N-gram models

- Unsmoothed n-gram models
- Smoothing
  - Add-one (Laplacian) **[last class]**
  - Witten-Bell discounting
  - Good-Turing
- Unknown words
- Evaluating n-gram models
- Combining estimators
  - (Deleted) interpolation
  - Backoff

today

Add-one smoothing

- Add one to all of the counts before normalizing into probabilities
- Smoothed unigram probabilities

$$P(w_x) = \frac{count(w_x) + 1}{N + V}$$

N : corpus length in word tokens
V : vocab size (# word types)

- Smoothed bigram probabilities

$$P(w_n \mid w_{n-1}) = \frac{count(w_{n-1}w_n) + 1}{count(w_{n-1}) + V}$$

Adjusted bigram counts

- Original

	I	want	to	eat	Chinese	food	lunch
I	8	1087	0	13	0	0	0
want	3	0	786	0	6	8	6
to	3	0	10	860	3	0	12
eat	0	0	2	0	19	2	52
Chinese	2	0	0	0	0	120	1
food	19	0	17	0	0	0	0
lunch	4	0	0	0	0	1	0

- Adjusted add-one

	I	want	to	eat	Chinese	food	lunch
I	6	740	.68	10	.68	.68	.68
want	2	.42	331	.42	3	4	3
to	3	.69	8	594	3	.69	9
eat	.37	.37	1	.37	7.4	1	20
Chinese	.36	.12	.12	.12	.12	15	.24
food	10	.48	9	.48	.48	.48	.48
lunch	1.1	.22	.22	.22	.22	.44	.22

Too much probability mass is moved

- Estimated bigram frequencies (adjusted counts) for bigrams appearing *r* times
- AP data, 44 million words
  - Church and Gale (1991)
- In general, add-one smoothing is a poor method of smoothing
- Often much worse than other methods in predicting the actual probability for unseen bigrams

$r = f_{MLE}$	f_{emp}	$f_{add\text{-}1}$
0	0.000027	0.000137
1	0.448	0.000274
2	1.25	0.000411
3	2.24	0.000548
4	3.23	0.000685
5	4.21	0.000822
6	5.23	0.000959
7	6.21	0.00109
8	7.21	0.00123
9	8.26	0.00137

N-gram models

- Unsmoothed n-gram models
- Smoothing
  - Add-one (Laplacian)
  - Witten-Bell discounting
  - Good-Turing
- Unknown words
- Evaluating n-gram models
- Combining estimators
  - (Deleted) interpolation
  - Backoff

I scream . You scream . We all scream for ice cream .

What is the number of unseen bigrams?

A. 10
B. 81
C. 70
D. Something else
E. You can’t count unseen bigrams

Methodology: Options

- Divide data into training set and test set
  - Train the statistical parameters on the training set; use them to compute probabilities on the test set
  - Test set: 5%-20% of the total data, but large enough for reliable results

- Divide training into training and validation set
  - » Validation set might be ~10% of original training set
  - » Obtain counts from training set
  - » Tune smoothing parameters on the validation set
- Divide test set into development and final test set
  - Do all algorithm development by testing on the dev set
  - Save the final test set for the very end…use for reported results

Don't train on the test corpus!! Report results on the test data not the training data.

Witten-Bell discounting

- Model the probability of seeing a zero-frequency N-gram by the probability of seeing an N-gram for the first time.
  - *Use the count of things you've seen once to help estimate the count of things you've never seen.*
- Need to compute the probability of seeing an N-gram for the first time
- Estimate the *total* probability mass of all the zero N-grams:

$$\frac{T}{N+T}$$

of *observed* N-gram types

of tokens in the corpus

- Probability of each of *Z* unseen N-grams:

$$p_i^* = \frac{T}{Z(N+T)}$$

Witten-Bell discounting results

- Much better than add-one smoothing
- Used frequently for smoothing speech language models
- Seems to perform poorly when used on small training sets

	I	want	to	eat	Chinese	food	lunch
I	6	740	.68	10	.68	.68	.68
want	2	.42	331	.42	3	4	3
to	3	.69	8	594	3	.69	9
eat	.37	.37	1	.37	7.4	1	20
Chinese	.36	.12	.12	.12	.12	15	.24
food	10	.48	9	.48	.48	.48	.48
lunch	1.1	.22	.22	.22	.22	.44	.22

	I	want	to	eat	Chinese	food	lunch
I	8	1060	.062	13	.062	.062	.062
want	3	.046	740	.046	6	8	6
to	3	.085	10	827	3	.085	12
eat	.075	.075	2	.075	17	2	46
Chinese	2	.012	.012	.012	.012	109	1
food	18	.059	16	.059	.059	.059	.059
lunch	4	.026	.026	.026	.026	1	.026

Good-Turing discounting

- Re-estimates the amount of probability mass to assign to N-grams with zero or low counts by looking at the number of N-grams with higher counts.
- Let N_c be the number of N-grams that occur *c* times.
  - For bigrams, N_0 is the number of bigrams of count 0, N_1 is the number of bigrams with count 1, etc.
- Revised counts:

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

Good-Turing discounting results

- Works very well in practice
- Usually, the GT discounted estimate c* is used only for unreliable counts (e.g. < 5)

$r = f_{MLE}$	f_{emp}	$f_{add\text{-}1}$	f_{GT}
0	0.000027	0.000137	0.000027
1	0.448	0.000274	0.446
2	1.25	0.000411	1.26
3	2.24	0.000548	2.24
4	3.23	0.000685	3.24
5	4.21	0.000822	4.22
6	5.23	0.000959	5.19
7	6.21	0.00109	6.21
8	7.21	0.00123	7.24
9	8.26	0.00137	8.25

N-gram models

- Unsmoothed n-gram models
- Smoothing
  - Add-one (Laplacian)
  - Witten-Bell discounting
  - Good-Turing
- Unknown words
- Evaluating n-gram models
- Combining estimators
  - (Deleted) interpolation
  - Backoff

Unknown words

- Closed vocabulary
  - Vocabulary is known in advance
  - Test set will contain only these words
- Open vocabulary
  - Unknown, out of vocabulary words can occur
  - Add a pseudo-word <UNK>
- Training the unknown word model???

Evaluating n-gram models

- Best way: extrinsic evaluation
  - Embed in an application and measure the total performance of the application
  - End-to-end evaluation
- Intrinsic evaluation
  - Measure quality of the model independent of any application
  - *Perplexity*
    - » *Intuition: the better model is the one that has a tighter fit to the test data or that better predicts the test data*

Perplexity

For a test corpus $W = w_1\ w_2 \ldots w_{N,}$

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

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

The higher the (estimated) probability of the word sequence, the **lower** the perplexity.

Must be computed with models that have no knowledge of the test set.

N-gram models

- Unsmoothed n-gram models (review)
- Smoothing
  - Add-one (Laplacian)
  - Good-Turing
- Unknown words
- Evaluating n-gram models
- Combining estimators
  - (Deleted) interpolation
  - Backoff

Combining estimators

- Smoothing methods
  - Provide the same estimate for all unseen n-grams with the same prefix
  - Make use only of the raw frequency of an n-gram
- But there is an additional source of knowledge we can draw on --- the n-gram "hierarchy"
  - If there are no examples of a particular trigram, $w_{n-2}w_{n-1}w_n$, to compute $P(w_n|w_{n-2}w_{n-1})$, we can estimate its probability by using the bigram probability $P(w_n|w_{n-1})$.
  - If there are no examples of the bigram to compute $P(w_n|w_{n-1})$, we can use the unigram probability $P(w_n)$.
- For n-gram models, suitably combining various models of different orders is often the secret to success.

Simple linear interpolation

- Construct a linear combination of the multiple probability estimates.
  - Weight each contribution so that the result is another probability function.

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

  - Lambda's sum to 1.
- Also known as (finite) *mixture models*
- *Deleted* interpolation
  - Each lambda is a function of the most discriminating context

Backoff (Katz 1987)

- Non-linear method
- The estimate for an n-gram is allowed to back off through progressively shorter histories.
- The most detailed model that can provide sufficiently reliable information about the current context is used.
- Trigram version (high-level):

$$\hat{P}(w_i \mid w_{i-2}w_{i-1}) = \begin{cases} P(w_i \mid w_{i-2}w_{i-1}), & \mathit{if}\ C(w_{i-2}w_{i-1}w_i) > 0 \\ \alpha_1 P(w_i \mid w_{i-1}), & \mathit{if}\ C(w_{i-2}w_{i-1}w_i) = 0 \\ & \mathit{and}\ C(w_{i-1}w_i) > 0 \\ \alpha_2 P(w_i), & \mathit{otherwise.} \end{cases}$$

Final words…

- Problems with backoff?
  - Probability estimates can change suddenly on adding more data when the back-off algorithm selects a different order of n-gram model on which to base the estimate.
  - Works well in practice **in combination with smoothing**.
- Good option: simple linear interpolation with MLE n-gram estimates plus some allowance for unseen words (e.g. Good-Turing discounting)