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

Predicting the next word

- Bigram model

$$P(w_n|w_1^{n-1}) \approx P(w_n|w_{n-1})$$

- Trigram model

$$P(w_n \mid w_1^{n-1}) \approx P(w_n \mid w_{n-2} w_{n-1})$$

- N-gram approximation

$$P(w_1^n) \approx \prod_{k=1}^{n} P(w_k|w_{k-N+1}^{k-1})$$

- Markov assumption: probability of some future event (next word) depends only on a limited history of preceding events (previous words)

Training N-gram models

- N-gram models can be trained by counting and normalizing
  - Bigrams

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

  - General case

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

  - An example of Maximum Likelihood Estimation (MLE)
    - Resulting parameter set is one in which the likelihood of the training set T given the model M (i.e. P(T|M)) is maximized.

Probability of a word sequence

- P (w_1, w_2,…, w_{n-1}, w_n)

$$P(w_1^n) = P(w_1)\ P(w_2|w_1)\ P(w_3|w_1^2)\ \ldots P(w_n|w_1^{n-1})$$

$$= \prod_{k=1}^{n} P(w_k|w_1^{k-1})$$

Bigram counts

	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

- Note the number of 0’s…

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

Smoothing

- Need better estimators than MLE for rare events
- Approach
  - Somewhat decrease the probability of previously seen events, so that there is a little bit of probability mass left over for previously unseen events
    - » Smoothing
    - » Discounting methods

Add-one smoothing

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

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

- Smoothed unigram probabilities

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

- Adjusted counts (unigrams)

$$c_i^* = (c_i+1)\frac{N}{N+V}$$

Add-one smoothing: bigrams

Add-one bigram counts

- Original counts

	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

- New counts

	I	want	to	eat	Chinese	food	lunch
I	9	1088	1	14	1	1	1
want	4	1	787	1	7	9	7
to	4	1	11	861	4	1	13
eat	1	1	3	1	20	3	53
Chinese	3	1	1	1	1	121	2
food	20	1	18	1	1	1	1
lunch	5	1	1	1	1	2	1

Add-one smoothed bigram probabilites

- Original

	I	want	to	eat	Chinese	food	lunch
I	.0023	.32	0	.0038	0	0	0
want	.0025	0	.65	0	.0049	.0066	.0049
to	.00092	0	.0031	.26	.00092	0	.0037
eat	0	0	.0021	0	.020	.0021	.055
Chinese	.0094	0	0	0	0	.56	.0047
food	.013	0	.011	0	0	0	0
lunch	.0087	0	0	0	0	.0022	0

- Add-one smoothing

	I	want	to	eat	Chinese	food	lunch
I	.0018	.22	.00020	.0028	.00020	.00020	.00020
want	.0014	.00035	.28	.00035	.0025	.0032	.0025
to	.00082	.00021	.0023	.18	.00082	.00021	.0027
eat	.00039	.00039	.0012	.00039	.0078	.0012	.021
Chinese	.0016	.00055	.00055	.00055	.00055	.066	.0011
food	.0064	.00032	.0058	.00032	.00032	.00032	.00032
lunch	.0024	.00048	.00048	.00048	.00048	.00096	.00048

Too much probability mass is moved

- Adjusted bigram counts
- AP data, 44million words
- Church and Gale (1991)
- In general, add-one smoothing is a poor method of smoothing
- Much worse than other methods in predicting the actual probability for unseen bigrams

$r = f_{MLE}$	f_{emp}	f_{add-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

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)
- As with other discounting methods, it is the norm to treat N-grams with low counts (e.g. counts of 1) as if the count was 0

$r = f_{MLE}$	f_{emp}	f_{add-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 (review)
- Smoothing
  - Add-one (Laplacian)
  - 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>

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 set $W = w_1\ w_2 \ldots w_{N,}$

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

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

The higher the conditional 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 (or rare) n-grams
  - 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 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 (first try):

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

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.
- Good option: simple linear interpolation with MLE n-gram estimates plus some allowance for unseen words (e.g. Good-Turing discounting)