Smoothing

Parameter Estimation

$p(x_1=h, x_2=o, x_3=r, x_4=s, x_5=e, x_6=s, ...)$
$\approx p(h \mid BOS, BOS)$
$* p(o \mid BOS, h)$
$* p(r \mid h, o)$
$* p(s \mid o, r)$
$* p(e \mid r, s)$
$* p(s \mid s, e)$
$* \ldots$

trigram model's parameters

values of those parameters, as naively estimated from Brown corpus.

4470/52108
395/ 4470
1417/14765
1573/26412
1610/12253
2044/21250

How to Estimate?

- p(z | xy) = ?
- Suppose our training data includes
  ... xya ..
  ... xyd ...
  ... xyd ...
  but never xyz
- Should we conclude
  p(a | xy) = 1/3?
  p(d | xy) = 2/3?
  p(z | xy) = 0/3?
- NO! Absence of xyz might just be bad luck.

Smoothing the Estimates

- Should we conclude
  p(a | xy) = 1/3? *reduce this*
  p(d | xy) = 2/3? *reduce this*
  p(z | xy) = 0/3? *increase this*
- **Discount** the positive counts somewhat
- **Reallocate** that probability to the zeroes
- Especially if the denominator is small ...
  - 1/3 probably too high, 100/300 probably about right
- Especially if numerator is small ...
  - 1/300 probably too high, 100/300 probably about right

Add-One Smoothing

xya	1	1/3	2	2/29
xyb	0	0/3	1	1/29
xyc	0	0/3	1	1/29
xyd	2	2/3	3	3/29
xye	0	0/3	1	1/29
...				
xyz	0	0/3	1	1/29
Total xy	3	3/3	29	29/29

Add-One Smoothing

300 observations instead of 3 – better data, less smoothing

xya	100	100/300	101	101/326
xyb	0	0/300	1	1/326
xyc	0	0/300	1	1/326
xyd	200	200/300	201	201/326
xye	0	0/300	1	1/326
...				
xyz	0	0/300	1	1/326
Total xy	300	300/300	326	326/326

Add-One Smoothing

Suppose we're considering 20000 word types, not 26 letters

xya	1	1/3	2	2/29
xyb	0	0/3	1	1/29
xyc	0	0/3	1	1/29
xyd	2	2/3	3	3/29
xye	0	0/3	1	1/29
...				
xyz	0	0/3	1	1/29
Total xy	3	3/3	29	29/29

Add-One Smoothing

As we see more word types, smoothed estimates keep falling

see the abacus	1	1/3	2	2/20003
see the abbot	0	0/3	1	1/20003
see the abduct	0	0/3	1	1/20003
see the above	2	2/3	3	3/20003
see the Abram	0	0/3	1	1/20003
...				
see the zygote	0	0/3	1	1/20003
Total	3	3/3	20003	20003/20003

Add-Lambda Smoothing

- Suppose we're dealing with a vocab of 20000 words
- As we get more and more training data, we see more and more words that need probability – the probabilities of existing words keep dropping, instead of converging
- This can't be right – eventually they drop too low.

- So instead of adding 1 to all counts, add $\lambda = 0.01$
- This gives much less probability to those extra events
- But how to pick *best value* for α? (for the size of our training corpus)
- Try lots of values on a simulated test set! "held-out data"
- Or even better: 10-fold cross validation (aka "jackknifing")
  - Divide data into 10 subsets
  - To evaluate a given alpha:
    - Measure performance on each subset when other 9 are used for training
    - Average performance over the 10 subsets tells us how good alpha is

Terminology

- Word **type** = distinct vocabulary item
- Word **token** = occurrence of that type
- A dictionary is a list of types (once each)
- A corpus is a list of tokens (each type has many tokens)

26 types	*300 tokens*	
a	100	*100 tokens of this type*
b	0	*0 tokens of this type*
c	0	
d	200	*200 tokens of this type*
e	0	

Always treat zeroes the same?

20000 types	*300 tokens*		*300 tokens*
a	150		0
both	18		0
candy	0		1
donuts	0		2
every	50	versus	0
farina	0		0
grapes	0	0/300	1
his	38		0
ice cream	0		7

0/300

which zero would you expect is really rare?

Always treat zeroes the same?

20000 types	*300 tokens*		*300 tokens*
a	150		0
both	18		0
candy	0		1
donuts	0		2
every	50	versus	0
farina	0		0
grapes	0		1
his	38		0
ice cream	0		7

determiners:
a closed class

Always treat zeroes the same?

20000 types	300 tokens		300 tokens
a	150		0
both	18		0
candy	0		1
donuts	0		2
every	50	versus	0
farina	0		0
grapes	0		1
his	38		0
ice cream	0		7

(food) nouns: an open class

Good-Turing Smoothing

- Intuition: Can judge rate of novel events by rate of singletons.
- Let N_r = # of word types with r training tokens
  - e.g., N_0 = number of unobserved words
  - e.g., N_1 = number of singletons
- Let N = Σ r N_r = total # of training tokens

Good-Turing Smoothing

- Let N_r = # of word types with r training tokens
- Let N = Σ r N_r = total # of training tokens
- Naïve estimate: if x has r tokens, p(x) = ?
  - Answer: r/N
- Total naïve probability of **all** words with r tokens?
  - Answer: N_r r / N.
- Good-Turing estimate of this **total** probability:
  - Defined as: N_{r+1} (r+1) / N
  - So proportion of novel words in test data is estimated by proportion of singletons in training data.
  - Proportion in test data of the N_1 singletons is estimated by proportion of the N_2 doubletons in training data. Etc.
- So what is Good-Turing estimate of p(x)?

Use the backoff, Luke!

- Why are we treating all novel events as the same?
- p(zygote | see the) vs. p(baby | see the)
  - Suppose both trigrams have zero count
- baby beats zygote as a unigram
- the baby beats the zygote as a bigram
- see the baby beats see the zygote ?
- As always for backoff:
  - Lower-order probabilities (unigram, bigram) aren't quite what we want
  - But we *do* have enuf data to estimate them & they're better than nothing.

Smoothing + backoff

- Basic smoothing (e.g., add-λ or Good-Turing):
  - Holds out some probability mass for novel events
  - E.g., Good-Turing gives them total mass of N_1/N
  - Divided up **evenly** among the novel events
- Backoff smoothing
  - Holds out same amount of probability mass for novel events
  - But divide up **unevenly** in proportion to backoff prob.
  - For p(z | xy):
    - Novel events are types z that were never observed after xy
    - Backoff prob for p(z | xy) is p(z | y) ... which in turn backs off to p(z)!
  - Note: How much mass to hold out for novel events in context xy?
    - Depends on whether position following xy is an open class
    - Usually not enough data to tell, though, so aggregate with other contexts (all contexts? similar contexts?)

Deleted Interpolation

- Can do even simpler stuff:
  - Estimate p(z | xy) as weighted average of the naïve MLE estimates of p(z | xy), p(z | y), p(z)
- The weights can depend on the context xy
  - If a lot of data are available for the context, then trust p(z | xy) more since well-observed
  - If there are not many singletons in the context, then trust p(z | xy) more since closed-class
  - Learn the weights on held-out data w/ jackknifing