COS 484/584

L2: Language Models

Last class

$$p(w_1, w_2, w_3, \ldots, w_N) =$$
$$p(w_1)\; p(w_2|w_1)\; p(w_3|w_1, w_2) \times \ldots \times p(w_N|w_1, w_2, \ldots w_{N-1})$$

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

Implicit order

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

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

$$\vdots$$

Maximum likelihood estimate (MLE)

- With a vocabulary of size v,
  - # sequences of length n = v^n
- Typical vocabulary ~ 40k words
  - even sentences of length <= 11 results in more than 4 * 10^50 sequences!
    (# of atoms in the earth ~ 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

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

- 2nd order

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

Andrey Markov

kth 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:

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

(assume $w_j = \phi \quad \forall j < 0$)

(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!

Generations

Unigram

release millions See ABC accurate President of Donald Will cheat them a CNN megynkelly experience @ these word out- the

Bigram

Thank you believe that @ ABC news, Mississippi tonight and the false editorial I think the great people Bill Clinton . ''

Trigram

We are going to MAKE AMERICA GREAT AGAIN! #MakeAmericaGreatAgain https: //t.co/DjkdAzT3WV

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

Generations

Unigram	*release millions See ABC accurate President of Donald Will cheat them a CNN megynkelly experience @ these word out- the*
Bigram	*Thank you believe that @ ABC news, Mississippi tonight and the false editorial I think the great people Bill Clinton . ''*
Trigram	*We are going to MAKE AMERICA GREAT AGAIN! #MakeAmericaGreatAgain https: //t.co/DjkdAzT3WV*

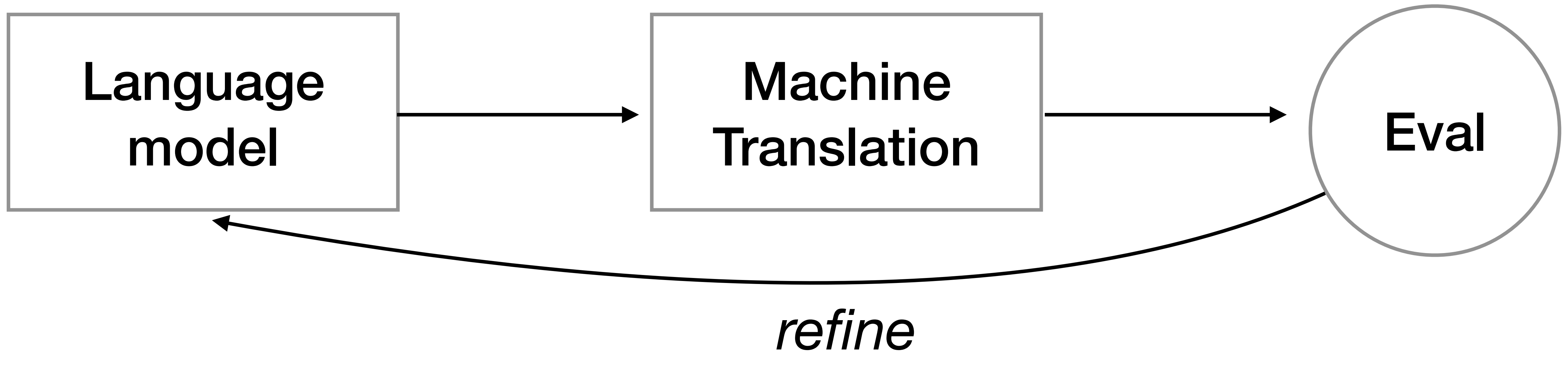
Typical LMs are not sufficient to handle long-range dependencies

"Alice/Bob could not go to work that day because she/he had a doctor's appointment"

Evaluating language models

- A good language model should assign **higher probability** to typical, grammatically correct sentences
- Research process:
  - Train parameters on a suitable training corpus
    - Assumption: observed sentences ~ good sentences
  - Test on *different, unseen* corpus
    - Training on any part of test set not acceptable!
  - Evaluation metric

Extrinsic evaluation



- Train LM -> apply to task -> observe accuracy
- Directly optimized for downstream tasks
  - higher task accuracy -> better model
- Expensive, time consuming
- Hard to optimize downstream objective (indirect feedback)

Perplexity (ppl)

- Measure of how well a probability distribution (or LM) **predicts** a sample

- For a corpus S with sentences $S^1, S^2, \ldots, S^n$

$$\text{ppl}(S) = 2^x \quad \text{where} \quad x = -\frac{1}{W}\sum_{i=1}^{n}\log_2 P(S^i)$$

$\longrightarrow$ Cross-Entropy

where W is the total number of words in test corpus

- Unigram model: $x = -\frac{1}{W}\sum_{i=1}^{n}\sum_{j=1}^{m} log_2 P(w_j^i)$ (since $P(S) = \prod_j P(w_j)$)

- Minimizing perplexity ~ maximizing probability of corpus $P(S^1 S^2 \ldots S^n)$

Intuition on perplexity

If our n-gram model (with vocabulary V) has following probability:

$$P(w_i|w_{i-n}, ...w_{i-1}) = \frac{1}{|V|} \quad \forall w_i$$

what is the perplexity of the test corpus?

$$\text{ppl} = 2^{-\frac{1}{W} W * log(1/|V|)} = |V|$$

(model is 'fine' with observing any word at every step)

Measure of model's uncertainty about next word

$\text{ppl}(S) = 2^x$ where

$$x = -\frac{1}{W}\sum_{i=1}^{n} \log_2 P(S^i)$$

Perplexity as a metric

Pros	Cons

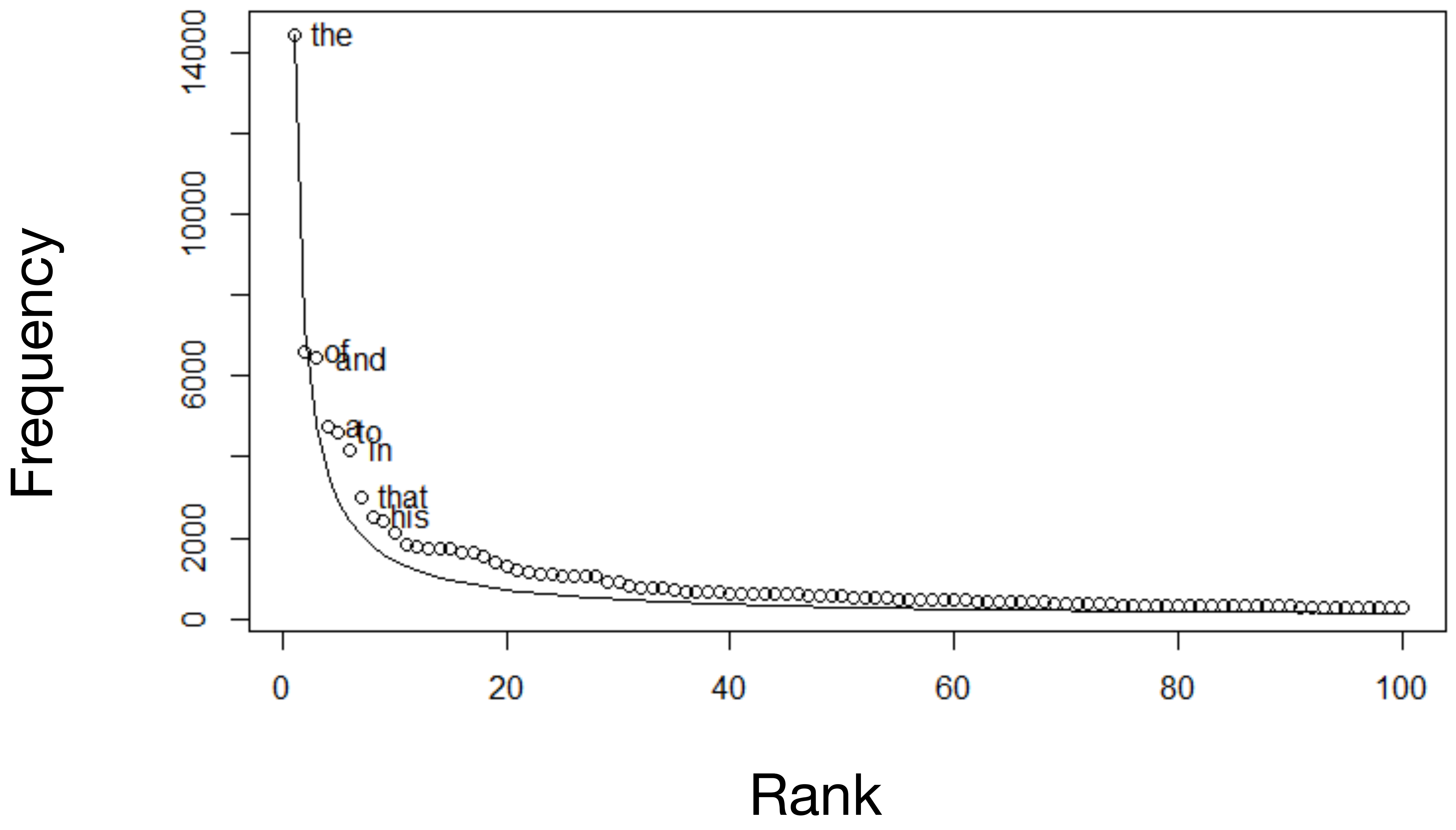
Perplexity as a metric

Pros	Cons
Easy to compute	Requires domain match between train and test
standardized	might not correspond to end task optimization
directly useful, easy to use to correct sentences	log 0 undefined
nice theoretical interpretation - matching distributions	can be ‘cheated’ by predicting common tokens
	size of test set matters
	can be sensitive to low prob tokens/sentences

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
  - Undefined perplexity

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

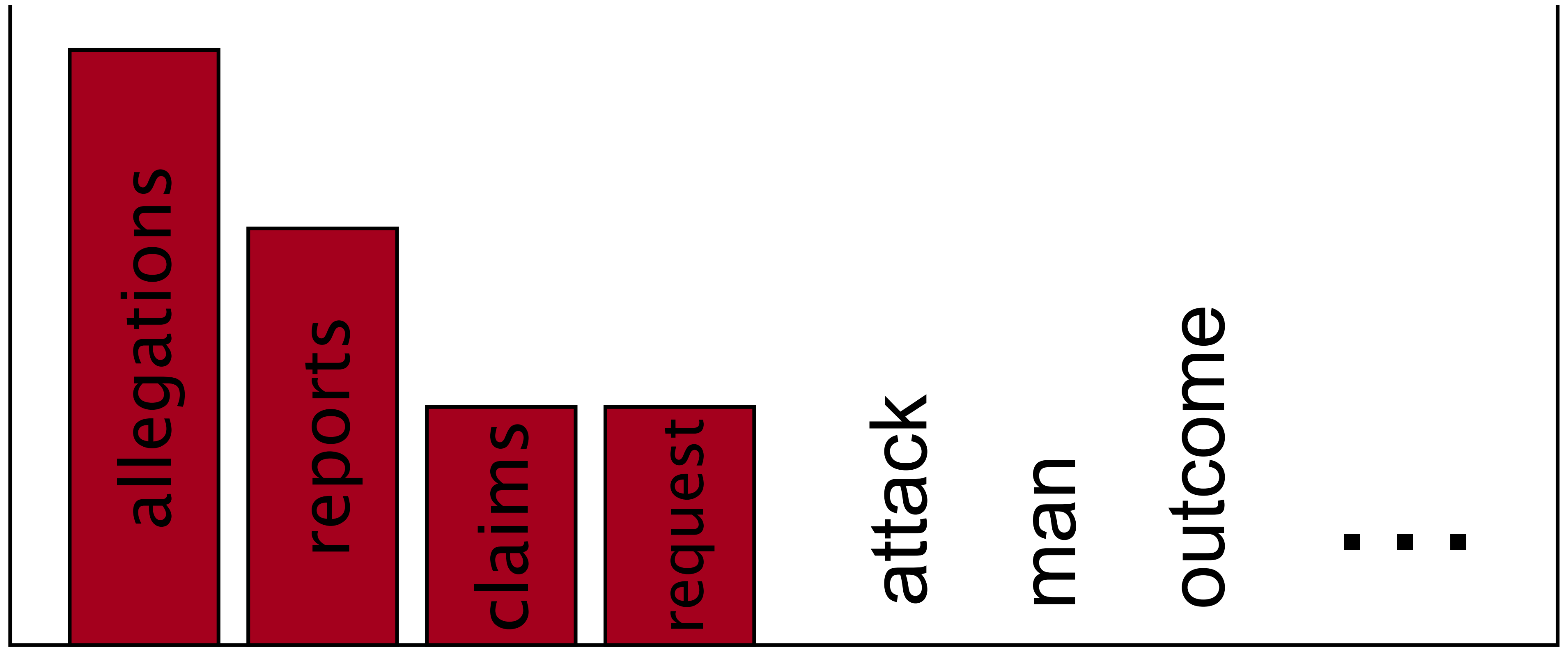
- Handle sparsity by making sure all probabilities are non-zero in our model
  - Additive: Add a small amount to all probabilities
  - Discounting: Redistribute probability mass from observed n-grams to unobserved ones
  - Back-off: Use lower order n-grams if higher ones are too sparse
  - Interpolation: Use a combination of different granularities of n-grams

Smoothing intuition

When we have sparse statistics:

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

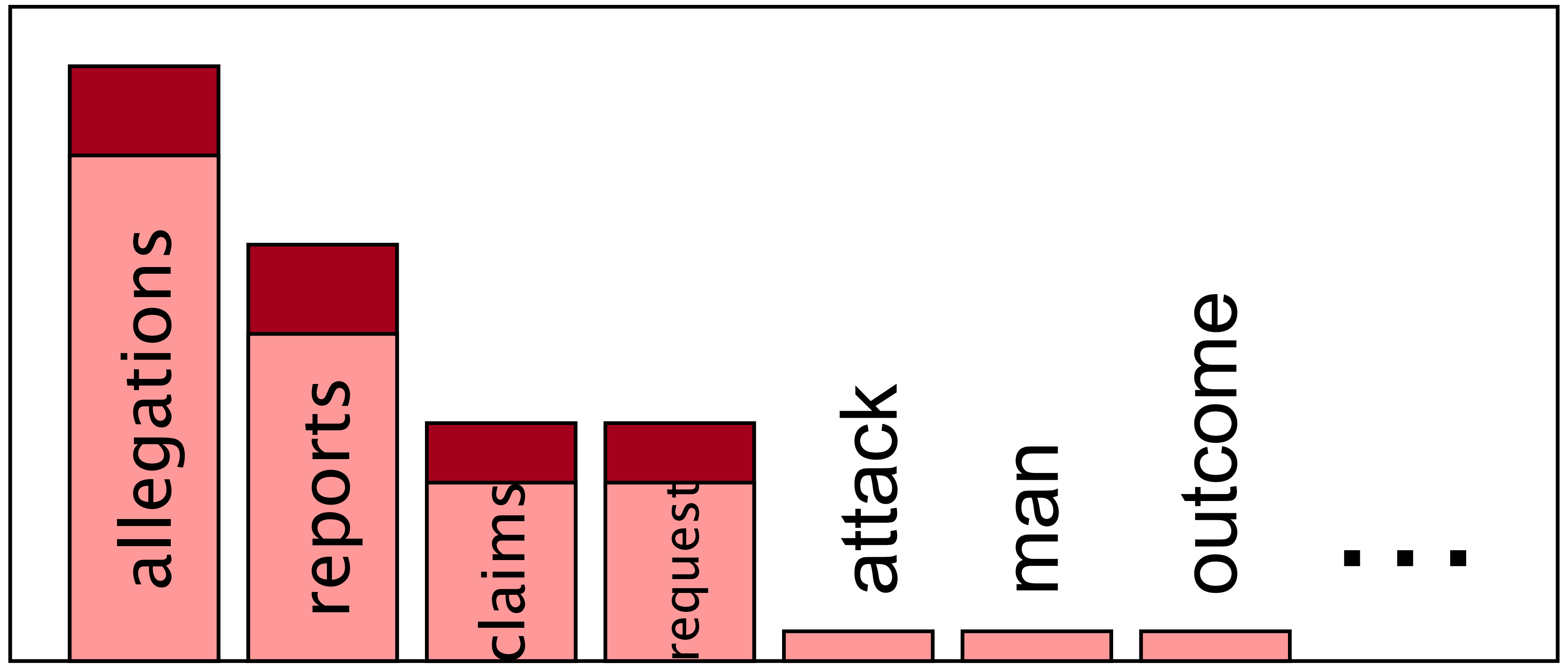
7 total



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)

Laplace smoothing

- Also known as add-alpha

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

- Max likelihood estimate for bigrams:

$$P(w_i|w_{i-1}) = \frac{C(w_{i-1}, w_i)}{C(w_{i-1})}$$

- After smoothing:

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

Raw bigram counts (Berkeley restaurant corpus)

- 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

(Credits: Dan Jurafsky)

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

Add 1 to all the entries in the matrix

(Credits: Dan Jurafsky)

Smoothed bigram probabilities

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

	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

(Credits: Dan Jurafsky)

Problem with Laplace smoothing

Raw counts

$$C(w_{n-1}w_n) = \frac{C(w_{n-1}w_n)}{C(w_{n-1})} \times C(w_{n-1})$$

	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

(Credits: Dan Jurafsky)

Problem with Laplace smoothing

Raw counts

$$C(w_{n-1}w_n) = \frac{C(w_{n-1}w_n)}{C(w_{n-1})} \times C(w_{n-1})$$

	i	want	to	eat	chinese	food	lunch	spend
i	5	$\boxed{827}$	0	9	0	0	0	2
want	$\boxed{2}$	0	608	1	6	6	5	1
to	$\boxed{2}$	0	4	686	2	0	6	211
eat	0	0	2	0	16	2	42	0
chinese	1	0	0	0	0	$\boxed{82}$	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

Reconstituted counts

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

	i	want	to	eat	chinese	food	lunch	spend
i	3.8	$\boxed{527}$	0.64	6.4	0.64	0.64	0.64	1.9
want	$\boxed{1.2}$	0.39	238	0.78	2.7	2.7	2.3	0.78
to	$\boxed{1.9}$	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	$\boxed{8.2}$	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

(Credits: Dan Jurafsky)

Linear Interpolation

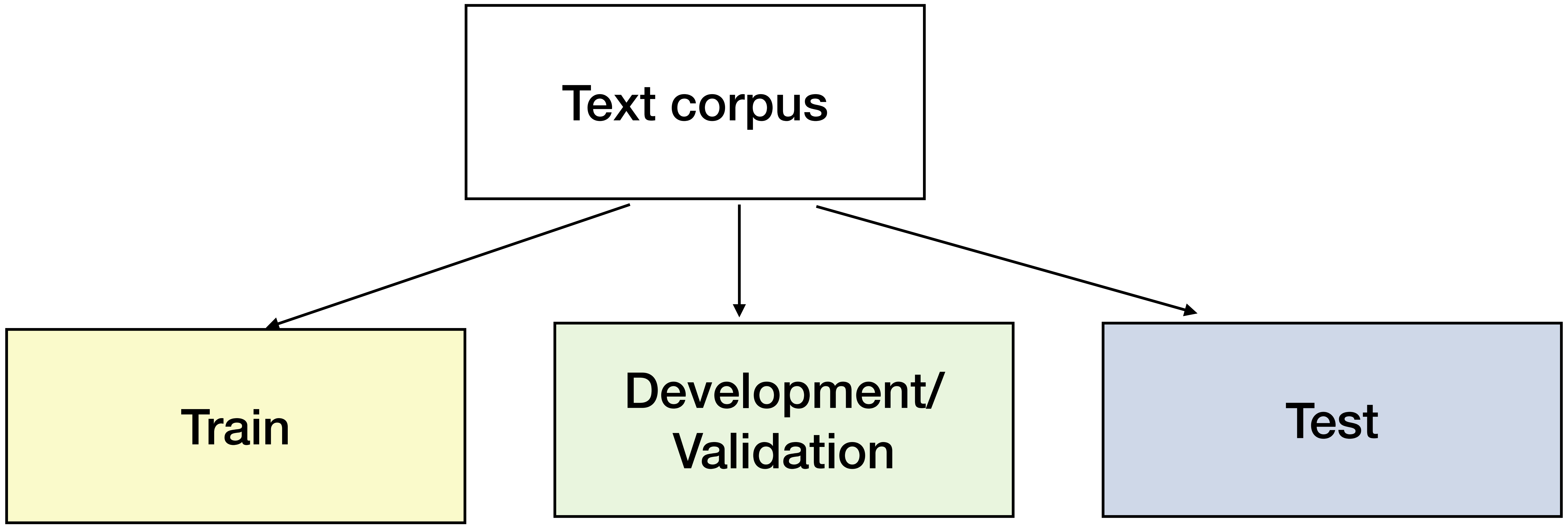
$$\begin{aligned}\hat{P}(w_i|w_{i-1}, w_{i-2}) = \lambda_1 P(w_i|w_{i-1}, w_{i-2}) \\ +\lambda_2 P(w_i|w_{i-1}) \\ +\lambda_3 P(w_i)\end{aligned}$$

Trigram
Bigram
Unigram

$$\sum_i \lambda_i = 1$$

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

Choosing lambdas



- First, estimate n-gram prob. on training set
- Then, estimate lambdas (hyperparameters) to maximize probability on the held-out development/validation set
- Use best model from above to evaluate on test set

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

- Can we do better than naive interpolation?
- Case 1: C (on the mat) = 10, C(on the cat) = 10, C(on the rat) = 10, C(on the bat) = 10, …
- Case 2: C (on the mat) = 40, C(on the cat) = 5, C (on the rat) = 0, C(on the bat) = 0, …
- Which provides a better trigram estimate for P(mat | on the)?
- Larger weights (λ) on non-sparse estimates

Average-count (Chen and Goodman, 1996)

$$P_{\text{interp}}(w_i|w_{i-n+1}^{i-1}) = \lambda_{w_{i-n+1}^{i-1}} P_{\text{ML}}(w_i|w_{i-n+1}^{i-1}) + (1 - \lambda_{w_{i-n+1}^{i-1}})\, P_{\text{interp}}(w_i|w_{i-n+2}^{i-1})$$

- Like simple interpolation, but with more specific lambdas, $\lambda_{w_{i-n+1}^{i-1}}$

- Partition $\lambda_{w_{i-n+1}^{i-1}}$ according to average number of counts per non-zero element:

$$\frac{c(w_{i-n+1}^{i-1})}{|w_i : c(w_{i-n+1}^{i}) > 0|}$$

- Larger $\lambda_{w_{i-n+1}^{i-1}}$ for denser estimates of n-gram probabilities

Discounting

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

- Determine some "mass" to remove from probability estimates
- Redistribute mass among unseen n-grams
- Just choose an absolute value to discount (usually <1)

$$P_{\text{abs\_discount}}(w_i | w_{i-1}) = \frac{c(w_{i-1}, w_i) - d}{c(w_{i-1})} \text{ if } c(w_{i-1}, w_i) > 0$$

$$\alpha(w_{i-1}) \frac{P(w_i)}{\sum_{w'} P(w')} \text{ for all } w' \text{ s.t. } c(w_{i-1}, w') = 0 \text{ if } c(w_{i-1}, w_i) = 0$$

Unigram probabilities

Absolute Discounting

- Define Count*(x) = Count(x) - 0.5

- Missing probability mass:

$$\alpha(w_{i-1}) = 1 - \sum_{w} \frac{\text{Count}^*(w_{i-1}, w)}{\text{Count}(w_{i-1})}$$

$$\alpha(the) = 10 \times 0.5/48 = 5/48$$

- Divide this mass between words w for which Count(the, w) = 0

x	$\text{Count}(x)$	$\text{Count}^*(x)$	$\frac{\text{Count}^*(x)}{\text{Count}(x)}$
the	48		
the, dog	15	14.5	14.5/48
the, woman	11	10.5	10.5/48
the, man	10	9.5	9.5/48
the, park	5	4.5	4.5/48
the, job	2	1.5	1.5/48
the, telescope	1	0.5	0.5/48
the, manual	1	0.5	0.5/48
the, afternoon	1	0.5	0.5/48
the, country	1	0.5	0.5/48
the, street	1	0.5	0.5/48

Back-off

- Use n-gram if enough evidence, else back off to (n-1)-gram

$$P_{bo}(w_i \mid w_{i-n+1} \cdots w_{i-1})$$
$$= \begin{cases} d_{w_{i-n+1}\cdots w_i} \dfrac{C(w_{i-n+1}\cdots w_{i-1}w_i)}{C(w_{i-n+1}\cdots w_{i-1})} & \text{if } C(w_{i-n+1}\cdots w_i) > k \\ \alpha_{w_{i-n+1}\cdots w_{i-1}} P_{bo}(w_i \mid w_{i-n+2}\cdots w_{i-1}) & \text{otherwise} \end{cases}$$

(Katz back-off)

- d = amount of discounting
- α = back-off weight

Other language models

- Discriminative models:
  - train n-gram probabilities to directly maximize performance on end task (e.g. as feature weights)
- Parsing-based models
  - handle syntactic/grammatical dependencies
- Topic models
- Neural networks

We'll see these later on