SI425 : NLP

Set 4

Smoothing Language Models

Fall 2017 : Chambers

Review: evaluating n-gram models

- Best evaluation for an N-gram
  - Put model A in a speech recognizer
  - Run recognition, get word error rate (WER) for A
  - Put model B in speech recognition, get word error rate for B
  - Compare WER for A and B
  - **In-vivo evaluation**

Difficulty of in-vivo evaluations

- In-vivo evaluation
  - Very time-consuming
- Instead: perplexity

Perplexity

- Perplexity is the probability of the test set (assigned by the language model), normalized by the number of words:

$$\begin{aligned} \mathrm{PP}(W) &= P(w_1 w_2 \ldots w_N)^{-\frac{1}{N}} \\ &= \sqrt[N]{\frac{1}{P(w_1 w_2 \ldots w_N)}} \end{aligned}$$

- Chain rule:

$$\mathrm{PP}(W) = \sqrt[N]{\prod_{i=1}^{N} \frac{1}{P(w_i | w_1 \ldots w_{i-1})}}$$

- For bigrams:

$$\mathrm{PP}(W) = \sqrt[N]{\prod_{i=1}^{N} \frac{1}{P(w_i | w_{i-1})}}$$

  - Minimizing perplexity is the same as maximizing probability
    - **The best language model is one that best predicts an unseen test set**

Lesson 1: the perils of overfitting

- N-grams only work well for word prediction if the test corpus looks like the training corpus
  - In real life, it often doesn’t
  - We need to train robust models, adapt to test set, etc

Lesson 2: zeros or not?

- Zipf's Law:
  - A small number of events occur with high frequency
  - A large number of events occur with low frequency

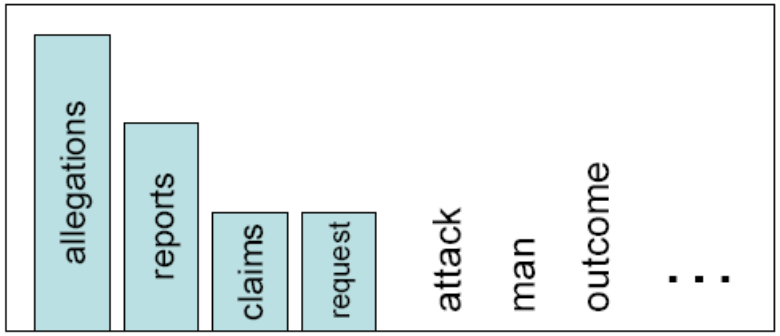
- Resulting Problem:
  - You might have to wait an arbitrarily long time to get valid statistics on low frequency events
  - Our estimates are sparse! No counts exist for the vast bulk of things we want to estimate!

- Solution:
  - **Estimate** the likelihood of unseen N-grams

Smoothing is like Robin Hood: Steal from the rich, give to the poor (probability mass)

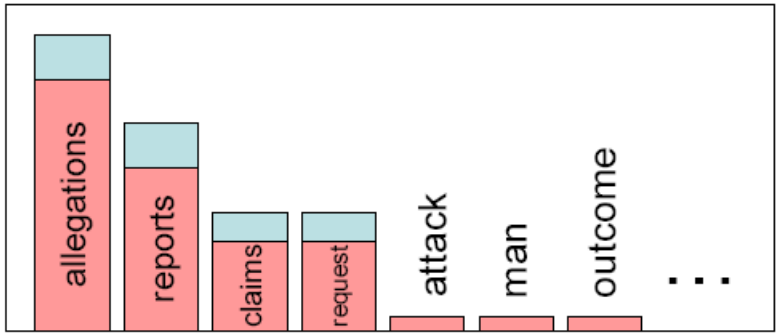
- We often want to make predictions from sparse statistics:

P(w | denied the)
3 allegations
2 reports
1 claims
1 request
7 total



- Smoothing flattens spiky distributions so they generalize better

P(w | denied the)
2.5 allegations
1.5 reports
0.5 claims
0.5 request
2 other
7 total



- Very important all over NLP, but easy to do badly!

Slide from Dan Klein

Laplace smoothing

- Also called "*add-one smoothing*"
- Just add one to all the counts!

- MLE estimate: $$P(w_i) = \frac{c_i}{N}$$

- Laplace estimate: $$P_{\text{Laplace}}(w_i) = \frac{c_i + 1}{N + V}$$

- Reconstructed counts: $$c_i^* = (c_i + 1)\frac{N}{N + V}$$

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 bigrams

$$P^*(w_n|w_{n-1}) = \frac{C(w_{n-1}w_n)+1}{C(w_{n-1})+V}$$

	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

Reconstituted counts

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

	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

Note big change to counts

- C(“want to”) went from 609 to 238!
- P(to|want) from .66 to .26!

- Laplace smoothing not often used for n-grams, as we have much better methods
- Despite its flaws, Laplace (add-k) is still used to smooth other probabilistic models in NLP, especially
  - For pilot studies
  - In domains where the number of zeros isn’t so huge.

Exercise

I stay out too late
Got nothing in my brain
That's what people say mmmm
That's what people say mmmm

- Using a *unigram model* and Laplace smoothing (+1)
  - Calculate P("what people mumble")
  - Assume a vocabulary based on the above, plus the word "possibly"

- Now instead of k=1, set k=0.01
  - Calculate P("what people mumble")

Better discounting algorithms

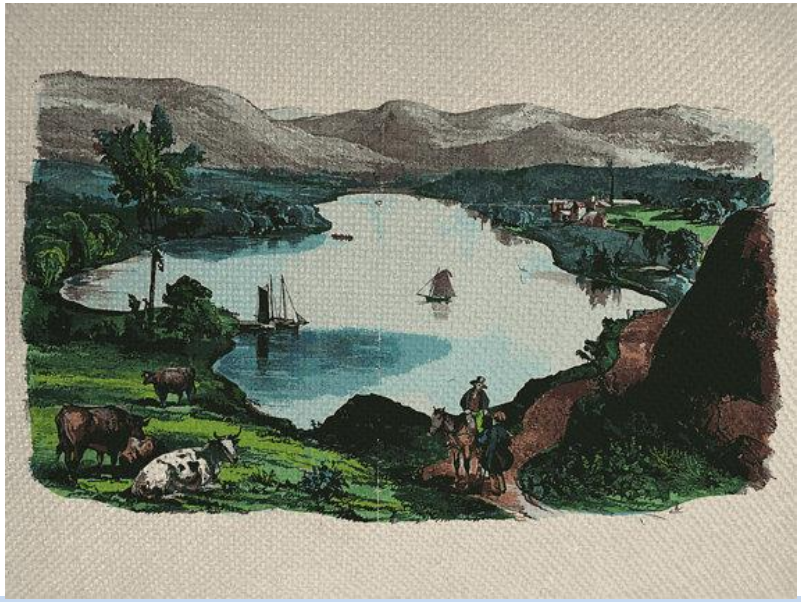
- Intuition: use the count of things we've seen once to help estimate the count of things we've never seen

- Intuition in many smoothing algorithms:
  - Good-Turing
  - Kneser-Ney
  - Witten-Bell

Good-Turing: Josh Goodman intuition

- Imagine you are fishing
  - There are 8 species in the lake: carp, perch, whitefish, trout, salmon, eel, catfish, bass
- You catch:
  - 10 carp, 3 perch, 2 whitefish, 1 trout, 1 salmon, 1 eel = **18 fish**

- How likely is it the next species is *new* (catfish or bass)?
  - **3/18**
- And how likely is it that the next species is another trout?
  - **Less than 1/18**



Good Turing Counts

- How probable is an *unseen* fish?
- What number can we use based on our evidence?

- Use the counts of what we have seen ***once*** to estimate things we have seen ***zero*** times.

Good-Turing Counts

- N[x] is the frequency-of-frequency-x
  - So for the fish: N[10]=1, N[1]=3, etc.

- To estimate the total number of unseen species:
  - Use the number of species (words) we've seen once
  - $c[0]^* = N[1]$ $p_0 = c[0]^*/N = N[1]/N = 3/18$
  - P_{GT}*(things with frequency zero in training)* $= \frac{N[1]}{N}$

- All other estimates are adjusted (down)

$$c[x]^* = (x+1)\frac{N[x+1]}{N[x]} \qquad P_{GT}(occurred\ x\ times) = \frac{c[x]^*}{N}$$

	unseen (bass or catfish)	trout
c	0	1
MLE p	$p = \frac{0}{18} = 0$	$\frac{1}{18}$
c^*		$c^*(\text{trout}) = 2 \times \frac{N_2}{N_1} = 2 \times \frac{1}{3} = .67$
GT p^*_{GT}	$p^*_{\text{GT}}(\text{unseen}) = \frac{N_1}{N} = \frac{3}{18} = .17$	$p^*_{\text{GT}}(\text{trout}) = \frac{.67}{18} = \frac{1}{27} = .037$

Bigram frequencies of frequencies and GT re-estimates

AP Newswire			Berkeley Restaurant—		
c (MLE)	N_c	c^* (GT)	c (MLE)	N_c	c^* (GT)
0	74,671,100,000	0.0000270	0	2,081,496	0.002553
1	2,018,046	0.446	1	5315	0.533960
2	449,721	1.26	2	1419	1.357294
3	188,933	2.24	3	642	2.373832
4	105,668	3.24	4	381	4.081365
5	68,379	4.22	5	311	3.781350
6	48,190	5.19	6	196	4.500000

Complications

- In practice, assume large counts (c>k for some k) are reliable:

$$c^* = c \text{ for } c > k$$

- That complicates c*, making it:

$$c^* = \frac{(c+1)\frac{N_{c+1}}{N_c} - c\frac{(k+1)N_{k+1}}{N_1}}{1 - \frac{(k+1)N_{k+1}}{N_1}}, \text{ for } 1 \leq c \leq k.$$

- Also: we assume singleton counts c=1 are unreliable, so treat N-grams with count of 1 as if they were count=0
- Also, need the Nk to be non-zero, so we need to smooth (interpolate) the Nk counts before computing c* from them

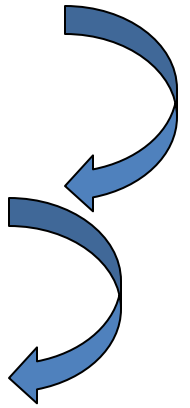
GT smoothed bigram probs

	i	want	to	eat	chinese	food	lunch	spend
i	0.0014	0.326	0.00248	0.00355	0.000205	0.0017	0.00073	0.000489
want	0.00134	0.00152	0.656	0.000483	0.00455	0.00455	0.00384	0.000483
to	0.000512	0.00152	0.00165	0.284	0.000512	0.0017	0.00175	0.0873
eat	0.00101	0.00152	0.00166	0.00189	0.0214	0.00166	0.0563	0.000585
chinese	0.00283	0.00152	0.00248	0.00189	0.000205	0.519	0.00283	0.000585
food	0.0137	0.00152	0.0137	0.00189	0.000409	0.00366	0.00073	0.000585
lunch	0.00363	0.00152	0.00248	0.00189	0.000205	0.00131	0.00073	0.000585
spend	0.00161	0.00152	0.00161	0.00189	0.000205	0.0017	0.00073	0.000585

Backoff and Interpolation

- Don’t try to account for unseen n-grams, just backoff to a simpler model until you’ve seen it.

- Start with estimating the trigram: P(z | x, y)
  - but C(x,y,z) is zero!
- Backoff and use info from the bigram: P(z | y)
  - but C(y,z) is zero!
- Backoff to the unigram: P(z)



- How to combine the trigram/bigram/unigram info?

Backoff versus interpolation

- **Backoff**: use trigram if you have it, otherwise bigram, otherwise unigram

- **Interpolation**: always mix all three

Interpolation

- Simple interpolation

$$\begin{aligned}\hat{P}(w_n|w_{n-1}w_{n-2}) &= \lambda_1 P(w_n|w_{n-1}w_{n-2}) \\ &\quad +\lambda_2 P(w_n|w_{n-1}) \\ &\quad +\lambda_3 P(w_n)\end{aligned} \qquad \sum_i \lambda_i = 1$$

- Lambdas conditional on context:

$$\begin{aligned}\hat{P}(w_n|w_{n-2}w_{n-1}) &= \lambda_1(w_{n-2}^{n-1}) P(w_n|w_{n-2}w_{n-1}) \\ &\quad +\lambda_2(w_{n-2}^{n-1}) P(w_n|w_{n-1}) \\ &\quad +\lambda_3(w_{n-2}^{n-1}) P(w_n)\end{aligned}$$

How to set the lambdas?

- Use a **held-out** corpus
- Choose lambdas which maximize the probability of some held-out data
  - I.e. fix the N-gram probabilities
  - Then search for lambda values
  - That when plugged into previous equation
  - Give largest probability for held-out set

Katz Backoff

- Use the trigram probabilty if the trigram was observed:
  - P(dog | the, black) if C(“the black dog”) > 0
- “Backoff” to the bigram if it was unobserved:
  - P(dog | black) if C(“black dog”) > 0
- “Backoff” again to unigram if necessary:
  - P(dog)

Katz Backoff

- Gotcha: You can't just backoff to the shorter n-gram.
- Why not? It is no longer a probability distribution. The entire model must sum to one.
  - The individual trigram and bigram distributions are valid, but we can't just combine them.

- Each distribution now needs a factor. See the book for details.
  - P(dog|the,black) = alpha(dog,black) * P(dog | black)