SFU NatLangLab

CMPT 413/825: Natural Language Processing

Language Models

Fall 2020
2020-09-11

Adapted from slides from Anoop Sarkar, Danqi Chen and Karthik Narasimhan

Announcements

- **Sign up on Piazza for announcements, discussion, and course materials:**

  piazza.com/sfu.ca/fall2020/cmpt413825

- Homework 0 is out — due 9/16, 11:59pm
  - Review problems on probability, linear algebra, and calculus
  - Programming - Setup group, github, and starter problem
    - Try to have unique group name
    - Make sure your Coursys group name and your GitHub repo name match
    - Avoid strange characters in your group name
- Interactive Tutorial Session
  - 11:50am to 12:20pm - last 30 minutes of lecture
  - (optional) but recommended review of math background

Consider

Today, in Vancouver, it is 76 F and <u>red</u>

vs

Today, in Vancouver, it is 76 F and <u>sunny</u>

- Both are grammatical
- But which is more likely?

Language Modeling

- We want to be able to estimate the probability of a sequence of words
    - How likely is a given phrase / sentence / paragraph / document?

Why is this useful?

Applications

- Predicting words is important in many situations
  - Machine translation

$$P(\text{a } \textbf{smooth} \text{ finish}) > P(\text{a } \textbf{flat} \text{ finish})$$

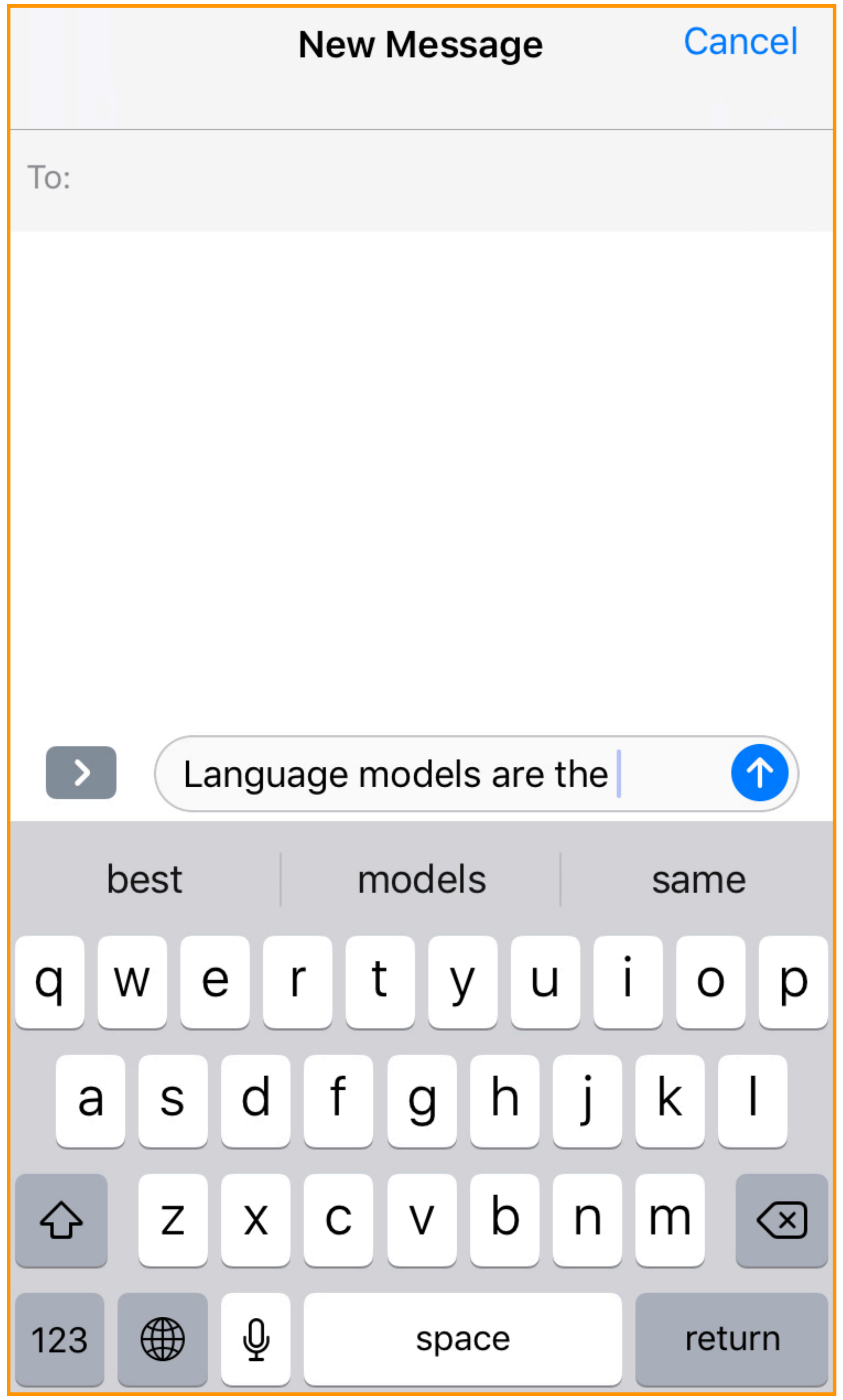
  - Speech recognition/Spell checking

$$P(\text{high school } \textbf{principal}) > P(\text{high school } \textbf{principle})$$

  - Information extraction, Question answering

Language models are everywhere

Autocomplete



Impact on downstream applications

Language Resources	Adaptation	Word		PP
		Cor.	Acc.	
1. Doc-A		54.5%	45.1%	49972
2. Trans-C(L)		63.3%	50.6%	1856.5
3. Trans-B(L)		70.2%	60.3%	318.4
4. Trans-A(S)		70.4%	59.3%	442.3
5. Trans-B(L)+Trans-A(S)	CM	72.6%	63.9%	225.1
6. Trans-B(L)+Doc-A	KW	72.1%	64.2%	247.5
7. Trans-B(L)+Doc-A	KP	73.1%	65.6%	259.7
8. Trans-A(L)		75.2%	67.3%	148.6

(Miki et al., 2006)

New Approach to Language Modeling Reduces Speech Recognition Errors by Up to 15%

December 13, 2018
Ankur Gandhe

Alexa Alexa research Alexa science

What is a language model?

Probabilistic model of a sequence of words

Setup: Assume a finite vocabulary of words V

$$V = \{\text{killer, crazy, clown}\}$$

V can be used to construct a infinite set of sentences (sequences of words)

$$V^+ = \{\text{clown, killer clown, crazy clown,}$$
$$\text{crazy killer clown, killer crazy clown, } \ldots\}$$

where a sentence is defined as $s \in V^+$ where $s = \{w_1, \ldots, w_n\}$

What is a language model?

Probabilistic model of a sequence of words

Given a training data set of example sentences

$S = \{s_1, s_2, \ldots, s_N\}, s_i \in V^+$

Estimate a probability model

$$\sum_{s_i \in V^+} p(s_i) = \sum_i p(w_1, \ldots, w_{n_i}) = 1.0$$

Language Model

- p(clown) = 1e-5
- p(killer) = 1e-6
- p(killer clown) = 1e-12
- p(crazy killer clown) = 1e-21
- p(crazy killer clown killer) = 1e-110
- p(crazy clown killer killer) = 1e-127

Learning language models

How to estimate the probability of a sentence?

- We can directly count using a training data set of sentences
  - $P(w_1, \ldots, w_n) = \dfrac{c(w_1, \ldots, w_n)}{N}$
  - c is a function that counts how many times each sentence occurs
  - N is the sum over all possible $c(\cdot)$ values

Learning language models

How to estimate the probability of a sentence?

$$P(w_1, \ldots, w_n) = \frac{c(w_1, \ldots, w_n)}{N}$$

- **Problem**: does not generalize to new sentences unseen in the training data
  - What are the chances you will see a sentence

    crazy killer clown crazy killer

  - In NLP applications, we often need to assign non-zero probability to previously unseen sentences

Estimating joint probabilities with the chain rule

$$p(w_1, w_2, \ldots, w_n) = p(w_1)p(w_2 \mid w_1)p(w_3 \mid w_1, w_2) \times \ldots \times p(w_n \mid w_1, w_2, \ldots, w_{n-1})$$

Example

Sentence: “the cat sat on the mat”

$$\begin{aligned} P(\text{the cat sat on the mat}) = P(\text{the}) * P(\text{cat}|\text{the}) * P(\text{sat}|\text{the cat}) \\ *P(\text{on}|\text{the cat sat}) * P(\text{the}|\text{the cat sat on}) \\ *P(\text{mat}|\text{the cat sat on the}) \end{aligned}$$

Estimating probabilities

Let's count again!

$$P(\text{sat}|\text{the cat}) = \frac{\text{count(the cat sat)}}{\text{count(the cat)}}$$

$$P(\text{on}|\text{the cat sat}) = \frac{\text{count(the cat sat on)}}{\text{count(the cat sat)}}$$

$$\vdots$$

Maximum likelihood estimate (MLE)

- With a vocabulary of size $|V|$
  - # sequences of length n: $|V|^n$
  - Typical vocabulary ~ 50k words
  - even sentences of length ≤ 11 results in $\approx 4.9 \times 10^{51}$ sequences! *(# of atoms in the earth* $\approx 10^{50}$*)*

Markov assumption

- Use only the recent past to predict the next word
- Reduces the number of estimated parameters in exchange for modeling capacity
- 1st order

$$P(\text{mat}|\text{the cat sat on the}) \approx P(\text{mat}|\text{the})$$

- 2nd order

$$P(\text{mat}|\text{the cat sat on the}) \approx P(\text{mat}|\text{on the})$$

k^{th} order Markov

- Consider only the last *k* words for context

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

which implies the probability of a sequence is:

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

(k+1) gram

n-gram models

Unigram $$P(w_1, w_2, ...w_n) = \prod_{i=1}^{n} P(w_i)$$

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

and Trigram, 4-gram, and so on.

Larger the n, more accurate and better the language model (but also higher costs)

Caveat: Assuming infinite data!

Unigram Model

Apply the Chain Rule: the unigram model

$$\begin{aligned} p(w_1, \ldots, w_n) &\approx p(w_1)p(w_2)\ldots p(w_n) \\ &= \prod_i p(w_i) \end{aligned}$$

Big problem with a unigram language model

$p(\text{the the the the the the the}) > p(\text{we must also discuss a vision .})$

Bigram Model

Apply the Chain Rule: the bigram model

$$
\begin{aligned}
p(w_1, \ldots, w_n) &\approx p(w_1)p(w_2 \mid w_1) \ldots p(w_n \mid w_{n-1}) \\
&= p(w_1) \prod_{i=2}^{n} p(w_i \mid w_{i-1})
\end{aligned}
$$

Better than unigram

$$p(\text{the the the the the the the}) < p(\text{we must also discuss a vision .})$$

Trigram Model

Apply the Chain Rule: the trigram model

$$p(w_1, \ldots, w_n) \approx$$
$$p(w_1)p(w_2 \mid w_1)p(w_3 \mid w_1, w_2) \ldots p(w_n \mid w_{n-2}, w_{n-1})$$
$$p(w_1)p(w_2 \mid w_1)\prod_{i=3}^{n} p(w_i \mid w_{i-2}, w_{i-1})$$

Better than bigram, but ...

p(we must also discuss a vision .) might be zero because we have not seen p(discuss | must also)

Maximum Likelihood Estimate

Using training data to learn a trigram model

- Let $c(u, v, w)$ be the count of the trigram u, v, w, e.g. $c(crazy, killer, clown)$. $P(u, v, w) = \frac{c(u,v,w)}{\sum_{u,v,w} c(u,v,w)}$
- Let $c(u, v)$ be the count of the bigram u, v, e.g. $c(crazy, killer)$. $P(u, v) = \frac{c(u,v)}{\sum_{u,v} c(u,v)}$
- For any u, v, w we can compute the conditional probability of generating w given u, v:
$$p(w \mid u, v) = \frac{c(u, v, w)}{c(u, v)}$$
- For example:
$$p(clown \mid crazy, killer) = \frac{c(crazy, killer, clown)}{c(crazy, killer)}$$

Number of Parameters

How many probabilities in each n-gram model

- Assume $\mathcal{V} = \{killer, crazy, clown, UNK\}$

Question

How many unigram probabilities: $P(x)$ for $x \in \mathcal{V}$?

4

Number of Parameters

How many probabilities in each n-gram model

- Assume $\mathcal{V} = \{killer, crazy, clown, UNK\}$

Question

How many bigram probabilities: $P(y|x)$ for $x, y \in \mathcal{V}$?

$4^2 = 16$

Number of Parameters

How many probabilities in each n-gram model

- Assume $\mathcal{V} = \{killer, crazy, clown, UNK\}$

Question

How many trigram probabilities: $P(z|x, y)$ for $x, y, z \in \mathcal{V}$?

$4^3 = 64$

Number of parameters

- Assume $|\mathcal{V}| = 50{,}000$ (a realistic vocabulary size for English)
- What is the minimum size of training data in tokens?
  - If you wanted to observe all unigrams at least once.
  - If you wanted to observe all trigrams at least once.

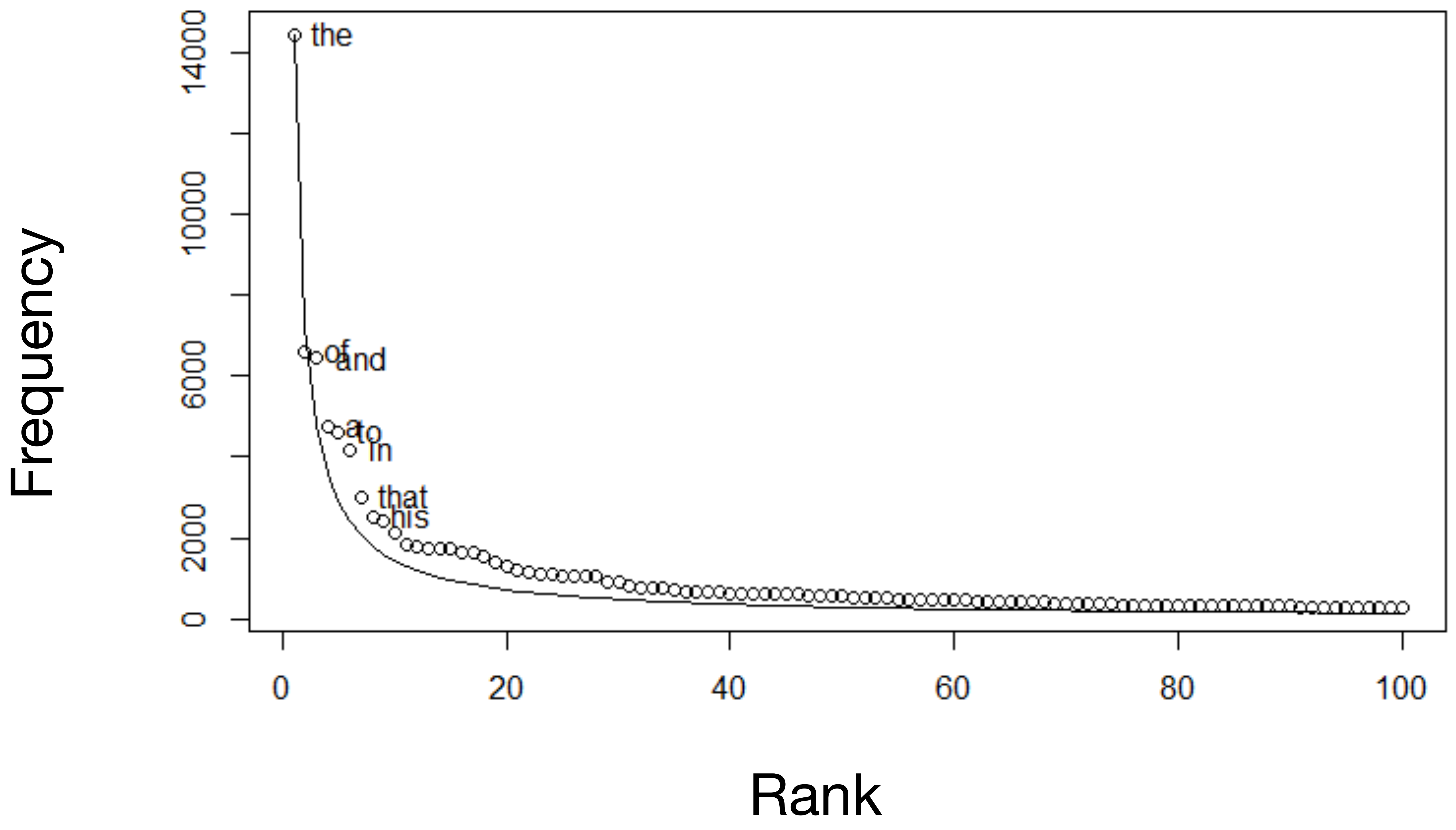
125,000,000,000,000 (125 Ttokens)

Some trigrams should be zero since they do not occur in the language, $P(\mathit{the} \mid \mathit{the}, \mathit{the})$.
But others are simply unobserved in the training data, $P(\mathit{idea} \mid \mathit{colourless}, \mathit{green})$.

Generalization of n-grams

- Not all n-grams will be observed in training data!
- Test corpus might have some that have zero probability under our model
  - Training set: *Google news*
  - Test set: *Shakespeare*
  - P (affray | voice doth us) = 0 ➡ P(test corpus) = 0

Sparsity in language



$$freq \propto \frac{1}{rank}$$

Zipf's Law

- Long tail of infrequent words
- Most finite-size corpora will have this problem.

Smoothing n-gram Models

Handling unknown words

Assume closed vocabulary

In some situations we can make this assumption, e.g. our vocabulary is ASCII characters

Interpolate with unknown words distribution

We will call this *smoothing*. We combine the n-gram probability with a distribution over unknown words

$$P_{\mathrm{unk}}(w) = \frac{1}{V_{\mathrm{all}}}$$

V_{all} is an estimate of the vocabulary size including unknown words.

Add an `<unk>` word

Modify the training data L by changing words that appear only once to the `<unk>` token. Since this probability can be an over-estimate we multiply it with a probability $P_{\mathrm{unk}}(\cdot)$.

Smoothing

- **Smoothing** deals with events that have been observed zero or very few times
- Handle sparsity by making sure all probabilities are non-zero in our model
    - Additive: Add a small amount to all probabilities
    - Interpolation: Use a combination of different n-grams
    - Discounting: Redistribute probability mass from observed n-grams to unobserved ones
    - Back-off: Use lower order n-grams if higher ones are too sparse

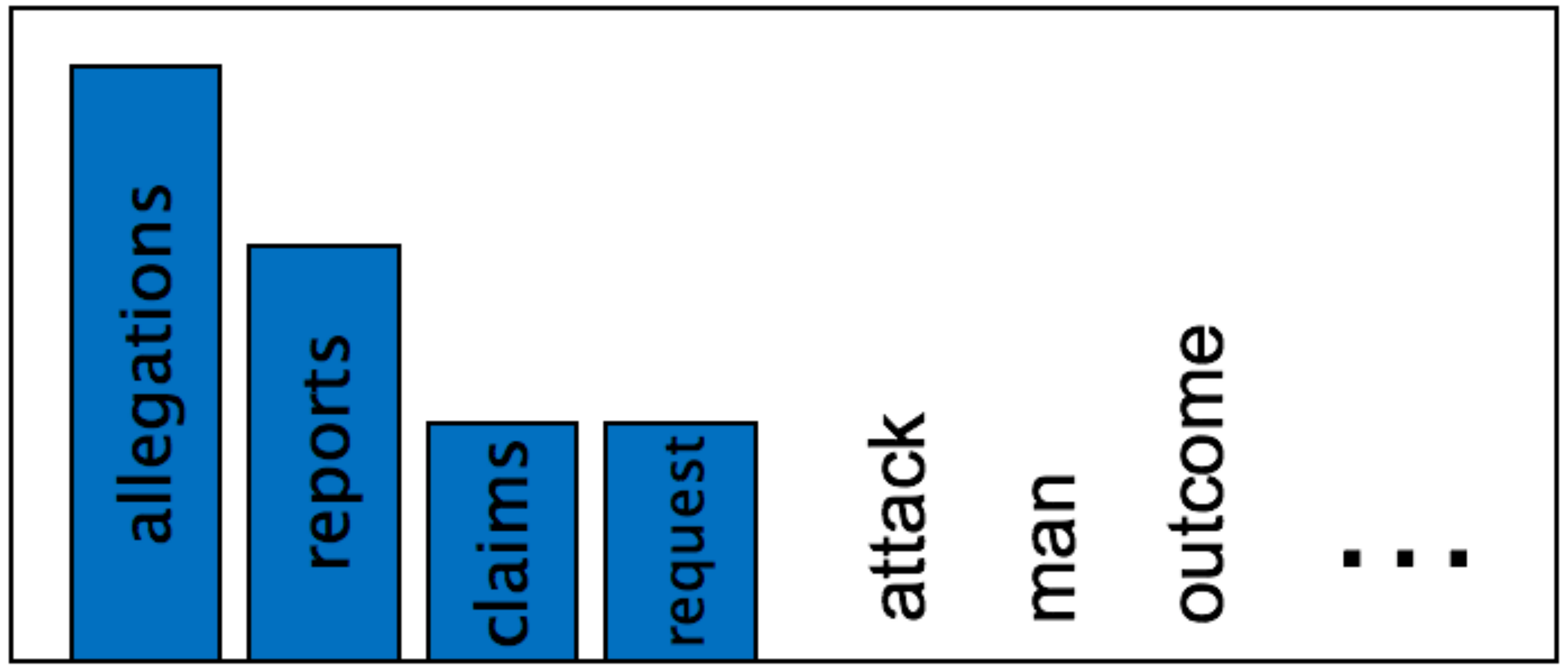
Smoothing intuition

Taking from the rich and giving to the poor

When we have sparse statistics:

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

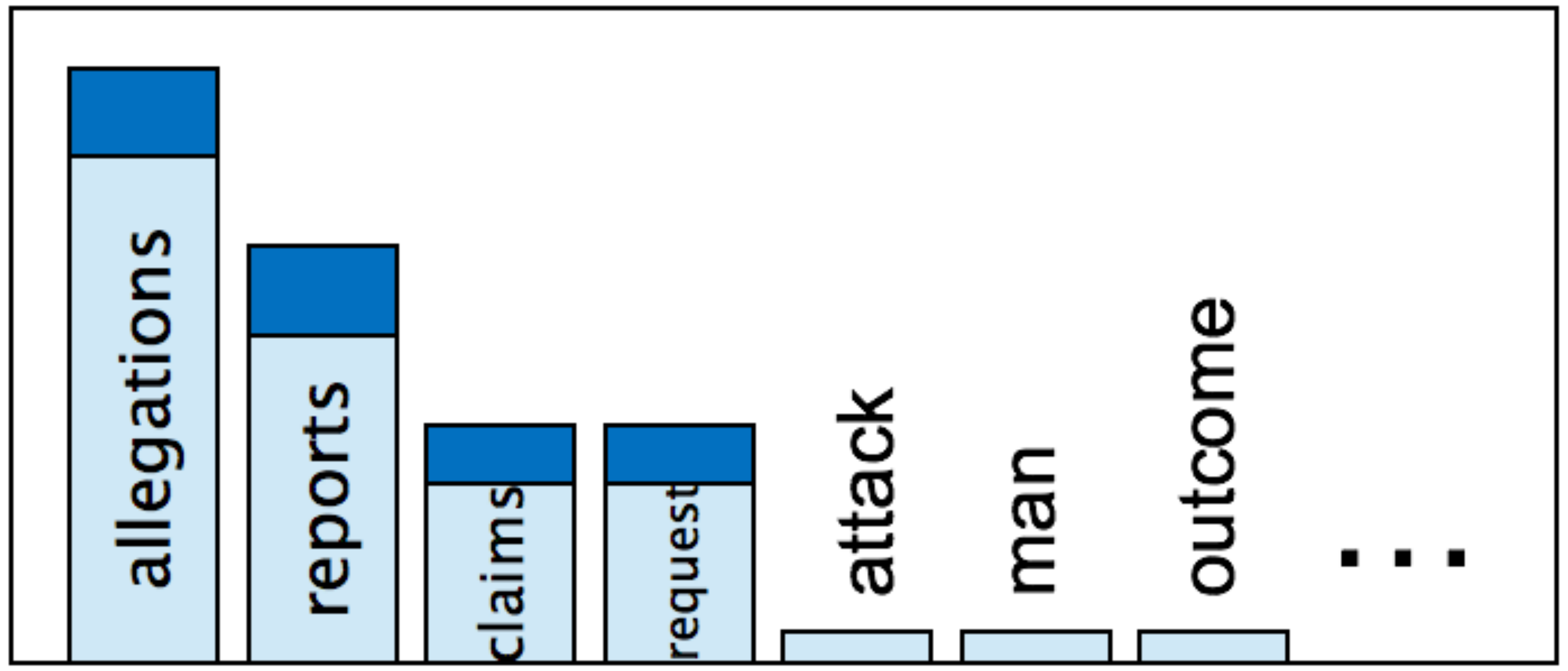
7 total



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



(Credits: Dan Klein)

Add-one (Laplace) smoothing

- Simplest form of smoothing: Just add 1 to all counts and renormalize!

- Max likelihood estimate for bigrams:

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

- Let $|V|$ be the number of words in our vocabulary. Assign count of 1 to unseen bigrams

- After smoothing:

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

Add-one (Laplace) smoothing

$$P(\texttt{insane killer clown}) = \\ P(\texttt{insane} \mid \texttt{<s>}) \times P(\texttt{killer} \mid \texttt{insane}) \times \\ P(\texttt{clown} \mid \texttt{killer}) \times P(\texttt{</s>} \mid \texttt{clown})$$

- Without smoothing:

$$P(\texttt{killer} \mid \texttt{insane}) = \frac{c(\texttt{insane, killer})}{c(\texttt{insane})} = 0$$

- With add-one smoothing (assuming initially that c(insane) = 1 and c(insane, killer) = 0):

$$P(\texttt{killer} \mid \texttt{insane}) = \frac{1}{|V| + 1}$$

Additive smoothing

(Lidstone 1920, Jeffreys 1948)

- Why add 1? 1 is an overestimate for unobserved events

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

- Additive smoothing ($0 < \delta \leq 1$):

$$P(w_i \mid w_{i-1}) = \frac{\delta + c(w_{i-1}, w_i)}{(\delta \times |V|) + c(w_{i-1})}$$

- Also known as add-alpha (the symbol α is used instead of δ)

Linear Interpolation (Jelinek-Mercer Smoothing)

$$\begin{aligned}\hat{P}(w_i|w_{i-1}, w_{i-2}) = \lambda_1 P(w_i|w_{i-1}, w_{i-2}) \\ +\lambda_2 P(w_i|w_{i-1}) \\ +\lambda_3 P(w_i)\end{aligned}$$

$$\sum_i \lambda_i = 1$$

- Use a combination of models to estimate probability
- Strong empirical performance

Linear Interpolation (Jelinek-Mercer Smoothing)

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

- $P_{JM}(w_i \mid w_{i-1}) = \lambda P_{ML}(w_i \mid w_{i-1}) + (1-\lambda) P_{ML}(w_i)$
  where, $0 \leq \lambda \leq 1$
- Jelinek and Mercer (1980) describe an elegant form of this **interpolation**:

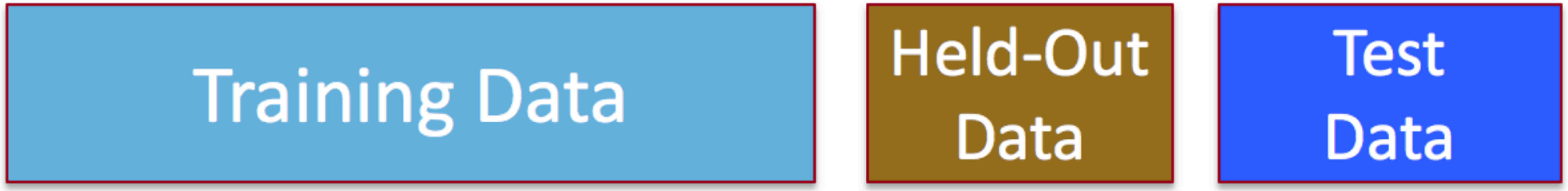
$$P_{JM}(ngram) = \lambda P_{ML}(ngram) + (1-\lambda) P_{JM}(n-1gram)$$

- What about $P_{JM}(w_i)$?
  For missing unigrams: $P_{JM}(w_i) = \lambda P_{ML}(w_i) + (1-\lambda)\frac{\delta}{|V|}$
  $0 < \delta \leq 1$

Linear Interpolation: Finding lambda

$$P_{JM}(ngram) = \lambda P_{ML}(ngram) + (1 - \lambda) P_{JM}(n-1gram)$$

- Deleted Interpolation (Jelinek, Mercer)
  compute λ values to minimize cross-entropy on **held-out** data
  which is deleted from the initial set of training data



- Improved JM smoothing, a separate λ for each w_{i-1}:

$$P_{JM}(w_i \mid w_{i-1}) = \lambda(w_{i-1}) P_{ML}(w_i \mid w_{i-1}) + (1 - \lambda(w_{i-1})) P_{ML}(w_i)$$

Next Week

- More on language models
  - Using language models for generation
  - Evaluating language models
- Text classification
- Video lecture on levels of linguistic representation