CS 333: Natural Language Processing

Fall 2025

Prof. Carolyn Anderson
Wellesley College

Reminders

- HW 3 will be released today
- My help hours: Monday 4-5:30
- Quiz 3 (Tuesday) will cover Appendix B
- Ashley's drop-in hour next week shifting to 5-6pm due to CS colloquium conflict.

WELLESLEY
COMPUTER SCIENCE

CS COLLOQUIUM

INNOVATING WITH UNDERSERVED COMMUNITIES: SUCCESSES, FAILURES, AND LESSONS LEARNED

Bill Thies

Everwell Technologies,

WEDNESDAY, SEPTEMBER 24
SCI-H105

4:00 PM – 5:00 PM

SNACKS WILL BE SERVED AT 3:45 PM

WELLESLEY

???: sb129
Accessibility and Disability: accessibility@wellesley.edu

New Policy

Earn an extra late day by attending a CS research talk!

To qualify, the talk must be **in-person** with the opportunity to **ask questions**.

Send me a **brief summary** of the talk and what you learned (1-2 paragraphs), and I will note down an extra late day.

Language Modeling

Language Modeling

Goal: compute the probability of a sentence or sequence of words:

$$P(W) = P(w_1,w_2,w_3,w_4,w_5 \ldots w_n)$$

Related task: probability of an upcoming word:

$$P(w_5 \mid w_1,w_2,w_3,w_4)$$

A model that computes either of these:

$$P(W) \quad \text{or} \quad P(w_n \mid w_1,w_2 \ldots w_{n-1})$$

is called a **language model**.

Chain Rule

The Chain Rule applied to compute the joint probability of words in sentence:

$$P(w_1, w_2 \ldots w_n) = \prod_n P(w_n \mid w_1 w_2 \ldots w_{n-1})$$

P("its water is so transparent") =

Markov Assumption

Approximate each component in the product:

$$P(w_1, w_2 \ldots w_n) = \prod_n P(w_n \mid w_1 w_2 \ldots w_{n-1})$$

$$P(w_1, w_2 \ldots w_n) \approx \prod_n P(w_n \mid w_{n-k} \ldots w_{n-1})$$

Estimating Bigram Probabilities

The Maximum Likelihood Estimate

$$P(w_i \mid w_{i-1}) = \frac{count(w_{i-1}, w_i)}{count(w_{i-1})}$$

Approximating Shakespeare

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

Evaluation and Perplexity

Evaluation: How Good is Our Model?

- Does our language model prefer good sentences to bad ones?
  - Assign higher probability to "real" or "frequently observed" sentences compared to "ungrammatical" or "rarely observed" sentences?
- We train parameters of our model on a **training set**.
- We test the model's 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.

Evaluation: How Good is Our Model?

- Does our language model prefer good sentences to bad ones?
  - Assign higher probability to “real” or “frequently observed” sentences compared to “ungrammatical” or “rarely observed” sentences?

Ethics alert!

Extrinsic Evaluation

- Best evaluation for comparing models A and B
  - Put each model in a task
    - spelling corrector, speech recognizer, MT system
  - Run the task, get an accuracy for A and for B
    - How many misspelled words corrected properly
    - How many words translated correctly
  - Compare accuracy for A and B

Intrinsic Evaluation

- Extrinsic evaluation
  - Time-consuming; can take days or weeks
- Instead, we can use **intrinsic** evaluation:

Perplexity:

a measure of probability distribution similarity

Perplexity

Perplexity is a bad approximation unless the test data looks **just** like the training data.

So it is generally only useful in pilot experiments or to compare models on the **same dataset**.

Perplexity: An Intuition

- The **Shannon Game**:
  - How well can we predict the next word?

    I always order pizza with cheese and ____

    The 33rd President of the US was ____

    I saw a ____

  - Unigrams are terrible at this game. (Why?)

mushrooms 0.1

pepperoni 0.1

anchovies 0.01

….

fried rice 0.0001

….

and 1e-100

The best model of a text is the one that assigns the highest probability to the word that **actually occurs.**

Perplexity

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

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

- Gives the highest P(sentence)

$$PP(W) = P(w_1 w_2 \ldots w_N)^{-\frac{1}{N}}$$

Minimizing perplexity is the same as maximizing probability!

Perplexity

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

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

$$PP(W) = P(w_1 w_2 \ldots w_N)^{-\frac{1}{N}}$$

$$= \sqrt[N]{\frac{1}{P(w_1 w_2 \ldots w_N)}}$$

Chain rule:

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

For bigrams:

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

Minimizing perplexity is the same as maximizing probability!

Perplexity As Branching Factor

- Take a language of strings that consist of random digits.
- What is the perplexity of a sentence according to a model that assign P=1/10 to each digit?

$$PP(W) = P(w_1 w_2 \ldots w_N)^{-\frac{1}{N}}$$
$$= \left(\frac{1}{10}\right)^{N\left(-\frac{1}{N}\right)}$$
$$= \frac{1}{10}^{-1}$$
$$= 10$$

Lower perplexity = better model

- Training 38 million words, test 1.5 million words, WSJ

N-gram Order	Unigram	Bigram	Trigram
Perplexity	962	170	109

p(cat | the) = 0.8

p(dog | the) = 0.2

C_1: the cat the dog

C_2: the cat the cat

Generalization and Zeroes

The Perils of Overfitting

N-grams only work well for word prediction if the test corpus looks like the training corpus. But 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

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)

Zero Probability Bigrams

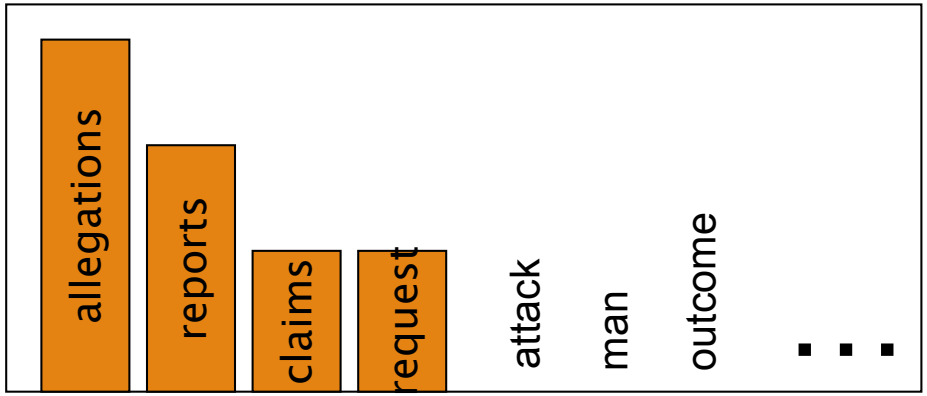
- Bigrams with zero probability mean that we will assign 0 probability to the test set!
- This means we can't compute perplexity (can't divide by 0)!

Smoothing

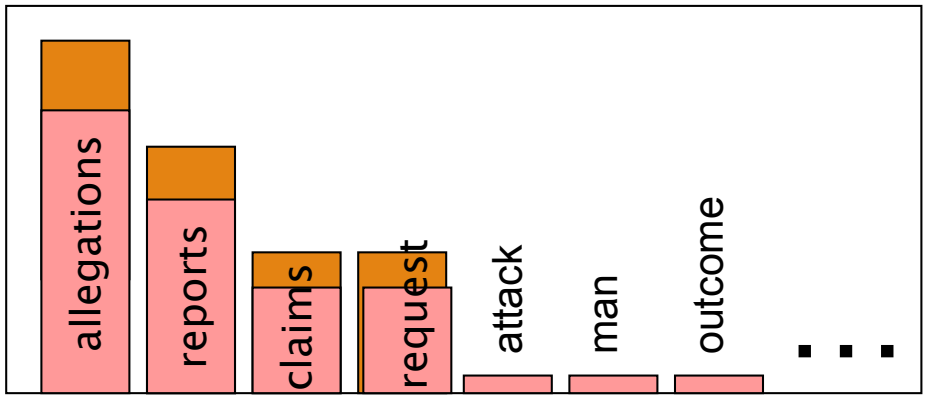
The Intuition of Smoothing

- When we have sparse statistics:

  P(w | denied the)
  - 3 allegations
  - 2 reports
  - 1 claims
  - 1 request
  - 7 total



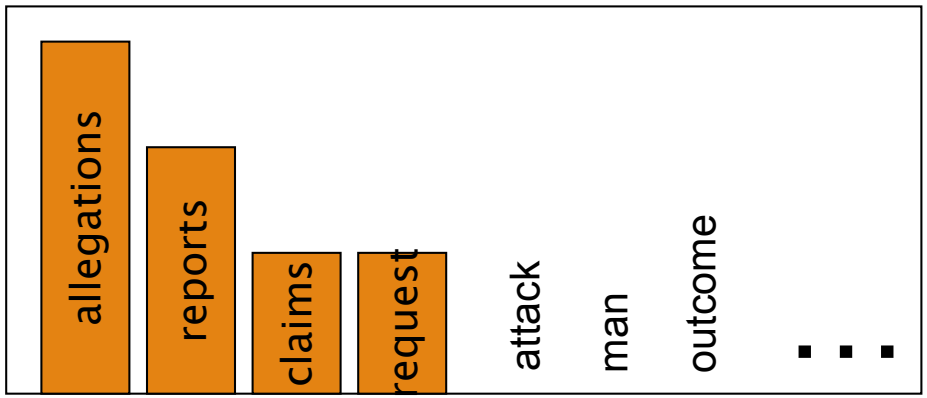
- Steal probability mass to generalize better



The Intuition of Smoothing

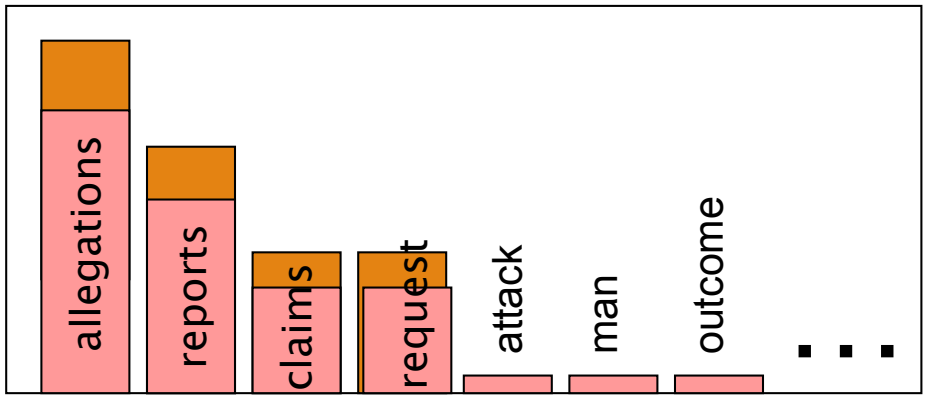
- When we have sparse statistics:

P(w | denied the)
3 allegations
2 reports
1 claims
1 request
7 total



- Steal probability mass to generalize better

P(w | denied the)
2.5 allegations
1.5 reports
0.5 claims
0.5 request
2 other
7 total



Add-One Smoothing

- Also called Laplace smoothing
- Key idea: pretend we saw each word one more time than we did. Just add one to all the counts!
- MLE estimate: $$P_{MLE}(w_i \mid w_{i-1}) = \frac{c(w_{i-1}, w_i)}{c(w_{i-1})}$$
- Add-1 estimate: $$P_{add1}(w_i \mid w_{i-1}) = \frac{c(w_{i-1}, w_i)+1}{c(w_{i-1})+V}$$

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:
- 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 counts

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

	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

Comparison 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 is a Naive Smoothing Approach

- Add-1 isn't commonly used for language models— there are 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.

Interpolation and Backoff

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 models

Interpolation works better

Linear Interpolation

- **Simple interpolation**: estimate the trigram probabilities by mixing unigram, bigram, and trigram probabilities.

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

$$\sum_i \lambda_i = 1$$

Interpolation

Simple interpolation: estimate the trigram probabilities by mixing unigram, bigram, and trigram probabilities.

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

$$\sum_i \lambda_i = 1$$

Let's set arbitrary weights: $\lambda_1 = 0.4$ $\lambda_2 = 0.6$

	i	want	to	eat	
i	5	827	0	9	1200
want	2	0	608	1	900
to	2	0	4	686	930
eat	0	0	2	0	39
	1200	900	930	39	

p(want|i) =

$0.4\, p(\text{want}) + 0.6(p(\text{want}|i))$

$= 0.4 \cdot 900 = 0.6(827)$

$= 856.2$

Interpolation

Simple interpolation: estimate the trigram probabilities by mixing unigram, bigram, and trigram probabilities.

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

$$\sum_i \lambda_i = 1$$

Let's set arbitrary weights: $\lambda_1 = 0.4$ $\lambda_2 = 0.6$

	i	want	to	eat	
i	5	827	0	9	1200
want	2	0	608	1	900
to	2	0	4	686	930
eat	0	0	2	0	39
	1200	900	930	39	

p(want|to) =

Linear Interpolation

- **Simple interpolation**: estimate the trigram probabilities by mixing unigram, bigram, and trigram probabilities.

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

$$\sum_i \lambda_i = 1$$

- Lambdas conditional on context:

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

How to set the lambdas?

- Use a **held-out** corpus

- 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:

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

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

Stupid Backoff

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

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

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

Smoothing

- Add-1 smoothing:
  - OK for text categorization, not for language modeling
- The most commonly used method:
  - Extended Interpolated Kneser-Ney (Intuition: instead of asking “How likely is w?”, ask “How likely is w to appear as a novel continuation?
- For very large N-grams like the Web:
  - Stupid backoff

Homework 3

- Will be released today
- Two parts:
  - Part 1: using n-gram counts to quantify differences between collections of text
  - Part 2: improving our in-class n-gram language model