N-gram models

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

Goals

- Determine the next word in a sequence
  - ***Probability distribution*** across all words in the language
  - $P\,(w_n \mid w_1\ w_2\ \ldots\ w_{n-1})$
- Determine the ***probability of a sequence*** of words
  - $P\,(w_1\ w_2\ \ldots\ w_{n-1}\ w_n)$

Probability of a word sequence

- $P(w_1\ w_2 \ldots w_{n-1}\ w_n)$

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

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

- Problem?

Predict the next word

- $P(w_n | w_1 w_2 \ldots w_{n-1}) = P(w_n | w_1^{n-1})$
  - Same problem

- Solution: *approximate* the probability of a word given all the previous words…

N-gram approximations

- Bigram model

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

- Trigram model

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

Training N-gram models

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

A. count(w_{n-2} w_{n-1} w_n) / count(w_{n-2} w_{n-1})

B. count(w_{n-1} w_n) / count(w_{n-1})

C. count(w_{n-N+1}^{n-1} w_n) / count(w_{n-N+1}^{n-1})

D. A and C

E. B and C

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.

Bigram counts: MLE

	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

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

corpus length in word tokens

- Smoothed unigram probabilities

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

vocab size (# word types)

- Add-one smoothing: bigrams

  [example on board]

Add-one smoothing: bigrams

- MLE bigram probabilities

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

- Laplacian bigram probabilities

count(w_{n-1} w_n) **+ 1** / count(w_{n-1})

A. YES
B. NO
C. I don’t know

Add-one smoothing: bigrams

- MLE bigram probabilities

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

- Laplacian bigram probabilities

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

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 (add-one 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

Bigram probabilities

- 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

Adjusted counts

- Adjusted/discounted counts

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

bigram case

$$c_i^*(w_{n-1}w_n) = (c(w_{n-1}w_n) + 1)\frac{\mathrm{N}}{c(w_{n-1}) + V}$$

- Discount d_c $\quad d_c = \frac{c^*}{c}$

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!

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

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.

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-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