N-gram language modeling (continued)

Chapter 4 J&M'09
Chapter 6 of M&S'00

Smoothing

Coping with the sparseness problem: **smoothing**

1. Laplace
2. Good-Turing discounting
3. Kneser-Ney
4. Katz backoff
5. ...

Laplace Smoothing (aka. *add-one* smoothing)

Very simple: just add 1 to all the counts.

- Unigram case:
  $$P^*(w_i) = \frac{C(w_i) + 1}{N + |V|}$$
- Bigram case:
  $$P^*(w_n|w_{n-1}) = \frac{C(w_{n-1}w_n) + 1}{C(w_{n-1}) + |V|}$$

N = total number of tokens

(This smoothing technique works best when there aren't too many 0's.)

Add-alpha smoothing generalizes Laplace smoothing:

For example, for unigrams:

$$P_\alpha(w_i) = \frac{C(w_i) + \alpha}{N + (\alpha \times |V|)}$$

Original unsmoothed bigram counts:

	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

Laplace 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

Laplace 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

Original bigram probabilities:

	i	want	to	eat	chinese	food	lunch	spend
i	0.002	0.33	0	0.0036	0	0	0	0.00079
want	0.0022	0	0.66	0.0011	0.0065	0.0065	0.0054	0.0011
to	0.00083	0	0.0017	0.28	0.00083	0	0.0025	0.087
eat	0	0	0.0027	0	0.021	0.0027	0.056	0
chinese	0.0063	0	0	0	0	0.52	0.0063	0
food	0.014	0	0.014	0	0.00092	0.0037	0	0
lunch	0.0059	0	0	0	0	0.0029	0	0
spend	0.0036	0	0.0036	0	0	0	0	0

Smoothing

Measure effect of Laplace Smoothing on the counts by reconstructing them:

$$c^*(w_{n-1}, w_n) = \frac{[C(w_{n-1}, w_n) + 1] \times C(w_{n-1})}{C(w_{n-1}) + |V|}$$

Sometimes too much probability is moved to the zeros...

	i	want	to	eat	chinese	food	lunch	spend
i	3.8	527	0.64	6.4	0.64	0.64	0.64	1.9
want	1.2	0.39	238	0.78	2.7	2.7	2.3	0.78
to	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	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

C(want to) went from 608 to 238.

Discount: $d = \frac{c^*}{c}$ (ratio between new and old counts)

Discount for 'want to' is .39, for 'Chinese food' is .10

Good-Turing Smoothing

(textbook is a bit misleading; see this for previous slide version)

Bigram case:

- Let c^* be the smoothed count of N-grams that occur c times:

$$c^*(w_{n-1}, w_n) = (C(w_{n-1}, w_n) + 1)\frac{N_{c+1}}{N_c}$$

- P^*_{GT}(n-grams with zero counts) =

$$\frac{c^*(w_{n-1}, w_n)}{C(w_{n-1})} = \frac{(0+1)\frac{N_{0+1}}{N_0}}{C(w_{n-1})} = \frac{\frac{N_1}{N_0}}{C(w_{n-1})}$$

- P^*_{GT}(n-grams with non-zero counts) =

$$\frac{c^*(w_{n-1}, w_n)}{C(w_{n-1})} = \frac{(c+1)\frac{N_{c+1}}{N_c}}{C(w_{n-1})}$$

Smoothing

Example (based on the bigram model on UB Learns)

Number of observed bigrams $N = 423053$
$N_0 =$ 1195975868 ($= V^2$ – N)
$N_1 = 300348$
$N_3 = 20969$
$N_4 = 11661$
$N_9 = 2142$
$N_{10} = 1751$

$P(\cdot\ \textit{today is friday}\ \cdot) =$
$P_{GT}(\cdot\ \textit{today}) \times P_{GT}(\textit{today is}) \times P_{GT}(\textit{is, friday}) \times P_{GT}(\textit{friday}\ \cdot)$

$P_{GT}(\cdot\ \textit{today}) = (9+1)\frac{1751}{2142}/70751 = 0.0001$

$P_{GT}(\textit{today is}) = \frac{300348}{1195975868}/94 = 0.0000026$

Smoothing

Although Good-Turing is not often used, when it is, the following rules of thumb apply:

- Sometimes $N_{c+1} = 0$, in which case the zeros are replaced with some smoothed value. Eg. **Simple Good-Turing**.
- Counts above 5 are assumed to be reliable, so: $c^* = c$ for $c > 5$
- $c = 1$ are often treated as if they were $c = 0$

Smoothing

AP Newswire corpus, 22 million bigrams

AP Newswire		
c (MLE)	N_c	c^* (GT)
0	74,671,100,000	0.0000270
1	2,018,046	0.446
2	449,721	1.26
3	188,933	2.24
4	105,668	3.24
5	68,379	4.22
6	48,190	5.19

In this example, the 'discounting' (difference) is of approx 0.75:

$1 - 0.446 = 0.554$

$3 - 2.24 = 0.76$

$5 - 4.22 = 0.78$

Why not save time and simply subtract $d = 0.75$?

Smoothing

Absolute Discounting Interpolation (bigram case)

$$P_{abs}(w_{n-1}, w_n) = \begin{cases} \dfrac{C^*(w_{n-1}, w_n)}{C(w_{n-1})} & \text{if} \quad C(w_{n-1}, w_n) > 0 \\ \alpha(w_n) P(w_n) & \text{otherwise} \end{cases}$$

Where

$$C^*(w_{n-1}, w_n) = C(w_{n-1}, w_n) - d$$

For $d = 0.75$.

$$\alpha(w_n) = 1 - \sum_{w} \frac{C^*(w_n, w)}{C(w_n)}$$

Smoothing

Katz backoff: only n-gram with zero counts are re-estimated based on non-zero lower-order n-grams (like in GT smoothing)

For the bigram case, define two sets:

$$A(w_{n-1}) = \{w : C(w_{n-1}, w) > 0\}$$

$$B(w_{n-1}) = \{w : C(w_{n-1}, w) = 0\}$$

$$P_{katz}(w_{n-1}, w_n) = \begin{cases} \dfrac{C^*(w_{n-1}, w_n)}{C(w_n)} & \text{if} \quad w_n \in A(w_{n-1}) \\ \alpha(w_{n-1}) P^*(w_n) & \text{if} \quad w_n \in B(w_{n-1}) \end{cases}$$

Smoothing

The missing probability mass is

$$\alpha(w_n) = 1 - \sum_{w \in A(w_{n-1})} \frac{c^*(w_n, w)}{C(w_n)}$$

... which is distributed among the unigrams, in proportion:

$$P^*(w_n) = \frac{C(w_n)/N}{\sum_{w \in B(w_{n-1})} C(w)/N}$$

(where N is as usual the total frequency of all observed words)

Smoothing

Kneser-Ney Smoothing:

- The more often smoothing is needed to cope with 0 probabilities, the larger the smoothing parameter should be.
- KN backs off to a number proportional to the number of different word types that can precede a word.

Unigram case:

$$P_{KN}(w_i) = \frac{|\{w_{i-1} : C(w_{i-1}, w_i) > 0\}|}{\sum_{w_i} |\{w_i : C(w_{i-1}, w_i) > 0\}|}$$

Smoothing

Bigram case (interpolated; textbook's version is incomplete):

$$P_{KN}(w_{i-1}, w_i) = \frac{max(C(w_{i-1}, w_i) - d, 0)}{C(w_{i-1})} + \lambda(w_{i-1}) P_{continuation}(w_i)$$

$$\lambda(w_{i-1}) = \frac{d}{C(w_{i-1})} |\{w : C(w_{i-1}, w) > 0\}|$$

$$P_{continuation}(w) = \frac{|\{w_{i-1} : C(w_{i-1}, w) > 0\}|}{|\{(w_{i-1}, w_j) : C(w_{j-1}, w_j) > 0\}|}$$

How to evaluate the performance of a Language Model (LM)?

- Take a corpus (a collection of texts) and select two subsets:
- a **training set**: 90%, used to create the LM
- a **test set**: 10% data, used to evaluate the LM
  A good LM assigns on average high probability to the test set.

Smoothing

Perplexity: the inverse probability of the test set, normalized by the number of words
(minimizing perplexity is the same as maximizing probability)

$$PP(w_1 ... w_n) = P(w_1 ... w_n)^{-\frac{1}{n}} = \sqrt[n]{\frac{1}{P(w_1 ... w_n)}}$$

(normalized by the length n of the test set)

Applying the chain rule:

$$PP(w_1 ... w_n) = \sqrt[n]{\prod_{i=1}^{n} \frac{1}{P(w_i | w_1 ... w_{i-1})}}$$

For bigrams:

$$PP(w_1 ... w_n) = \sqrt[n]{\prod_{i=1}^{n} \frac{1}{P(w_i | w_1)}}$$

Smoothing

[J&M'09] used the Wall Street Journal to evaluate N-grams:

- a 35 million word training set
- a 15 million word test set
- PP of unigram LM: 962
- PP of bigram LM: 170
- PP of trigram LM: 109

However, the best way to actually evaluate an LM is **extrinsically**

1. Put model A into a larger application, and evaluate the performance of the whole system
2. Put model B into a larger application, and evaluate the performance of the whole system
3. Compare performance of the application with the two models

Drawback: very time-consuming