CS674 Natural Language Processing

- Last class
  - Introduction to generative models of language
    - Statistics of natural language
    - Unsmoothed N-grams
- Today
  - Smoothing
    - Add-one
    - Witten-Bell
    - Good-Turing
  - Training issues

N-gram approximations

- Markov assumption: only the prior local context --- the last few words --- matters
- N-gram approximation

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

Training N-gram models

- N-gram models can be trained by counting and normalizing
  - MLE estimates from relative frequencies
  - Bigram model

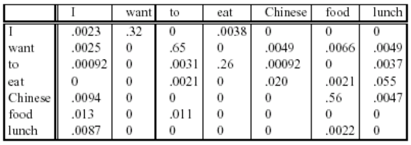
$$P(w_n \mid w_1^{n-1}) = \frac{C(w_{n-1}w_n)}{C(w_{n-1})}$$

  - General form

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

Bigram probabilities

- Problem with the maximum likelihood estimate: sparse data



	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

Smoothing

- Need better estimators 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
- Normal unigram probabilities

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

- Smoothed unigram probabilities

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

- Adjusted counts

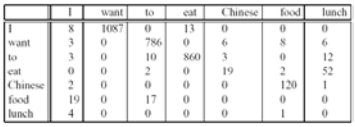
$$c_i^* = (c_i + 1)\frac{N}{N+V}$$

Adjusted bigram counts

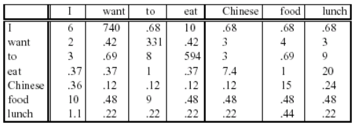
- Discount d_c

$$d_c = \frac{c^*}{c}$$

- Ratio of the discounted counts to the 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



	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
- 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
- Variances of the counts are worse than those from the unsmoothed MLE method

$r = f_{MLE}$	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

Methodology

- Cardinal sin: Testing on the training corpus
- 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: 5-10% of the total data, but large enough for reliable results
- Divide training into training and validation/held out set
  - » Obtain counts from training
  - » 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…

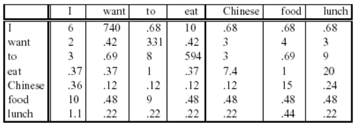
Witten-Bell discounting

- 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: $\frac{T}{N+T}$
- Probability of each of *Z* unseen N-grams:

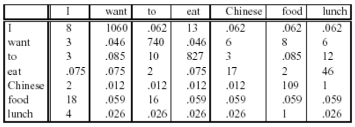
$$p_i^* = \frac{T}{Z(N+T)}$$

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.
- So, applying the idea to smoothing the joint probability of bigrams, N_0 is the number of bigrams *b* of count 0, N_1 is the number of bigrams *b* 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}	$f_{add\text{-}1}$	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