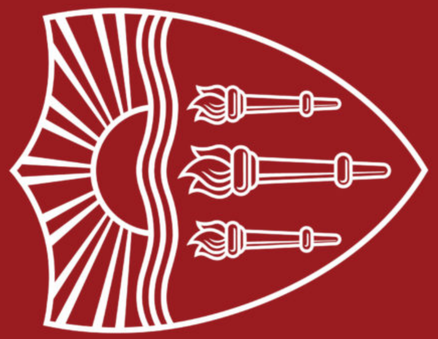


Lecture 2:
n-gram Language Models (contd.)

Instructor: Swabha Swayamdipta

USC CSCI 444 Natural Language Processing

Aug 27, 2025

Some slides adapted from Dan Jurafsky and Chris Manning

Announcements

+

Recap

Logistics and Announcements

- Upcoming:
  - Mon 9/1: Labor Day, no class. HW1 released
  - Mon 9/8: Project Pitches
    - Every student pitches a 5-minute project idea for which all the other students vote.
    - The pitch should outline the problem being solved and why should we care about it.
    - There should be a clear connection to language models
    - What the inputs and the outputs are, ideally with real-world examples
    - Name the project idea
    - See website for examples of projects from previous iterations of the class
  - Mon 9/8: Quiz 1 (Multiple choice questions on Brightspace; Bring your laptop!)
- Brightspace Discussions: Start a new thread under Activities / Discussions / Forums / Topics
  - Sign up for notifications
- Lecture Slides: Available right after class on website

Probabilistic Language Modeling

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

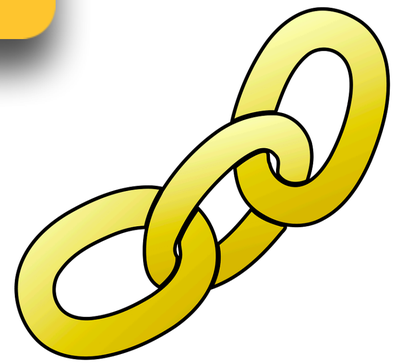
$$P(\mathbf{w}) = P(w_1, w_2, w_3, \ldots w_n)$$

Related task: probability of an upcoming word: $P(w_n \mid w_1, w_2, w_3, w_4, \ldots w_{n-1})$

A model that assigns probabilities to sequences of words is called a language model

Chain Rule

$$P(w_1, w_2, \ldots w_n) = \prod_{i=1}^{n} P(w_i \mid w_{i-1} \ldots w_1)$$



How to estimate the probability of the next word?

$$P(\text{that} \mid \text{its water is so transparent}) = \frac{Count(\text{its water is so transparent that})}{Count(\text{its water is so transparent})}$$

Maximum Likelihood Estimate

Too many possibilities to count! Too few sentences that look like this…

Need to make some simplifying assumptions…

Markov Assumption

$$P(w_i \mid w_1, w_2, \ldots w_{i-1}) \approx P(w_i \mid w_{i-k+1} \ldots w_{i-1})$$

k-th order Markov Assumption

In other words, we approximate each component in the product such that it is only conditioned on the previous $k-1$ elements

$$P(w_1, w_2, \ldots w_n) = \prod_i P(w_i \mid w_{i-k+1} \ldots w_{i-1})$$

n-gram models

Unigram Model

$$P(w_1, w_2, \ldots w_n) \approx \prod_i P(w_i)$$

Bigram Model

$$P(w_1, w_2, \ldots w_n) \approx \prod_i P(w_i \mid w_{i-1})$$

k-gram Model

$$P(w_1, w_2, \ldots w_n) \approx \prod_i P(w_i \mid w_{i-k+1} \ldots w_{i-1})$$

Definitely true for tokens in natural language!

n-gram Models: Limitations

In general this is an insufficient model of language

- "The *computer* which I had just put into the machine room on the fifth floor *crashed*."

At times the dependencies are not even clear!

- "The complex houses married and single soldiers and their families."
- "The horse raced past the barn fell."
- "The old man the boat."

Garden Path Sentences

But we can often get away with n-gram models

Language has long-distance dependencies

Estimating bigram probabilities

Maximum Likelihood Estimate

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

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

Counts are whole numbers

We do everything in log space to handle overflow issues

Special edge case tokens: <s> and </s> for the beginning of a sentence and the end of a sentence, respectively

For the 9222 sentences in the Berkeley Restaurant Corpus:

Unigram Counts

i	want	to	eat	chinese	food	lunch	spend
2533	927	2417	746	158	1093	341	278

Bigram Counts

Next Word

	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

History

w_i

Most n-grams are never seen!

Bigram Probabilities

	i	want	to	eat	chinese	food	lunch	spend
i	0.002	0.33	0	0.0036	0	0	0	0.00079
want	0.0022	0	0.66	0.0011	0.0065	0.0065	0.0054	0.0011
to	0.00083	0	0.0017	0.28	0.00083	0	0.0025	0.087
eat	0	0	0.0027	0	0.021	0.0027	0.056	0
chinese	0.0063	0	0	0	0	0.52	0.0063	0
food	0.014	0	0.014	0	0.00092	0.0037	0	0
lunch	0.0059	0	0	0	0	0.0029	0	0
spend	0.0036	0	0.0036	0	0	0	0	0

w_{i-1}

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

Lecture Outline

1. Announcements + Recap
2. Evaluation of Language Models and Perplexity
3. Generating from an n-gram Language Model
    i. Zeroes
4. Smoothing

Evaluation of Language Models: Perplexity

How good is a language model?

Does our language model prefer good sentences to bad ones?

- Key Idea: Assign higher probability to “real” or “frequently observed” sentences than “ungrammatical” or “rarely observed” sentences?
  - In practice we don’t explicitly need to do the latter!

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.

Extrinsic evaluation of n-gram models

Best evaluation for comparing models A and B

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

Downsides??

Text Generation: Intrinsic or Extrinsic Evaluation?

Machine Learning 101

Train Set vs Test Set:

- We can't allow test sentences into the training set
- We will assign it an artificially high probability when we set it in the test set
- "Training on the test set" is bad science! And violates the honor code

Another risk of cheating:

- using a particular test set so often that we implicitly tune to its characteristics.
- So how to evaluate while developing a model? Use a fresh test set that is truly unseen: **development set**!

In practice, we often just divide our data into 80% training, 10% development, and 10% test.

How best to evaluate an LM?

- Extrinsic evaluation can be time-consuming; hard to design
  - Which is the best task? How many tasks to try?
- Therefore, we often use intrinsic evaluation:
  - Bad approximation
    - unless the test data looks just like the training data
  - Generally only useful in pilot experiments

Perplexity

Intuition of Perplexity

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 _____

- mushrooms 0.1
- pepperoni 0.1
- anchovies 0.01
- ….
- fried rice 0.0001
- ….
- and 1e-100

Unigrams are terrible at this game!

A better model of a text is one which assigns a higher probability to the word that actually occurs

Perplexity

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

- Gives the highest P(sentence), for most sentences acceptable to humans

$$PPL(\mathbf{w}) = P(w_1 w_2 \ldots w_N)^{-\frac{1}{N}}$$

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

$$PPL(\mathbf{w}) = P(w_1 w_2 \ldots w_N)^{-\frac{1}{N}}$$

Minimizing perplexity is the same as maximizing probability

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

$$= \sqrt[N]{\frac{1}{\prod_i P(w_i \mid w_1 \ldots w_{i-1})}}$$

Chain rule

$$= \sqrt[N]{\frac{1}{\prod_i P(w_i \mid w_{i-1})}}$$

Applying Markov's assumption for bigrams

Perplexity Example

Let's suppose a sentence of length 50 consisting of random digits

$$P(w) = \frac{1}{10}$$

What is the perplexity of this sentence according to a model that assigns uniform probability to each digit?

$$PPL(\mathbf{w}) = P(w_1 w_2 \ldots w_N)^{\frac{-1}{N}}$$

$$= \left(\frac{1}{10}^{50}\right)^{-\frac{1}{50}}$$

$$= 10$$

Lower perplexity = better model!

Training 38 million words, test 1.5 million words, from the Wall Street Journal

N-gram Order	Unigram	Bigram	Trigram
Perplexity	962	170	109

What are the two things that might affect perplexity?

Lecture Outline

1. Announcements + Recap
2. Evaluation and Perplexity
3. Generating from an n-gram Language Model
    i. Zeroes
4. Smoothing

Generating from an n-gram model and Zeros

Recall: BRP bigram probabilities

P(english|want) = .0011

P(chinese|want) = .0065

P(to|want) = .66

P(eat | to) = .28

P(food | to) = 0

P(want | spend) = 0

P (i | <s>) = .25

How can we generate sentences from this bigram model?

Generating from a unigram model

- Pick <s>
- While 1:
  - **Randomly** sample token w from $P(w)$
    - Construct cumulative distribution function
    - Randomly sample value q between 0-1
    - Pick w such that $P(w) \approx q$
  - If w ==</s> break

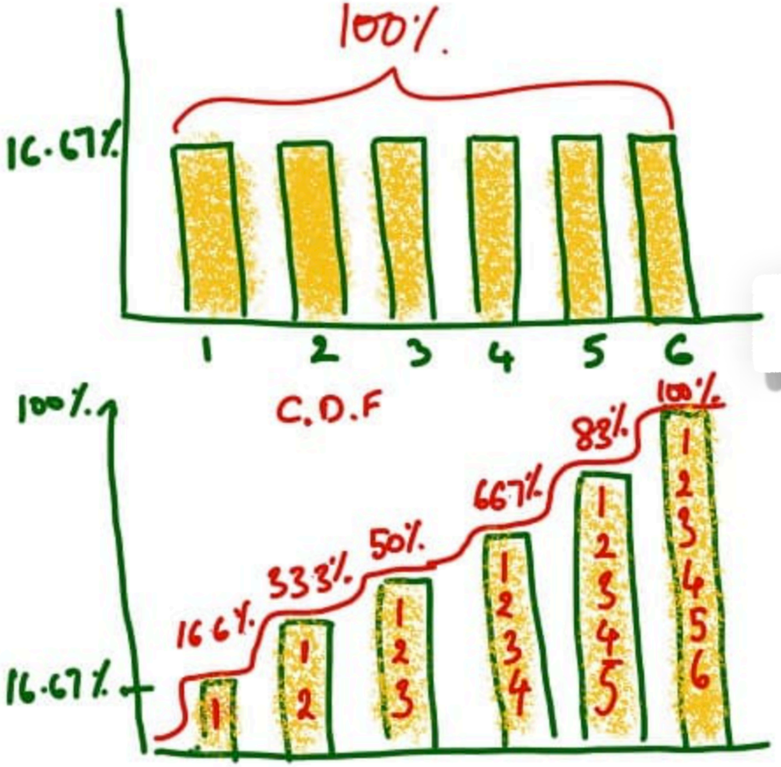


Image credit: GraduateTutor

Generating from a bigram model

- Choose a random bigram (<s>, w) according to its probability
- Now choose a random bigram (w, x) according to its probability
- And so on until we choose </s>
- 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

The WSJ is no Shakespeare!

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

Shakespearean *n*-grams

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.

Shakespeare as a corpus

The corpus contains $N = 884{,}647$ tokens, with vocabulary $V = 29{,}066$

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)

4-grams (quadrigrams) are rarer still…

What's coming out looks like Shakespeare because it is Shakespeare!

Most n-grams are never seen!

So why not just sample from very high order n-gram models? Do we even need GPT-style LLMs?

The successes we are seeing here is a phenomena commonly known as overfitting

Overfitting bad!

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**!
  - Technical terms for "doing well on the test data" or "doing well on any test data"
- One kind of generalization: Zeros!
  - Data that don't ever occur in the training set, but occur in the test set

Zeros

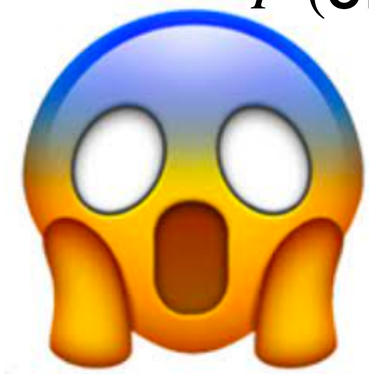
Training set:

… denied the allegations
… denied the reports
… denied the claims
… denied the request

Test set

… denied the offer
… denied the loan

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



will assign 0 probability to the test set!

What happens to perplexity??

One solution: the UNK token

A token is a technical term in NLP for what is commonly referred to as a word

Problem: Word "offer" didn't appear in the train set…many words like "Swayamdipta" won't appear in most training sets!

These are known as **OOV** for "out of vocabulary", or **unknown tokens**

One way to handle OOV tokens is by adding a pseudo-word called <UNK>

We can replace all words that occur fewer than n times in the training set—where n is some small number—by <UNK> and re-estimate the counts and probabilities

When not done carefully, may artificially lower perplexity

Lecture Outline

1. Announcements + Recap
2. Evaluation and Perplexity
3. Generating from an n-gram Language Model
    i. Zeroes
4. Smoothing
    i. Add-one / Laplace
    ii. Interpolation

Intuition for Smoothing

I like to **eat** cake but I want to **eat** pizza right now. Mary told her brother to **eat** pizza too.

→

P(next word = *pizza* | previous word = *eat*) = 2/3

P(next word = *cake* | previous word = *eat*) = 1/3

All other next words = 0 probability

- Types: I, like, to, eat, cake, but, want, pizza, right, now, ., Mary, told, her, brother, too
  - $|V| = ?$ $|V_{\text{bigrams}}| = ?$
- All other vocabulary tokens getting 0 probability just doesn't seem right. We want to assign some probability to other words
- We want to **smooth the distribution from our counts**

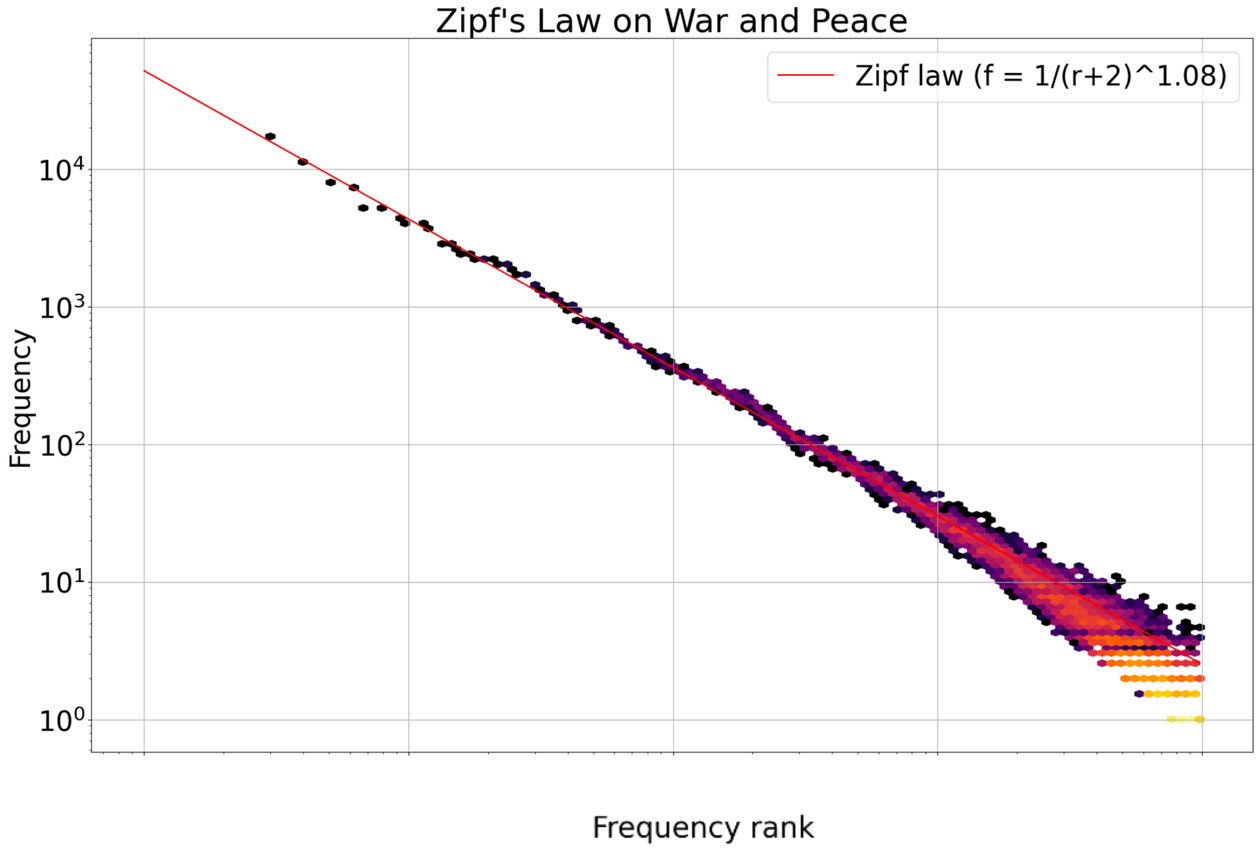
What does a count distribution look like?

Zipf's Law

The distribution over words resembles that of a power law:

- there will be a few words that are very frequent, and a long tail of words that are rare
- $freq_w(r) \approx r^{-s}$, where s is a constant

NLP algorithms must be especially robust to observations that do not occur or rarely occur in the training data

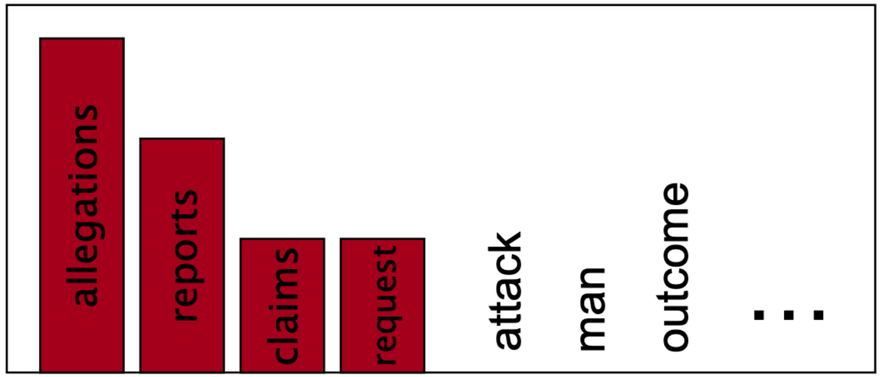


Zipf, G. K. (1949). Human behavior and the principle of least effort.

Smoothing ~ Massaging Probability Masses

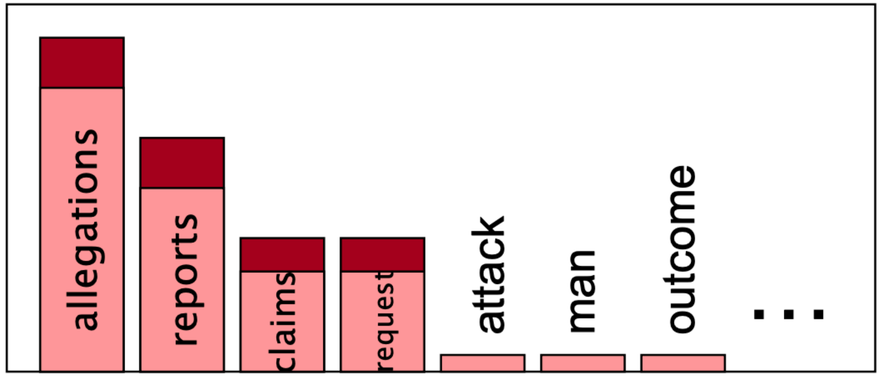
When we have sparse statistics: $Count(w \mid \text{denied the})$

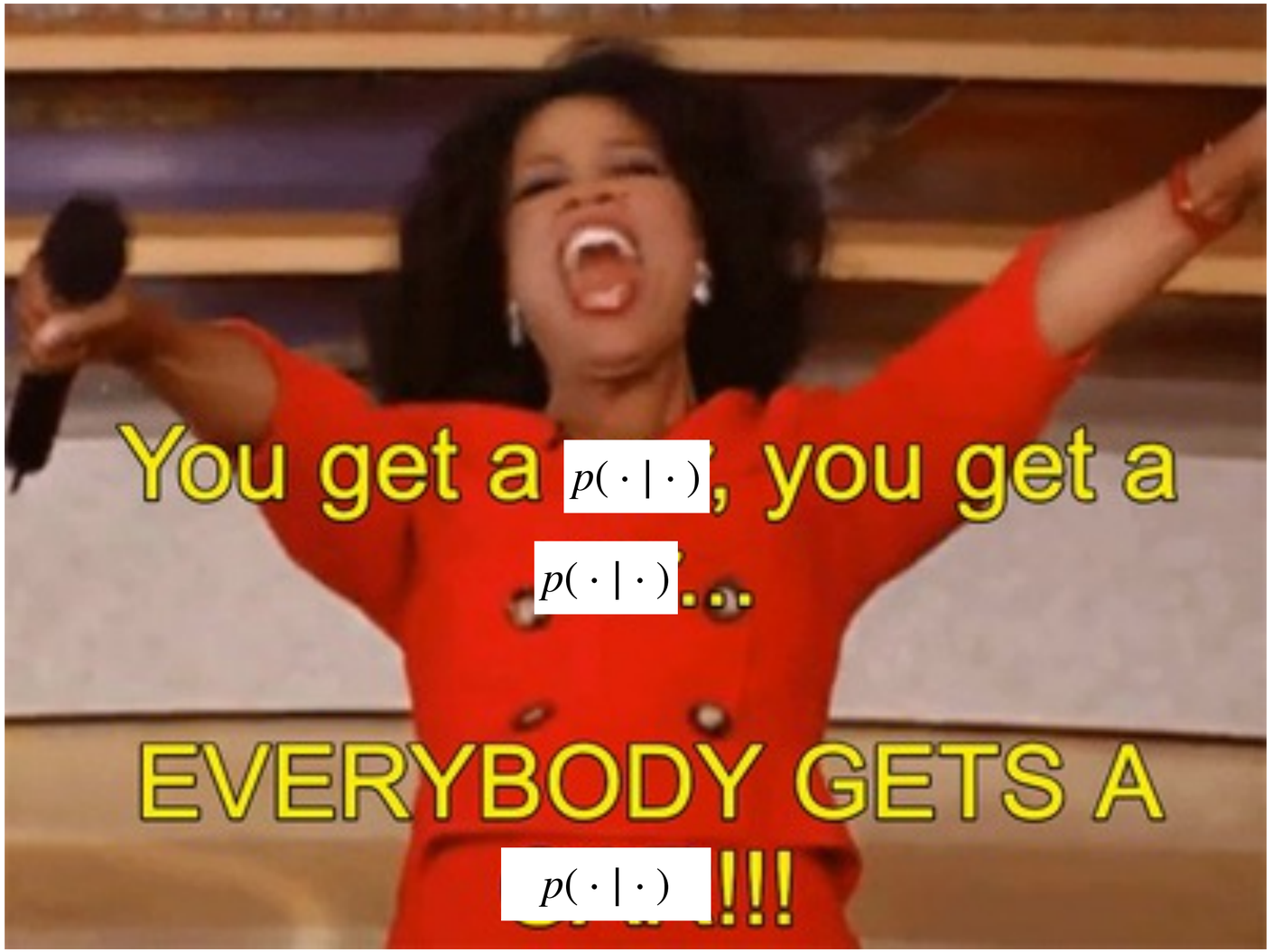
3 allegations
2 reports
1 claims
1 request
7 total



Steal probability mass to generalize better: $Count(w \mid \text{denied the})$

2.5 allegations
1.5 reports
0.5 claims
0.5 request
2 other
7 total




You get a $p(\cdot|\cdot)$, you get a
$p(\cdot|\cdot)$...
EVERYBODY GETS A
$p(\cdot|\cdot)$!!!

Smoothing

Add-One Estimation

MLE estimate

$$P_{MLE}(w_i) = \frac{c(w_i)}{\sum_w c(w)}$$

Laplace smoothing

75 year old method!

1. Pretend we saw each word one more time than we did
2. Just add one to all the counts!
3. All the counts that used to be zero will now have a count of 1…

Add-1 estimate

$$P_{Add-1}(w_i) = \frac{c(w_i)+1}{\sum_w (c(w)+1)} = \frac{c(w_i)+1}{V+\sum_w c(w)}$$

What happens to our P if we don't increase the denominator?

Add-1 Estimation Bigrams

MLE estimate

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

Pretend we saw each **bigram** one more time than we did

Add-1 estimate

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

What does this do to the unigram counts?

Keep the same denominator as before and reconstruct bigram counts

$$= \frac{c^*(w_{i-1}w_i)}{c(w_{i-1})}$$

Recall: BRP Corpus

- can you tell me about any good cantonese restaurants close by
- mid priced thai food is what i'm looking for
- tell me about chez panisse
- can you give me a listing of the kinds of food that are available
- i'm looking for a good place to eat breakfast
- when is caffe venezia open during the day

Unigrams

i	want	to	eat	chinese	food	lunch	spend
2533	927	2417	746	158	1093	341	278

Bigrams

w_i

w_{i-1}	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

Laplace-smoothed bigram counts

Just add one to all the counts!

w_i

w_{i-1}	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 bigram probabilities

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

	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

$$c*(w_{i-1}w_i) = \frac{[c(w_{i-1}w_i)+1]c(w_{i-1})}{c(w_{i-1})+V}$$

	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

Original, Raw

	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

Reconstructed

	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

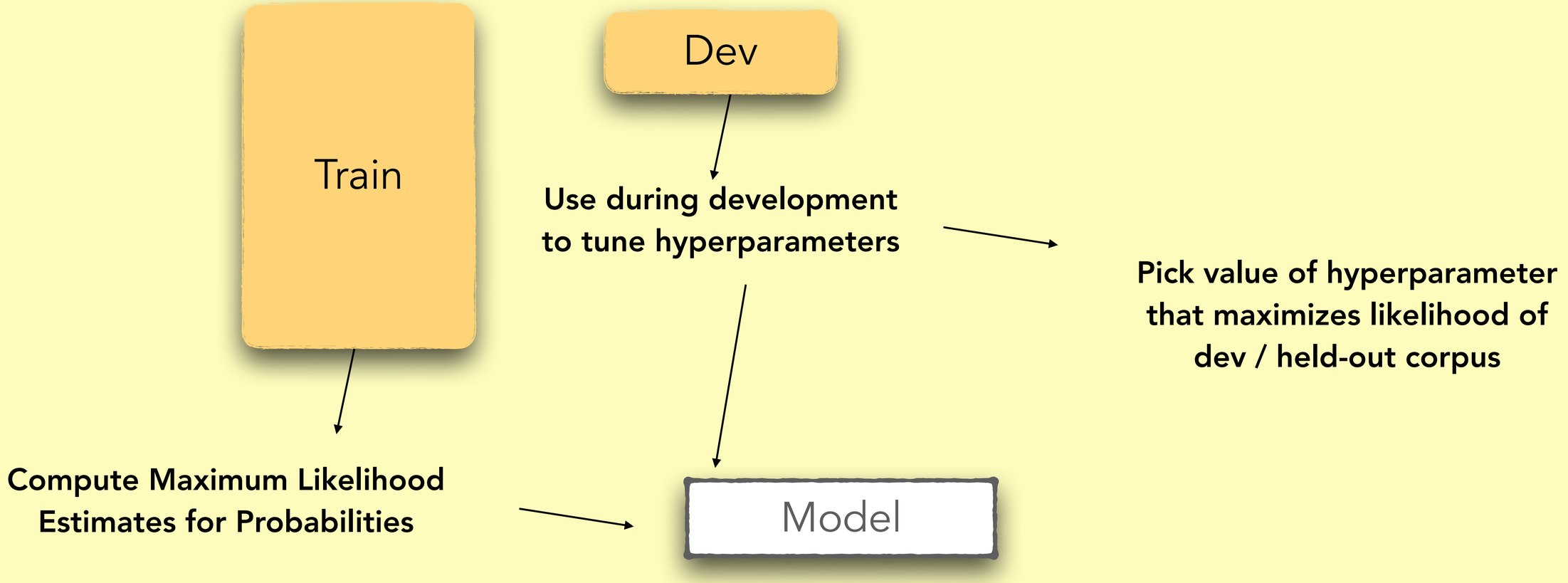
Big change to the counts!

Perhaps 1 is too much, add a fraction?

Add-k smoothing

k is a hyperparameter

Language Model Development



Add-1 Estimation: Last thoughts

So add-1 isn't used for n-grams, being something of a blunt instrument

- One-size-fits-all

Add-1 is used to smooth other NLP models though…

- For text classification (Naïve Bayes)
- In domains where the number of zeros isn't so huge

Next Lecture (After Labor Day)

- Interpolation Smoothing for n-gram models.
- Logistic Regression
- TODOs for you: Start thinking of project pitches, and look out for HW1 on Brightspace