CS 584 Natural Language Processing

Language Modeling

Department of Computer Science
Yue Ning
yue.ning@stevens.edu

Late Submission Policy

- **10%** penalty for late submission within **<u>24 hours</u>**.
- **40%** penalty for late submissions within **<u>24-48 hours</u>**.
- After 48 hours, you get **NO** points on the assignment.

Probabilistic Language Modeling

Goal: assign a probability to a sentence

- Machine Translation:
  - P(high winds tonite) > P(large winds tonite)
- Spell Correction
  - The office is about fifteen minuets from my house
    - P(about fifteen minutes from) > P(about fifteen minuets from)
- Speech Recognition
  - P(I saw a van) >> P(eyes awe of an)
- + Summarization, question-answering, etc., etc.!

Probabilistic Language Modeling

- Goal: compute the probability of a sentence or a sequence of words:
  - $P(W) = P(w_1,w_2,w_3,w_4,w_5\ldots w_n)$
- Related task: probability of an upcoming word:
  - $P(w_5|w_1,w_2,w_3,w_4)$
- A model that computes either of these:
  - $P(W)$ or $P(w_n|w_1,w_2\ldots w_{n-1})$ is called a language model.
- Better: the grammar But language model or LM is standard

How to compute P(W)

- How to compute this joint probability:
  - P(its, water, is, so, transparent, that)

- Intuition: let's rely on the Chain Rule of Probability

Reminder: the chain rule

- Recall the definition of conditional probability

- More variables:
  - $P(A,B,C,D) = P(A)P(B|A)P(C|A,B)P(D|A,B,C)$
- The Chain Rule in General
  - $P(x_1,x_2,x_3,\ldots,x_n) = P(x_1)P(x_2|x_1)P(x_3|x_1,x_2)\ldots P(x_n|x_1,\ldots,x_{n-1})$

The chain rule for joint probability

$$P(w_1 w_2 \ldots w_n) = \prod_i P(w_i \mid w_1 w_2 \ldots w_{i-1})$$

P("its water is so transparent") =

P(its) × P(water|its) × P(is|its water)

× P(so|its water is) × P(transparent|its water is so)

How to estimate these probabilities

Can we just count and divide?

P(“the” | “its water is so transparent that”) =

$$\frac{\text{count (its water is so transparent that the)}}{\text{count (its water is so transparent that)}}$$

No! Too many possible sentences!

We’ll never see enough data for estimating these.

Markov Assumption

- The probability of moving to the next state depends only on the present state and not on the previous states.
- Simplifying assumption:

P(“the” | “its water is so transparent that”) = P(“the”|”that”)

- Or maybe:

P(“the” | “its water is so transparent”) = P(“the”|“transparent that”)

N-gram language models

- Q: How to learn a language model?
- A: Learn an n-gram language model.
- Idea: collect statistics about how frequent different n-grams are and use these to predict next word.

$$P(w_1 w_2 ... w_n) \approx \prod_i P(w_i | w_{i-k} ... w_{i-1})$$

- In other words, we approximate each component in the product

$$P(w_i | w_1 ... w_{i-1}) \approx P(w_i | w_{i-k} ... w_{i-1})$$

https://web.stanford.edu/~jurafsky/slp3/3.pdf

Simplest case: unigram model

$$P(w_1 ... w_n) \approx \prod_i P(w_i)$$

- Some automatically generated from a unigram model:

  fifth, an, of, futures, the, an, incorporated, a, a, the, inflation, most, dollars, quarter, in, is, mass

  thrift, did, eighty, said, hard, 'm, july, bullish'

  that, or, limited, the

Bigram model

- Condition on the previous word:

$$P(w_1, ...w_n) \approx \prod_i P(w_i|w_{i-1})$$

- Some automatically generated from a bigram model:

texaco, rose, one, in, this, issue, is, pursuing, growth, in, a, boiler, house, said, mr., gurria, mexico, 's, motion, control, proposal, without, permission, from, five, hundred, fifty, five, yen

outside, new, car, parking, lot, of, the, agreement, reached

this, would, be, a, record, november

N-gram model

- We can extend to trigrams, 4-grams, 5-grams

- In general this is an insufficient model of language
  - language has long-distance dependencies:
    - "*The computer which I had just put into the machine room on the fifth floor crashed*"

- But we can often get away with N-gram models

Estimating bigram probabilities

- The Maximum Likelihood Estimate

$$P(w_i|w_{i-1}) = \frac{\text{count}(w_{i-1},w_i)}{\text{count}(w_{i-1})}$$

$$P(w_i|w_{i-1}) = \frac{c(w_{i-1},w_i)}{c(w_{i-1})}$$

An example

- <s> I am Emily </s>
- <s> Emily I am </s>
- <s> I do not like green beans and ham </s>

$$P(w_i|w_{i-1}) = \frac{c(w_{i-1},w_i)}{c(w_{i-1})}$$

```
P(I | <s>) = ⅔ = 0.67
P(Emily | <s>) = ⅓ = 0.33
P(am | I ) = ⅔=0.67
P(</s> | Emily)= ½=0.5
P(Emily | am) = ½ =0.5
P(do | I) = ⅓=0.33
```

The Berkeley Restaurant Project

https://github.com/wooters/berp-trans

- i want to find a really nice place to eat lunch next wednesday
- i want to eat either french food california food or italian food
- i have a car so the distance is not important
- i would like to eat american food or mexican food

Raw bigram counts

- out of 9222 sentences:

	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

Raw bigram probability

- Normalize by unigrams:

i	want	to	eat	chinese	food	lunch	spend
2533	927	2417	746	158	1093	341	278

- Results:

	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

Bigram estimates of sentence probabilities

- P (<s> I want english food </s>) =
  P(I | <s>)
  x P(want | I)
  x P(english | want)
  x P(food | english)
  x P(</s> | food)
  = 0.000031

Bigram probabilities

- P (english | want) = 0.0011
- P(chinese | want) = 0.0065
- P(to | want) = 0.66
- P(eat | to) = 0.28
- P(food | to) = 0
- P(want | spend) = 0
- P(i | <s>) = 0.25

Practical Issues

- We do everything in log space
  - Avoid underflow
  - also adding is faster than multiplying
- $\log (p_1 \times p_2 \times p_3 \times p_4) = \log p_1 + \log p_2 + \log p_3 + \log p_4$

Language Modeling Toolkits

- SRILM - The SRI Language Modeling Toolkit
  - http://www.speech.cs.cmu.edu/SLM/toolkit_documentation.html

- The CMU-Cambridge Statistical Language Modeling Toolkit v2
  - http://www.speech.cs.cmu.edu/SLM/toolkit_documentation.html

Google N-gram release (2006)

- All Our N-gram are belong to you
  - “We processed 1,024,908,267,229 words of running text and are publishing the counts for all 1,176,470,663 five-word sequences that appear at least 40 times. There are 13,588,391 unique words, after discarding words that appear less than 200 times.”

Google N-gram viewer

- https://books.google.com/ngrams

Sparsity problems in N-grams

Problem: what if "its water is so transparent that w" never occurred in data? Then w has probability 0!

(Partial) solution: add small value to the count for every w in the vocabulary. This is called smoothing.

$$P(w \mid \text{“its water is so transparent that”}) = \frac{\text{count (its water is so transparent that w)}}{\text{count (its water is so transparent that)}}$$

Problem: what if "its water is so transparent that" never occurred in data? Then we cannot calculate probability for any w.

(Partial) solution: Just condition on "so transparent that" instead. This is called *backoff*.

Note: increasing n makes sparsity problems worse. Typically we cannot have n bigger than 5.

Storage problems with n-gram

Storage: Need to store count for all n-grams you saw in the corpus

$$P(w \mid \text{“its water is so transparent that”}) = \frac{\text{count (its water is so transparent that w)}}{\text{count (its water is so transparent that)}}$$

Note: increasing n or increasing corpus makes model size bigger!

Generating text with an n-gram LM

You can use a language model to generate text:

today the ___

Condition on this

Get probability distribution

company	0.153
bank	0.153
price	0.077
italian	0.039
emirate	0.039
...	

sample

Generating text with an n-gram LM

You can use a language model to generate text:

today the price ____

Condition on this

Get probability distribution

of	0.308
for	0.050
it	0.046
to	0.046
is	0.031
...	

sample

Generating text with an n-gram LM

You can use a language model to generate text:

today the price of gold per ton , while production of shoe lasts and shoe industry , the bank intervened just after it considered and rejected an imf demand to rebuild depleted european stocks , sept 30 end primary 76 cts a share.

surprisingly grammatical! but incoherent.

Evaluation: how good is our model?

- Does our language model prefer good sentences to bad ones?
  - Assign higher probability to “real” or “frequently observed” sentences than “ungrammatical” or “rarely observed” sentences?
- We train parameters of our model on a training set.
- We test the model’s performance on data we haven’t seen.
  - A test set is an unseen dataset that is different from our training set, totally unused.
  - An evaluation metric tells us how well our model does on the test set.

Extrinsic evaluation of N-gram models

- Best evaluation for comparing models A and B
  - Put each model in a task
    - spelling corrector, speech recognizer, MT system
  - Run the task, get an accuracy for A and for B
    - How many misspelled words corrected properly
    - How many words translated correctly
  - Compare accuracy for A and B

Difficulty of extrinsic evaluation

- Extrinsic evaluation
  - Time-consuming; can take days or weeks
- So
  - Sometimes use intrinsic evaluation: perplexity
  - Bad approximation
    - unless the test data looks just like the training data
    - So generally only useful in pilot experiments
  - But is helpful to think about.

Intuition of Perplexity

- The Shannon Game:
  - How well can we predict the next word?

I always order pizza with cheese and ____

The 33rd President of the US was ____

I saw a ____

mushrooms 0.1

pepperoni 0.1

anchovies 0.01

....

fried rice 0.0001

....

and 1e-100

  - Unigram are terrible at this game (why)?
- A better model of a text
  - is one which assigns a higher probability to the word that actually occurs.

Perplexity

- The best language model is one that best predicts an unseen test set
  - Gives the highest P (sentence)

$$P(W) = P(w_1 w_2 ... w_N)^{-\frac{1}{N}}$$

$$= \sqrt[N]{\frac{1}{P(w_1 w_2 ... w_N)}}$$

Chain rule:

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

For bigrams:

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

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

Perplexity

- From Josh Goodman
  - How hard is the task of recognizing digits ‘0,1,2,3,4,5,6,7,8,9’
    - Perplexity 10
  - How hard is recognizing (30,000) names at Microsoft.
    - Perplexity = 30,000
- If a system has to recognize a switch-board phone operator
  - Word “Operator” (1 in 4)
  - Word “Sales” (1 in 4)
  - Word "Technical Support” (1 in 4)
  - 30,000 names (1 in 120,000 each)
  - Perplexity is 53
- Perplexity is weighted equivalent branching factor

Perplexity as a branching factor

- Let's suppose a sentence consisting of random digits
- What is the perplexity of this sentence according to a model that assign P = 1/10 to each digit?

$$
\begin{aligned}
P(W) &= P(w_1 w_2 ... w_N)^{-\frac{1}{N}} \\
&= (\frac{1}{10}^{N})^{-\frac{1}{N}} \\
&= \frac{1}{10}^{-1} \\
&= 10
\end{aligned}
$$

Low Perplexity = better model

- Training 38 millions words, test 1.5 million words (WSJ)

N-gram Order	Unigram	Bigram	Trigram
Perplexity	962	170	109

The Shannon Visualization Method

- Choose a random bigram (<s>, w) according to its probability
- Now choose a random bigram(w, x) according to its probability
- And so on until we choose </s>
- Then string the words together

```
<s> I
    I want
      want to
           to eat
              eat Chinese
                  Chinese food
                          food  </s>
I want to eat Chinese food
```

Approximating Shakespeare

Unigram

To him swallowed confess hear both. Which. Of save on trail for are ay device and rote life have
Every enter now severally so, let
Hill he late speaks; or! a more to leg less first you enter
Are where exeunt and sighs have rise excellency took of.. Sleep knave we. near; vile like

Bigram

What means, sir. I confess she? then all sorts, he is trim, captain.
Why dost stand forth thy canopy, forsooth; he is this palpable hit the King Henry. Live king. Follow.
What we, hath got so she that I rest and sent to scold and nature bankrupt, nor the first gentleman?

Trigram

Sweet prince, Falstaff shall die. Harry of Monmouth's grave.
This shall forbid it should be branded, if renown made it empty.
Indeed the duke; and had a very good friend.
Fly, and will rid me these news of price. Therefore the sadness of parting, as they say, 'tis done.

Quadrigram

King Henry.What! I will go seek the traitor Gloucester. Exeunt some of the watch. A great banquet serv'd in;
Will you not tell me who I am?
It cannot be but so.
Indeed the short and the long. Marry, 'tis a noble Lepidus.

Shakespeare as corpus

- N=884,647 tokens, V=29,066
- Shakespeare produced 300,000 bigram types out of V2= 844 million possible bigrams.
    - So 99.96% of the possible bigrams were never seen (have zero entries in the table)
- Quadrigrams worse: What's coming out looks like Shakespeare because it is Shakespeare

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 that generalize!
    - One kind of generalization: Zeros!
        - Things that don't ever occur in the training set
        - But occur in the test set

Zeros

- Training set:

... denied the allegations
... denied the reports
... denied the claims
... denied the request
P(“offer” | denied the) = 0

- Test set:

... denied the offer
... denied the loan

Zero probability bigrams

- Bigram with zero probability
  - mean that we will assign 0 probability to the test set
- Hence we cannot compute perplexity (divide by 0!)

The intuition of smoothing

- When we have sparse statistics
  - P(w | “denied the”)
  - 3 allegations
  - 2 reports
  - 1 claims
  - 1 request
  - 7 total

- Steal probability mass to generalize better
  - P(w | “denied the”)
  - 2.5 allegations
  - 1.5 reports
  - 0.5 claims
  - 0.5 request
  - 2 others
  - 7 total

Add-one estimation

- Also called Laplace smoothing
- Pretend we saw each word one more time that we did
- Just add one to all the counts
- MLE estimate:

$$P_{\text{MLE}}(w_i|w_{i-1}) = \frac{c(w_{i-1}, w_i)}{c(w_{i-1})}$$

- Add-1 estimate:

$$P_{\text{Add-1}}(w_i|w_{i-1}) = \frac{c(w_{i-1}, w_i) + 1}{c(w_{i-1}) + V}$$

Maximum Likelihood Estimates

- The maximum likelihood estimate
  - of some parameter of a model M from a training set T
  - maximizes the likelihood of the training set T given the model M
- Suppose the word “bagel” occurs 400 times in a corpus of a million words
- What is the probability that a random word from some other text will be “bagel”?
- MLE estimate is 400/1,000,000 = .0004
- This may be a bad estimate for some other corpus
  - But it is the estimate that makes it most likely that “bagel” will occur 400 times in a million word corpus.

Berkeley Restaurant Corpus: 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

$$P^*(w_i|w_{i-1}) = \frac{c(w_{i-1}, w_i) + 1}{c(w_{i-1}) + V}$$

	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

$$c^*(w_{i-1}w_i) = \frac{[c(w_{i-1}, w_i) + 1] \times c(w_{i-1})}{c(w_{i-1}) + V}$$

	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

Compare with raw 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

	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

Add-1 estimation is a blunt instrument

- Sometimes it helps to use less context
  - Condition on less context for contexts you haven't learned much about
- Backoff:
  - use trigram if you have good evidence,
  - otherwise bigram, otherwise unigram
- Interpolation: mix unigram, bigram, trigram
- Interpolation works better

Backoff and Interpolation

- So add-1 isn't used for N-grams:
  - we'll see better methods
- But add-1 is used to smooth other NLP models
  - for text classification
  - in domains where the number of zeros isn't so huge

Linear Interpolation

- Simple interpolation $$\hat{P}(w_n|w_{n-1}w_{n-2}) = \lambda_1 P(w_n|w_{n-1}w_{n-2}) + \lambda_2 P(w_n|w_{n-1}) + \lambda_3 P(w_n)$$
- Lambdas conditional on context: $$\sum_i \lambda_i = 1$$

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

Now lambdas are dependent on what the previous two words were.

How to set the lambdas?

- Use a held-out corpus

- Choose lambdas to maximize the probability of held-out data:
  - fix the N-gram probabilities (on the training data)
  - then search for lambdas that give largest probability to held-out set:

$$\log P(w_1 ... w_n | M(\lambda_1 ... \lambda_k)) = \sum_i \log P_{M(\lambda_1 ... \lambda_k)}(w_i | w_{i-1})$$

Unknown words:
open versus closed vocabulary task

- If we know all the words in advance
  - Vocabulary V is fixed
  - Closed vocabulary task
- Often we don't know this
  - Out Of Vocabulary = OOV words
  - Open vocabulary task
- Instead: create an unknown word token <UNK>
  - Training of <UNK> probabilities
    - Create a fixed lexicon L of size V
    - At text normalization phase, any training word not in L changed to <UNK>
    - Now we train its probabilities like a normal word
  - At decoding time
    - If text input: Use UNK probabilities for any word not in training

Huge web-scale n-grams

- How to deal with, e.g., Google N-gram corpus
- Pruning
  - Only store N-grams with count > threshold.
    - Remove singletons of higher-order n-grams
  - Entropy-based pruning

- Efficiency
  - Efficient data structures like tries
  - Bloom filters: approximate language models
  - Store words as indexes, not strings
    - Use Huffman coding to fit large numbers of words into two bytes
  - Quantize probabilities (4-8 bits instead of 8-byte float)

Smoothing for web-scale N-grams

- "Stupid backoff" (Brants *et al*. 2007)
- No discounting, just use relative frequencies

$$S(w_i \mid w_{i-k+1}^{i-1}) = \begin{cases} \dfrac{\text{count}(w_{i-k+1}^{i})}{\text{count}(w_{i-k+1}^{i-1})} & \text{if} \quad \text{count}(w_{i-k+1}^{i}) > 0 \\ 0.4 S(w_i \mid w_{i-k+2}^{i-1}) & \text{otherwise} \end{cases}$$

The recursion ends at unigrams

$$S(w_i) = \frac{\text{count}(w_i)}{N}$$

N-gram smoothing summary

- Add-1 smoothing:
  - OK for text categorization, not for language modeling
- The most commonly used method:
  - Extended Interpolated Kneser-Ney
- For very large N-grams like the Web:
  - Stupid backoff

Advanced language modeling

- Discriminative models:
  - choose n-gram weights to improve a task, not to fit the training set
- Parsing-based models
- Caching Models
  - Recently used words are more likely to appear

$$P_{CACHE}(w \mid history) = \lambda P(w_i \mid w_{i-2} w_{i-1}) + (1-\lambda)\frac{c(w \in history)}{|history|}$$

  - These perform very poorly for speech recognition (why?)

Reminder: add-1 (laplace) smoothing

$$P_{\text{Add}-1}(w_i|w_{i-1}) = \frac{c(w_{i-1}, w_i) + 1}{c(w_{i-1}) + V}$$

More general formulations: add-k

$$P_{add-k}(w_i|w_{i-1}) = \frac{c(w_{i-1},, w_i) + k}{c(w_{i-1}) + kV}$$

$$P_{add-k}(w_i|w_{i-1}) = \frac{c(w_{i-1},, w_i) + m(\frac{1}{V})}{c(w_{i-1}) + m}$$

Unigram prior smoothing

$$P_{add-k}(w_i|w_{i-1}) = \frac{c(w_{i-1},, w_i) + m(\frac{1}{V})}{c(w_{i-1}) + m}$$

$$P_{\text{unigramprior}}(w_i|w_{i-1}) = \frac{c(w_{i-1},, w_i) + mP(w_i)}{c(w_{i-1}) + m}$$

Advanced smoothing algorithms

- Intuition used by many smoothing algorithms
  - Good-Turing
  - Kneser-Ney
  - Witten-Bell

- Use the count of things we've seen once
  - to help estimate the count of things we've never seen

Frequency of frequency c

- N_c = the count of things we've seen c times
  - Emily I am I am Emily I do not eat

I	3
Emily	2
am	2
do	1
not	1
eat	1

$N_1 = 3$

$N_2 = 2$

$N_3 = 1$

Good Turing estimation

- Idea: reallocate the probability mass of n-grams that occur c+1 times in the training data to the n-grams that occur c times
- In particular, reallocate the probability mass of n-grams that were seen once to the n-grams that were never seen,
- For each count c, we compute an adjusted count c^*:

$$c^* = \frac{(c+1)N_{c+1}}{N_c}$$

where N_c is the number of n-grams seen exactly c times.

- Then we have:

$$P^*_{GT} = c^* / N$$

where $N = \sum_{r=1}^{\infty} cN_c$

Good Turing smoothing intuition

- You are fishing and caught:
  - 10 carps, 3 perch, 2 whitefish, 1 trout, 1 salmon, 1 eel = 18 fish
- How likely is it that the next species is trout?
  - 1/18
- How likely is it that the next species is new (i.e. catfish or bass)?
  - use our estimate of things-we-saw-once to estimate the news things: 3/18 (N_1=3)
- Assuming so, how likely is it that the next species is trout?
  - must be less that 1/18, how to estimate?

Good Turing calculations

- P^*_{GT} (things with zero frequency) = N_1/N
- Other frequencies: $c^* = \frac{(c+1)N_{c+1}}{N_c}$

- Unseen (bass or catfish)
- c=0
- MLE_p= 0/18 = 0
- P^*_{GT} (unseen) = N_1/N = 3/18

- Seen once (trount)
- c = 1,
- MLE_p=1/18
- c^* (trout) = 2* N_2/N_1 = 2*⅓ = ⅔
- P^*_{GT} (trout) = 2/3/18= 1/27

Ney et al. Good Turing Intuition

- Intuition from leave-one-out validation
  - Take each of the c training words out in turn
  - c training sets of size c–1, held-out of size 1
  - What fraction of held-out words are unseen in training?
    - N_1/c
  - What fraction of held-out words are seen k times in training?
    - $(k+1)N_{k+1}/c$
  - So in the future we expect $(k+1)N_{k+1}/c$ of the words to be those with training count k
  - There are N_k words with training count k
  - Each should occur with probability:
    - $(k+1)N_{k+1}/c/N_k$
  - …or expected count: $k^* = \frac{(k+1)N_{k+1}}{N_k}$

Good-Turing complications

- Problem: what about “the”? (say c=4417)
  - For small k, $N_k > N_{k+1}$
  - For large k, too jumpy, zeros wreck estimates
  - Simple Good-Turing [Gale and Sampson]: replace empirical N_k with a best-fit power law once counts get unreliable

Resulting Good-Turing numbers

- Numbers from Church and Gale(1991)
- 22 million words of AP Newswire

$$c^* = \frac{(c+1)N_{c+1}}{N_c}$$

- It sure looks like c* = (c- .75)

Count c	Good Turing c*
0	.0000270
1	0.446
2	1.26
3	2.24
4	3.24
5	4.22
6	5.19
7	6.21
8	7.24
9	8.25

Absolute Discounting Interpolation

- Same ourselves some time and just subtract 0.75 (or some d)!

discounted bigram

Interpolation weight

$$P_{\text{AbsoluteDiscounting}}(w_i \mid w_{i-1}) = \frac{c(w_{i-1}, w_i) - d}{c(w_{i-1})} + \lambda(w_{i-1})P(w)$$

unigram

- Maybe keeping a couple extra values of d from counts 1 and 2 (last slide)
- But should we really just use the regular unigram P(w)?

Kneser-Ney Smoothing I

- Better estimate for probabilities of lower-order unigrams!
  - Shannon game: I can't see without my reading_____?
  - "Francisco" is more common than "glasses"
  - … but "Francisco" always follows "San"
- The unigram is useful exactly when we haven't seen this bigram!
- Instead of P(w): "How likely is w"
- $P_{continuation(}w)$: "How likely is w to appear as a novel continuation?
  - For each word, count the number of bigram types it completes
  - Every bigram type was a novel continuation the first time it was seen:

$$P_{CONTINUATION}(w) \propto \left|\{w_{i-1} : c(w_{i-1}, w) > 0\}\right|$$

Kneser-Ney Smoothing II

- How many times does w appear as a novel continuation:

$$P_{CONTINUATION}(w) \propto \left|\{w_{i-1} : c(w_{i-1}, w) > 0\}\right|$$

- Normalized by the total number of word bigram types

$$\left|\{(w_{j-1}, w_j) : c(w_{j-1}, w_j) > 0\}\right|$$

$$P_{CONTINUATION}(w) = \frac{\left|\{w_{i-1} : c(w_{i-1}, w) > 0\}\right|}{\left|\{(w_{j-1}, w_j) : c(w_{j-1}, w_j) > 0\}\right|}$$

Kneser-Ney Smoothing III

- Alternative metaphor: The number (#) of word types seen to precede w

$$|\{w_{i-1} : c(w_{i-1}, w) > 0\}|$$

- Normalized by the # of words preceding all words:

$$P_{CONTINUATION}(w) = \frac{|\{w_{i-1} : c(w_{i-1}, w) > 0\}|}{\sum_{w'} |\{w'_{i-1} : c(w'_{i-1}, w') > 0\}|}$$

- A frequent word (Francisco) occurring in only one context (San) will have a low continuation probability

Kneser-Ney Smoothing IV

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

- lambda is a normalizing constant; the probability mass we've discounted

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

the normalized discount

The number of word types that can follow w_{i-1}
= # of word types we discounted
= # of times we applied normalized discount

Kneser-Ney Smoothing Recursive formulation

$$P_{KN}(w_i \mid w_{i-n+1}^{i-1}) = \frac{\max(c_{KN}(w_{i-n+1}^{i}) - d, 0)}{c_{KN}(w_{i-n+1}^{i-1})} + \lambda(w_{i-n+1}^{i-1}) P_{KN}(w_i \mid w_{i-n+2}^{i-1})$$

$$c_{KN}(\bullet) = \begin{cases} count(\bullet) & \text{for the highest order} \\ continuationcount(\bullet) & \text{for lower order} \end{cases}$$

Continuation count = Number of unique single word contexts for ($\bullet$)

Acknowledgements

Slides adapted from Dr. Dan Jurafsky's Introduction to Natural Language Processing at Stanford.

STEVENS
INSTITUTE of TECHNOLOGY
THE INNOVATION UNIVERSITY®

stevens.edu

Thank You