

Recap of the last lecture

- N-gram Language Models
 - Unigram, bigram, trigram...
 - Markov assumption
 - Computing individual probabilities

Perplexity

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

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

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

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

Chain rule:

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

For bigrams:

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

Minimizing perplexity is the same as maximizing probability

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

```
<s> I
    I want
      want to
        to eat
          eat Chinese
            Chinese food
              food </s>

I want to eat Chinese food
```

Approximating 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?

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 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 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

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

	eat	chinese	food	lunch	spend
i	0.002	0.33	0	0.0036	0
want	0.0072	0	0.014	0.0063	0.0045
to	0.00083	0	0.0017	0.28	0.00083
eat	0	0	0.0027	0	0.021
chinese	0.0043	0	0	0	0.0027
food	0.014	0	0.014	0	0.00092
lunch	0.0059	0	0	0	0.0029
spend	0.0036	0	0.0036	0	0

Zeros

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

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

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)!
- Zero mitigation
 - Various **smoothing** techniques

Basic Smoothing: Interpolation and Back-off

The intuition of smoothing

- When we have sparse statistics:

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

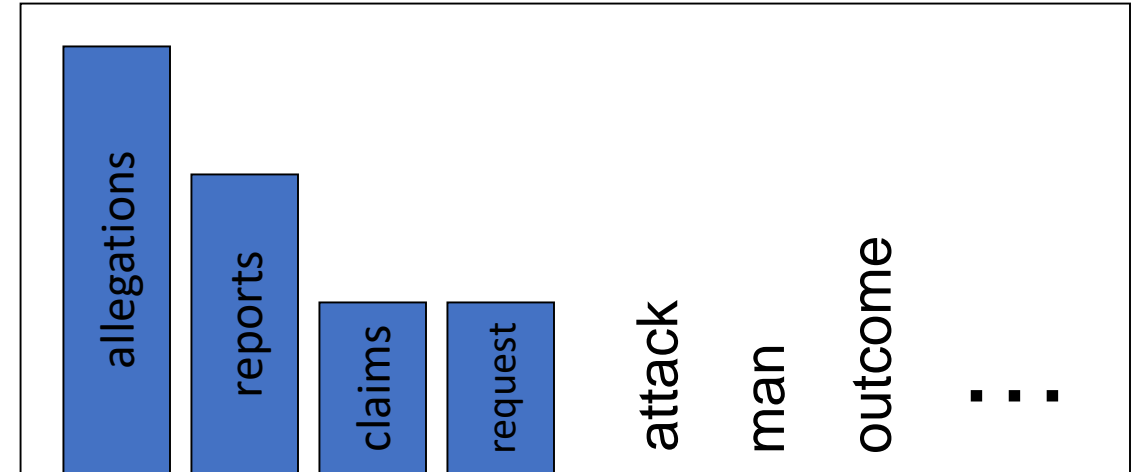
3 allegations

2 reports

1 claims

1 request

7 total



- Steal probability mass to generalize better

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

2.5 allegations

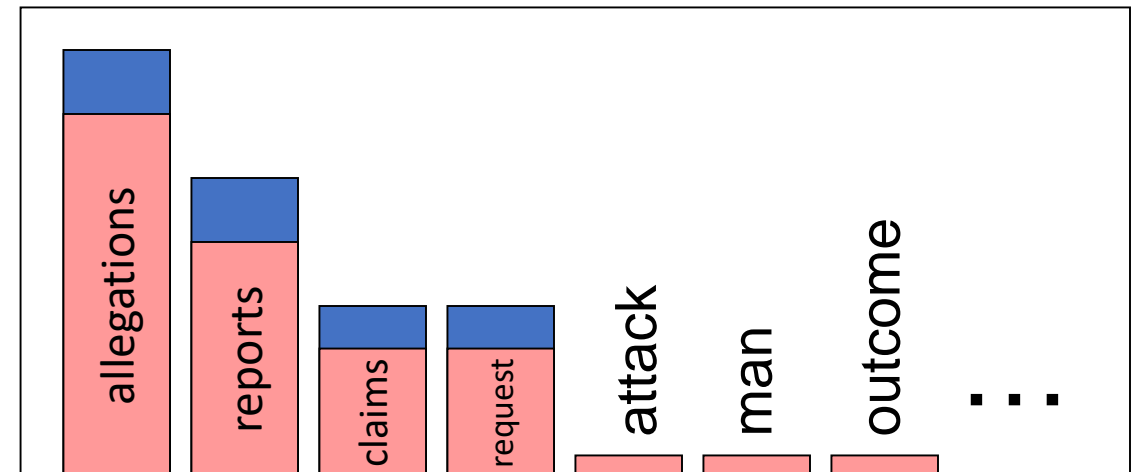
1.5 reports

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

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

Maximum Likelihood Estimates

- The maximum likelihood estimate
 - of some parameter of a model M from a training set T
 - maximizes the likelihood of the training set T given the model M
- Suppose the word “bagel” occurs 400 times in a corpus of a million words
- What is the probability that a random word from some other text will be “bagel”?
- MLE estimate is $400/1,000,000 = .0004$
- This may be a bad estimate for some other corpus
 - But it is the **estimate** that makes it **most likely** that “bagel” will occur 400 times in a million word corpus.

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	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

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.

when number of zeros are large then a lot of probability mass is shaved off (too much smoothing)

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

every context uses the same lambdas, some context might appear more times and we might want to give the higher order n-gram more weightage just for that bigram

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

- Lambdas conditional on context:

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

How to set the lambdas?

- Use a **held-out** corpus



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

$$\log P(w_1 \dots w_n \mid M(I_1 \dots I_k)) = \sum_i \log P_{M(I_1 \dots I_k)}(w_i \mid w_{i-1})$$

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

Smoothing for Web-scale N-grams

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

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

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

Advanced smoothing

- Good-Turing
- Kneser-Ney

Language Modeling

Advanced Smoothing

Reminder: Add-1 (Laplace) Smoothing

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

More general formulations: Add-k

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

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

Unigram prior smoothing

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

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

Advanced smoothing algorithms

- Intuition used by many smoothing algorithms
 - Good-Turing
 - Kneser-Ney
- 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 $p = 0/18 = 0$
- $P_{GT}^*(\text{unseen}) = N_1/N = 3/18$

- Seen once (trout)

- $c = 1$
- MLE $p = 1/18$
- $C^*(\text{trout}) = 2 * N_2/N_1$
 $= 2 * 1/3$
 $= 2/3$
- $P_{GT}^*(\text{trout}) = 2/3 / 18 = 1/27$

Resulting Good-Turing numbers

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

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

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:
Kneser-Ney Smoothing

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 $c^* = (c - .75)$

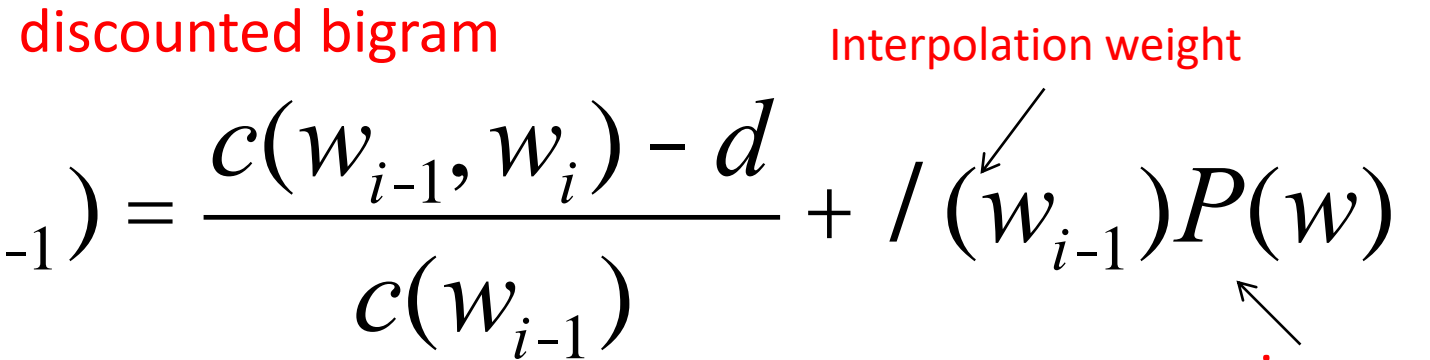
Count c	Good Turing c^*
0	.0000270
1	0.446
2	1.26
3	2.24
4	3.24
5	4.22
6	5.19
7	6.21
8	7.24
9	8.25

Absolute Discounting Interpolation

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

discounted bigram

Interpolation weight

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


- 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* glassesco?
 - “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 III

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

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

- normalized by the # of words preceding all words:

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

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

Kneser-Ney Smoothing IV

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

use continuation counts to distribute the probability mass from discounting

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

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

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