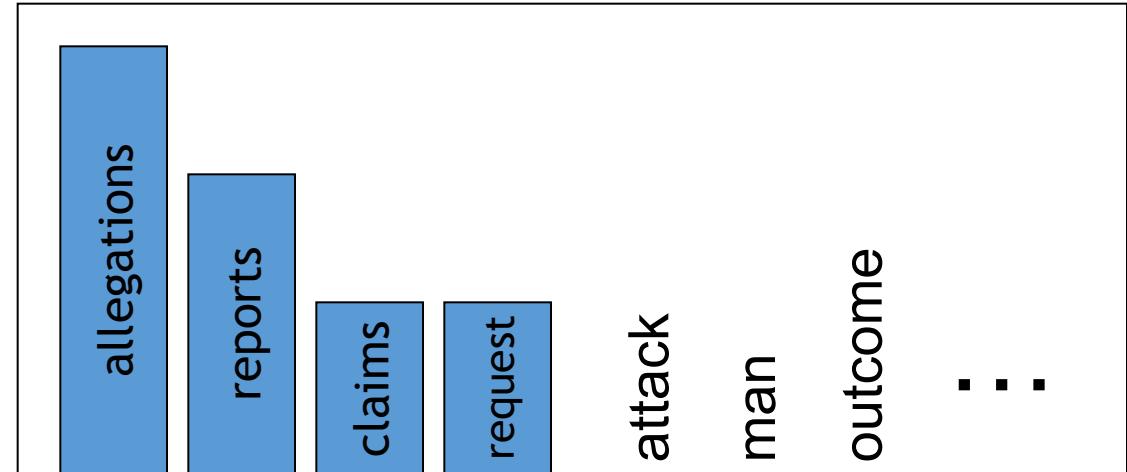
3 allegations

2 reports

1 claims

1 request

7 total



- Steal probability mass to generalize better

$P(w | \text{denied the})$

2.5 allegations

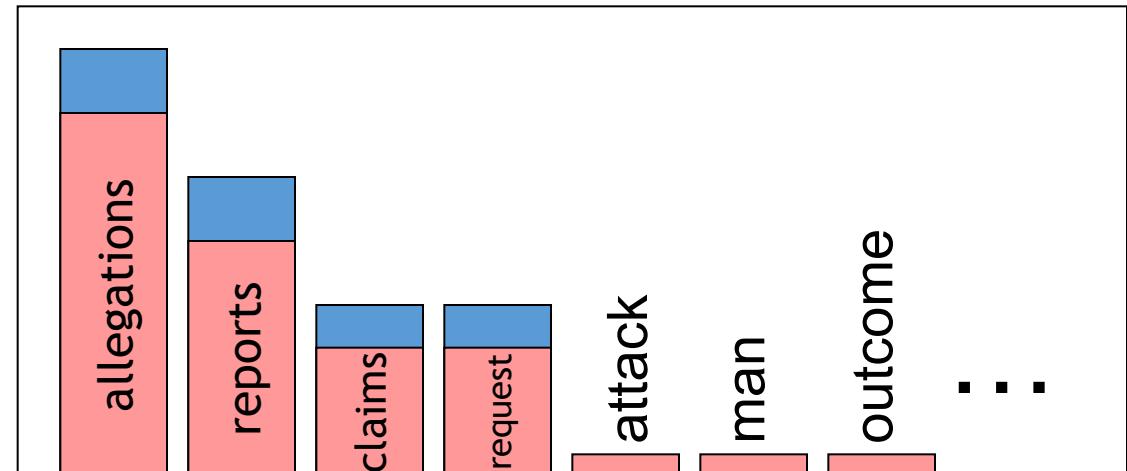
1.5 reports

0.5 claims

0.5 request

**2 other**

7 total



# Add-one estimation

- Also called Laplace smoothing
- Pretend we saw each word one more time than we did
- Just add one to all the counts!
- MLE estimate:
- Add-1 estimate:

$$P_{MLE}(w_i | w_{i-1}) = \frac{c(w_{i-1}, w_i)}{c(w_{i-1})}$$

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

# Berkeley Restaurant Corpus: 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 bigrams

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

	i	want	to	eat	chinese	food	lunch	spend
i	0.0015	0.21	<b>0.00025</b>	0.0025	<b>0.00025</b>	<b>0.00025</b>	<b>0.00025</b>	0.00075
want	0.0013	<b>0.00042</b>	0.26	0.00084	0.0029	0.0029	0.0025	0.00084
to	0.00078	<b>0.00026</b>	0.0013	0.18	0.00078	<b>0.00026</b>	0.0018	0.055
eat	<b>0.00046</b>	<b>0.00046</b>	0.0014	<b>0.00046</b>	0.0078	0.0014	0.02	<b>0.00046</b>
chinese	0.0012	<b>0.00062</b>	<b>0.00062</b>	<b>0.00062</b>	<b>0.00062</b>	0.052	0.0012	<b>0.00062</b>
food	0.0063	<b>0.00039</b>	0.0063	<b>0.00039</b>	0.00079	0.002	<b>0.00039</b>	0.00039
lunch	0.0017	<b>0.00056</b>	<b>0.00056</b>	<b>0.00056</b>	<b>0.00056</b>	0.0011	<b>0.00056</b>	<b>0.00056</b>
spend	0.0012	<b>0.00058</b>	0.0012	<b>0.00058</b>	<b>0.00058</b>	<b>0.00058</b>	<b>0.00058</b>	<b>0.00058</b>

# 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	<b>0.64</b>	6.4	<b>0.64</b>	<b>0.64</b>	<b>0.64</b>	1.9
want	1.2	<b>0.39</b>	238	0.78	2.7	2.7	2.3	0.78
to	1.9	<b>0.63</b>	3.1	430	1.9	<b>0.63</b>	4.4	133
eat	<b>0.34</b>	0.34	1	<b>0.34</b>	5.8	1	15	<b>0.34</b>
chinese	<b>0.2</b>	<b>0.098</b>	<b>0.098</b>	<b>0.098</b>	<b>0.098</b>	8.2	<b>0.2</b>	<b>0.098</b>
food	6.9	0.43	6.9	<b>0.43</b>	0.86	2.2	<b>0.43</b>	0.43
lunch	0.57	<b>0.19</b>	<b>0.19</b>	<b>0.19</b>	<b>0.19</b>	0.38	<b>0.19</b>	<b>0.19</b>
spend	0.32	<b>0.16</b>	0.32	<b>0.16</b>	<b>0.16</b>	<b>0.16</b>	<b>0.16</b>	<b>0.16</b>

# Compare with raw bigram counts

	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

	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

# Add-1 estimation is a blunt instrument

- So add-1 isn't used for N-grams:
  - We'll see better methods
- But add-1 is used to smooth other NLP models
  - For text classification
  - In domains where the number of zeros isn't so huge.

# 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

# Linear Interpolation

- Simple interpolation

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

- Lambdas conditional on context:

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

# How to set the lambdas?

- Use a **held-out** corpus

Training Data

Held-Out  
Data

Test  
Data

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

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

# Unknown words: Open vs. Closed Vocabulary

- 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

# 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

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

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

# Big language models help machine translation a lot

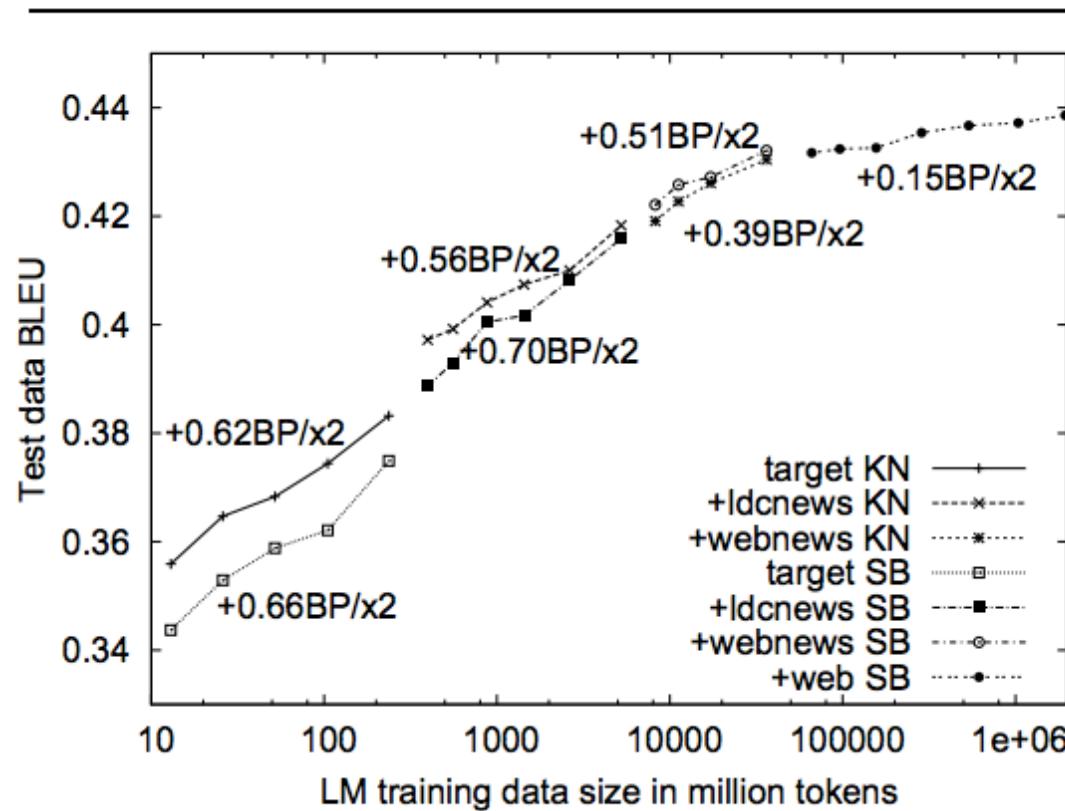


Figure 5: BLEU scores for varying amounts of data using Kneser-Ney (KN) and Stupid Backoff (SB).

# 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

- These perform very poorly for speech recognition (why?)
$$P_{CACHE}(w | history) = \lambda P(w_i | w_{i-2} w_{i-1}) + (1 - \lambda) \frac{\alpha(w \in history)}{|history|}$$

# Language Modeling

Advanced:  
Kneser-Ney Smoothing

# Absolute discounting: just subtract a little from each count

- Suppose we wanted to subtract a little from a count of 4 to save probability mass for the zeros
- How much to subtract ?
- Church and Gale (1991)'s clever idea
- Divide up 22 million words of AP Newswire
  - Training and held-out set
  - for each bigram in the training set
  - see the actual count in the held-out set!
- It sure looks like  $c^* = (c - .75)$

Bigram count in training	Bigram count in heldout set
0	.0000270
1	0.448
2	1.25
3	2.24
4	3.23
5	4.21
6	5.23
7	6.21
8	7.21
9	8.26

# Absolute Discounting Interpolation

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

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

discounted bigram                                  Interpolation weight

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 Fglasses*?
  - “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  $w$ ”
- $P_{\text{continuation}}(w)$ : “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

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

# Kneser-Ney Smoothing II

- How many times does  $W$  appear as a novel continuation:

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

- Normalized by the total number of word bigram types

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

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

# Kneser-Ney Smoothing IV

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

absolute discount      lower order weight      lower order probability

$\lambda$  is a normalizing constant; the probability mass we've discounted for higher order

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

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

# Kneser-Ney Smoothing: Recursive formulation

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

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

Continuation count = Number of unique single word contexts for  $\bullet$

# Language Modeling

Advanced:  
Kneser-Ney Smoothing