Smoothing N-gram Language Models

Shallow Processing Techniques for NLP
Ling570
October 24, 2011

Roadmap

- Comparing N-gram Models
- Managing Sparse Data: Smoothing
  - Add-one smoothing
  - Good-Turing Smoothing
  - Interpolation
  - Backoff
- Extended N-gram Models
  - Class-based n-grams
  - Long distance models

Perplexity Model Comparison

- Compare models with different history
- Train models
  - 38 million words – Wall Street Journal

Perplexity Model Comparison

- Compare models with different history
- Train models
  - 38 million words – Wall Street Journal
- Compute perplexity on held-out test set
  - 1.5 million words (~20K unique, smoothed)

Perplexity Model Comparison

- Compare models with different history
- Train models
  - 38 million words – Wall Street Journal
- Compute perplexity on held-out test set
  - 1.5 million words (~20K unique, smoothed)
- N-gram Order | Perplexity
  - Unigram | 962
  - Bigram | 170
  - Trigram | 109

Smoothing

Problem: Sparse Data

- Goal: Accurate estimates of probabilities
- Current maximum likelihood estimates
  - E.g. $P(w_i \mid w_{i=1}) = \frac{C(w_{i-1}w_i)}{C(w_{i-1})}$
  - Work fairly well for frequent events

Problem: Sparse Data

- Goal: Accurate estimates of probabilities
- Current maximum likelihood estimates
  - E.g. $P(w_i \mid w_{i=1}) = \frac{C(w_{i-1}w_i)}{C(w_{i-1})}$
  - Work fairly well for frequent events
- Problem: Corpora limited
  - Zero count events (event = ngram)

Problem: Sparse Data

- Goal: Accurate estimates of probabilities
- Current maximum likelihood estimates
  - E.g. $$P(w_i \mid w_{i=1}) = \frac{C(w_{i-1}w_i)}{C(w_{i-1})}$$
  - Work fairly well for frequent events
- Problem: Corpora limited
  - Zero count events (event = ngram)
- Approach: “Smoothing”
  - Shave some probability mass from higher counts to put on (erroneous) zero counts

How much of a problem is it?

- Consider Shakespeare
  - Shakespeare produced 300,000 bigram types out of V^2= 844 million possible bigrams...

How much of a problem is it?

- Consider Shakespeare
  - Shakespeare produced 300,000 bigram types out of V^2= 844 million possible bigrams...
  - So, 99.96% of the possible bigrams were never seen (have zero entries in the table)

How much of a problem is it?

- Consider Shakespeare
  - Shakespeare produced 300,000 bigram types out of V^2= 844 million possible bigrams...
  - So, 99.96% of the possible bigrams were never seen (have zero entries in the table)
  - Does that mean that any sentence that contains one of those bigrams should have a probability of 0?

What are Zero Counts?

- Some of those zeros are really zeros…
  - Things that really can't or shouldn't happen.

What are Zero Counts?

- Some of those zeros are really zeros...
  - Things that really can't or shouldn't happen.
- On the other hand, some of them are just rare events.
  - If the training corpus had been a little bigger they would have had a count (probably a count of 1!).

What are Zero Counts?

- Some of those zeros are really zeros...
  - Things that really can't or shouldn't happen.
- On the other hand, some of them are just rare events.
  - If the training corpus had been a little bigger they would have had a count (probably a count of 1!).
- Zipf's Law (long tail phenomenon):
  - A small number of events occur with high frequency
  - A large number of events occur with low frequency
  - You can quickly collect statistics on the high frequency events
  - You might have to wait an arbitrarily long time to get valid statistics on low frequency events

Laplace Smoothing

- Add 1 to all counts (aka Add-one Smoothing)

Laplace Smoothing

- Add 1 to all counts (aka Add-one Smoothing)
- V: size of vocabulary; N: size of corpus
- Unigram: P_{MLE}

Laplace Smoothing

- Add 1 to all counts (aka Add-one Smoothing)
- V: size of vocabulary; N: size of corpus
- Unigram: P_{MLE}: $P(w_i) = C(w_i)/N$
- $P_{Laplace}(w_i) =$

Laplace Smoothing

- Add 1 to all counts (aka Add-one Smoothing)
- V: size of vocabulary; N: size of corpus
- Unigram: P_{MLE}: $P(w_i) = C(w_i)/N$
- $P_{Laplace}(w_i) = \frac{C(w_i)+1}{N+V}$
- Bigram: $P_{laplace}(w_i | w_{i-1}) =$
-

Laplace Smoothing

- Add 1 to all counts (aka Add-one Smoothing)
- V: size of vocabulary; N: size of corpus
- Unigram: P_{MLE}: $P(w_i) = C(w_i)/N$
- $$P_{Laplace}(w_i) = \frac{C(w_i)+1}{N+V}$$
- Bigram: $$P_{laplace}(w_i \mid w_{i-1}) = \frac{C(w_{i-1}w_i)+1}{C(w_{i-1})+V}$$
- n-gram: $P_{laplace}(w_i \mid w_{i-1} \cdots w_{i-n+1}) =$.

Laplace Smoothing

- Add 1 to all counts (aka Add-one Smoothing)
- V: size of vocabulary; N: size of corpus
- Unigram: P_{MLE}: $P(w_i) = C(w_i)/N$
- $$P_{Laplace}(w_i) = \frac{C(w_i)+1}{N+V}$$
- Bigram: $$P_{laplace}(w_i \mid w_{i-1}) = \frac{C(w_{i-1}w_i)+1}{C(w_{i-1})+V}$$
- n-gram: $$P_{laplace}(w_i \mid w_{i-1}\cdots w_{i-n+1}) = \frac{C(w_{i-n+1}\ldots w_{i-1}w_i)+1}{C(w_{i-n+1}w_{i-1})+V}$$

BERP Corpus Bigrams

- Original bigram probabilites

	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

BERP Smoothed Bigrams

- Smoothed bigram probabilities from the BERP

	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

Laplace Smoothing Example

- Consider the case where |V|= 100K
  - C(Bigram w_1w_2) = 10; C(Trigram $w_1w_2w_3$) = 9

Laplace Smoothing Example

- Consider the case where |V|= 100K
  - C(Bigram w_1w_2) = 10; C(Trigram $w_1w_2w_3$) = 9
- P_{MLE}=

Laplace Smoothing Example

- Consider the case where |V|= 100K
  - C(Bigram w_1w_2) = 10; C(Trigram $w_1w_2w_3$) = 9
- P_{MLE}=9/10 = 0.9
- P_{LAP}=

Laplace Smoothing Example

- Consider the case where |V|= 100K
  - C(Bigram w_1w_2) = 10; C(Trigram $w_1w_2w_3$) = 9
- P_{MLE}=9/10 = 0.9
- P_{LAP}=(9+1)/10+100K ~ 0.0001

Laplace Smoothing Example

- Consider the case where |V|= 100K
  - C(Bigram w_1w_2) = 10; C(Trigram $w_1w_2w_3$) = 9
- P_{MLE}=9/10 = 0.9
- P_{LAP}=(9+1)/10+100K ~ 0.0001
- Too much probability mass ‘shaved off’ for zeroes
- Too sharp a change in probabilities
  - Problematic in practice

Add-δ Smoothing

- Problem: Adding 1 moves too much probability mass

Add-δ Smoothing

- Problem: Adding 1 moves too much probability mass
- Proposal: Add smaller fractional mass δ
- $P_{add\text{-}\delta}(w_i|w_{i-1})$

Add-δ Smoothing

- Problem: Adding 1 moves too much probability mass
- Proposal: Add smaller fractional mass δ
- $P_{add\text{-}\delta}(w_i \mid w_{i-1}) = \frac{C(w_{i-1}w_i)+\delta}{C(w_{i-1})+V\delta}$

- Issues:

Add-δ Smoothing

- Problem: Adding 1 moves too much probability mass
- Proposal: Add smaller fractional mass δ
- $P_{add\text{-}\delta}(w_i \mid w_{i-1}) = \frac{C(w_{i-1}w_i)+\delta}{C(w_{i-1})+V\delta}$

- Issues:
  - Need to pick δ
  - Still performs badly

Good-Turing Smoothing

- New idea: Use counts of things you have seen to estimate those you haven't

Good-Turing Smoothing

- New idea: Use counts of things you have seen to estimate those you haven't
- Good-Turing approach: Use frequency of singletons to re-estimate frequency of zero-count n-grams

Good-Turing Smoothing

- New idea: Use counts of things you have seen to estimate those you haven't
- Good-Turing approach: Use frequency of singletons to re-estimate frequency of zero-count n-grams
- Notation: N_c is the frequency of frequency c
  - Number of ngrams which appear c times
  - N_0: # ngrams of count 0; N_1: # of ngrams of count 1

$$N_c = \sum_{x:count(x)=c} 1$$

Good-Turing Smoothing

- Estimate probability of things which occur c times with the probability of things which occur c+1 times

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

$$p^*_{GT}(thingswithfreq0) = \frac{N_1}{N}$$

Good-Turing
Josh Goodman Intuition

- Imagine you are fishing
  - There are 8 species: carp, perch, whitefish, trout, salmon, eel, catfish, bass
- You have caught
  - 10 carp, 3 perch, 2 whitefish, 1 trout, 1 salmon, 1 eel = 18 fish
- How likely is it that the next fish caught is from a new species (one not seen in our previous catch)?

Slide adapted from Josh Goodman, Dan Jurafsky

Good-Turing
Josh Goodman Intuition

- Imagine you are fishing
  - There are 8 species: carp, perch, whitefish, trout, salmon, eel, catfish, bass
- You have caught
  - 10 carp, 3 perch, 2 whitefish, 1 trout, 1 salmon, 1 eel = 18 fish
- How likely is it that the next fish caught is from a new species (one not seen in our previous catch)?
  - 3/18
- Assuming so, how likely is it that next species is trout?

Slide adapted from Josh Goodman, Dan Jurafsky

Good-Turing
Josh Goodman Intuition

- Imagine you are fishing
  - There are 8 species: carp, perch, whitefish, trout, salmon, eel, catfish, bass
- You have caught
  - 10 carp, 3 perch, 2 whitefish, 1 trout, 1 salmon, 1 eel = 18 fish
- How likely is it that the next fish caught is from a new species (one not seen in our previous catch)?
  - 3/18
- Assuming so, how likely is it that next species is trout?
  - Must be less than 1/18

Slide adapted from Josh Goodman, Dan Jurafsky

GT Fish Example

	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

Good-Turing Smoothing

- N-gram counts to conditional probability
  - c^* from GT estimate

$$P_{GT}(w_i \mid w_1 \ldots w_{i-1}) = \frac{c^*(w_1 \ldots w_i)}{c^*(w_1 \ldots w_{i-1})}$$

Backoff and Interpolation

- Another really useful source of knowledge
- If we are estimating:
  - trigram p(z|x,y)
  - but count(xyz) is zero
- Use info from:

Backoff and Interpolation

- Another really useful source of knowledge
- If we are estimating:
  - trigram p(z|x,y)
  - but count(xyz) is zero
- Use info from:
  - Bigram p(z|y)
- Or even:
  - Unigram p(z)

Backoff and Interpolation

- Another really useful source of knowledge
- If we are estimating:
  - trigram p(z|x,y)
  - but count(xyz) is zero
- Use info from:
  - Bigram p(z|y)
- Or even:
  - Unigram p(z)
- How to combine this trigram, bigram, unigram info in a valid fashion?

Backoff Vs. Interpolation

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

Backoff Vs. Interpolation

- **Backoff**: use trigram if you have it, otherwise bigram, otherwise unigram
- **Interpolation**: always mix all three

Interpolation

- Simple interpolation

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

$$\sum_i \lambda_i = 1$$

Interpolation

- Simple interpolation

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

$$\sum_i \lambda_i = 1$$

- Lambdas conditional on context:
  - Intuition: Higher weight on more frequent n-grams

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

How to Set the Lambdas?

- Use a **held-out, or development,** 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
  - Can use EM to do this search

Katz Backoff

$$P_{\text{katz}}(w_n|w_{n-N+1}^{n-1}) = \begin{cases} P^*(w_n|w_{n-N+1}^{n-1}), & \text{if } C(w_{n-N+1}^{n}) > 0 \\ \alpha(w_{n-N+1}^{n-1}) P_{\text{katz}}(w_n|w_{n-N+2}^{n-1}), & \text{otherwise.} \end{cases}$$

$$P_{\text{katz}}(z|x,y) = \begin{cases} P^*(z|x,y), & \text{if } C(x,y,z) > 0 \\ \alpha(x,y) P_{\text{katz}}(z|y), & \text{else if } C(x,y) > 0 \\ P^*(z), & \text{otherwise.} \end{cases}$$

$$P_{\text{katz}}(z|y) = \begin{cases} P^*(z|y), & \text{if } C(y,z) > 0 \\ \alpha(y) P^*(z), & \text{otherwise.} \end{cases}$$

Katz Backoff

- Note: We used P* (discounted probabilities) and α weights on the backoff values
- Why not just use regular MLE estimates?

Katz Backoff

- Note: We used P* (discounted probabilities) and α weights on the backoff values
- Why not just use regular MLE estimates?
  - Sum over all w_i in n-gram context $\sum_i P(w_i \mid w_j w_k) = 1$
  - If we back off to lower n-gram?

Katz Backoff

- Note: We used P* (discounted probabilities) and α weights on the backoff values
- Why not just use regular MLE estimates?
  - Sum over all w_i in n-gram context $\sum_i P(w_i \mid w_j w_k) = 1$
  - If we back off to lower n-gram?
    - Too much probability mass ➔ > 1

Katz Backoff

- Note: We used P* (discounted probabilities) and α weights on the backoff values
- Why not just use regular MLE estimates?
  - Sum over all w_i in n-gram context $\sum_i P(w_i \mid w_j w_k) = 1$
  - If we back off to lower n-gram?
    - Too much probability mass ➔ > 1
- Solution:
  - Use P* discounts to save mass for lower order ngrams
  - Apply α weights to make sure sum to amount saved
  - Details in 4.7.1

Toolkits

- Two major language modeling toolkits
  - SRILM
  - Cambridge-CMU toolkit

Toolkits

- Two major language modeling toolkits
  - SRILM
  - Cambridge-CMU toolkit

- Publicly available, similar functionality
  - Training: Create language model from text file
  - Decoding: Computes perplexity/probability of text

OOV words: <UNK> word

- **Out Of Vocabulary** = OOV words

OOV words: <UNK> word

- **Out Of Vocabulary** = OOV words
- We don't use GT smoothing for these

OOV words: <UNK> word

- **Out Of Vocabulary** = OOV words
- We don't use GT smoothing for these
  - Because GT assumes we know the number of unseen events
- Instead: create an unknown word token <UNK>

OOV words: <UNK> word

- **Out Of Vocabulary** = OOV words
- We don't use GT smoothing for these
  - Because GT assumes we know the number of unseen events
- 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

OOV words: <UNK> word

- **Out Of Vocabulary** = OOV words
- We don't use GT smoothing for these
  - Because GT assumes we know the number of unseen events
- 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

Google N-Gram Release

All Our N-gram are Belong to You

By Peter Norvig - 8/03/2006 11:26:00 AM

Posted by Alex Franz and Thorsten Brants, Google Machine Translation Team

Here at Google Research we have been using word n-gram models for a variety of R&D projects, such as statistical machine translation, speech recognition, spelling correction, entity detection, information extraction, and others. While such models have usually been estimated from training

to share this enormous dataset with everyone. 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 Release

- serve as the incoming 92
- serve as the incubator 99
- serve as the independent 794
- serve as the index 223
- serve as the indication 72
- serve as the indicator 120
- serve as the indicators 45
- serve as the indispensable 111
- serve as the indispensible 40
- serve as the individual 234

Google Caveat

- Remember the lesson about test sets and training sets... Test sets should be similar to the training set (drawn from the same distribution) for the probabilities to be meaningful.
- So... The Google corpus is fine if your application deals with arbitrary English text on the Web.
- If not then a smaller domain specific corpus is likely to yield better results.

Class-Based Language Models

- Variant of n-gram models using classes or clusters

Class-Based Language Models

- Variant of n-gram models using classes or clusters
- Motivation: Sparseness
  - Flight app.: P(ORD|to),P(JFK|to),.. P(airport_name|to)
    - Relate probability of n-gram to word classes & class ngram

Class-Based Language Models

- Variant of n-gram models using classes or clusters
- Motivation: Sparseness
  - Flight app.: P(ORD|to),P(JFK|to),.. P(airport_name|to)
    - Relate probability of n-gram to word classes & class ngram
- IBM clustering: assume each word in single class
  - $P(w_i|w_{i-1}) \sim P(c_i|c_{i-1}) \times P(w_i|c_i)$
  - Learn by MLE from data
- Where do classes come from?

Class-Based Language Models

- Variant of n-gram models using classes or clusters
- Motivation: Sparseness
  - Flight app.: P(ORD|to),P(JFK|to),.. P(airport_name|to)
    - Relate probability of n-gram to word classes & class ngram
- IBM clustering: assume each word in single class
  - $P(w_i|w_{i-1}) \sim P(c_i|c_{i-1}) \times P(w_i|c_i)$
  - Learn by MLE from data
- Where do classes come from?
  - Hand-designed for application (e.g. ATIS)
  - Automatically induced clusters from corpus

Class-Based Language Models

- Variant of n-gram models using classes or clusters
- Motivation: Sparseness
  - Flight app.: P(ORD|to),P(JFK|to),.. P(airport_name|to)
    - Relate probability of n-gram to word classes & class ngram
- IBM clustering: assume each word in single class
  - $P(w_i|w_{i-1}) \sim P(c_i|c_{i-1}) \times P(w_i|c_i)$
  - Learn by MLE from data
- Where do classes come from?
  - Hand-designed for application (e.g. ATIS)
  - Automatically induced clusters from corpus

LM Adaptation

- Challenge: Need LM for new domain
  - Have little in-domain data

LM Adaptation

- Challenge: Need LM for new domain
  - Have little in-domain data
- Intuition: Much of language is pretty general
  - Can build from ‘general’ LM + in-domain data

LM Adaptation

- Challenge: Need LM for new domain
  - Have little in-domain data
- Intuition: Much of language is pretty general
  - Can build from 'general' LM + in-domain data
- Approach: LM adaptation
  - Train on large domain independent corpus
  - Adapt with small in-domain data set
- What large corpus?

LM Adaptation

- Challenge: Need LM for new domain
  - Have little in-domain data
- Intuition: Much of language is pretty general
  - Can build from 'general' LM + in-domain data
- Approach: LM adaptation
  - Train on large domain independent corpus
  - Adapt with small in-domain data set
- What large corpus?
  - Web counts! e.g. Google n-grams

Incorporating Longer Distance Context

- Why use longer context?

Incorporating Longer Distance Context

- Why use longer context?
  - N-grams are approximation
    - Model size
    - Sparseness

Incorporating Longer Distance Context

- Why use longer context?
  - N-grams are approximation
    - Model size
    - Sparseness
- What sorts of information in longer context?

Incorporating Longer Distance Context

- Why use longer context?
  - N-grams are approximation
    - Model size
    - Sparseness
- What sorts of information in longer context?
  - Priming
  - Topic
  - Sentence type
  - Dialogue act
  - Syntax

Long Distance LMs

- Bigger n!
  - 284M words: <= 6-grams improve; 7-20 no better

Long Distance LMs

- Bigger n!
  - 284M words: <= 6-grams improve; 7-20 no better
- Cache n-gram:
  - Intuition: Priming: word used previously, more likely
    - Incrementally create ‘cache’ unigram model on test corpus
      - Mix with main n-gram LM

Long Distance LMs

- Bigger n!
  - 284M words: <= 6-grams improve; 7-20 no better
- Cache n-gram:
  - Intuition: Priming: word used previously, more likely
    - Incrementally create ‘cache’ unigram model on test corpus
      - Mix with main n-gram LM
- Topic models:
  - Intuition: Text is about some topic, on-topic words likely
    - $P(w|h) \sim \Sigma t\ P(w|t)P(t|h)$

Long Distance LMs

- Bigger n!
  - 284M words: <= 6-grams improve; 7-20 no better
- Cache n-gram:
  - Intuition: Priming: word used previously, more likely
    - Incrementally create ‘cache’ unigram model on test corpus
      - Mix with main n-gram LM
- Topic models:
  - Intuition: Text is about some topic, on-topic words likely
    - $P(w|h) \sim \Sigma t\ P(w|t)P(t|h)$
- Non-consecutive n-grams:
  - skip n-grams, triggers, variable lengths n-grams

Language Models

- N-gram models:
  - Finite approximation of infinite context history
- Issues: Zeroes and other sparseness
- Strategies: Smoothing
  - Add-one, add-δ, Good-Turing, etc
  - Use partial n-grams: interpolation, backoff
- Refinements
  - Class, cache, topic, trigger LMs

Knesser-Ney Smoothing

- Most commonly used modern smoothing technique
- Intuition: improving backoff
  - I can't see without my reading……
    - Compare P(Francisco|reading) vs P(glasses|reading)

Knesser-Ney Smoothing

- Most commonly used modern smoothing technique
- Intuition: improving backoff
  - I can't see without my reading.....
    - Compare P(Francisco|reading) vs P(glasses|reading)
      - P(Francisco|reading) backs off to P(Francisco)

Knesser-Ney Smoothing

- Most commonly used modern smoothing technique
- Intuition: improving backoff
  - I can't see without my reading.....
    - Compare P(Francisco|reading) vs P(glasses|reading)
      - P(Francisco|reading) backs off to P(Francisco)
      - P(glasses|reading) > 0
      - High unigram frequency of Francisco > P(glasses|reading)

Knesser-Ney Smoothing

- Most commonly used modern smoothing technique
- Intuition: improving backoff
  - I can't see without my reading.....
    - Compare P(Francisco|reading) vs P(glasses|reading)
      - P(Francisco|reading) backs off to P(Francisco)
      - P(glasses|reading) > 0
      - High unigram frequency of Francisco > P(glasses|reading)
      - However, Francisco appears in few contexts, glasses many

Knesser-Ney Smoothing

- Most commonly used modern smoothing technique
- Intuition: improving backoff
  - I can't see without my reading.....
    - Compare P(Francisco|reading) vs P(glasses|reading)
      - P(Francisco|reading) backs off to P(Francisco)
      - P(glasses|reading) > 0
      - High unigram frequency of Francisco > P(glasses|reading)
      - However, Francisco appears in few contexts, glasses many
- Interpolate based on # of contexts
- Words seen in more contexts, more likely to appear in others

Knesser-Ney Smoothing

- Modeling diversity of contexts
  - Continuation probability

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

Knesser-Ney Smoothing

- Modeling diversity of contexts
  - Continuation probability

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

- Backoff:

$$P_{KN}(w_i \mid w_{i-1}) = \begin{cases} \frac{C(w_{i-1}w_i) - D}{C(w_{i-1})} if C(w_{i-1}w_i) > 0 \\ \alpha(w_i) \frac{|\{w_{i-1} : C(w_{i-1}w_i) > 0\}|}{\sum_{w_i} |\{w_{i-1} : C(w_{i-1}w_i) > 0\}|} otherwise \end{cases}$$

Knesser-Ney Smoothing

- Modeling diversity of contexts
  - Continuation probability

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

- Backoff:

$$P_{KN}(w_i \mid w_{i-1}) = \begin{cases} \frac{C(w_{i-1}w_i) - D}{C(w_{i-1})} if C(w_{i-1}w_i) > 0 \\ \alpha(w_i) \frac{|\{w_{i-1} : C(w_{i-1}w_i) > 0\}|}{\sum_{w_i} |\{w_{i-1} : C(w_{i-1}w_i) > 0\}|} otherwise \end{cases}$$

- Interpolation:

$$P_{KN}(w_i \mid w_{i-1}) = \frac{C(w_{i-1}w_i) - D}{C(w_{i-1})} + \beta(w_{i-1}) P_{cont}(w_i)$$

Issues

- Relative frequency
  - Typically compute count of sequence
    - Divide by prefix

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

- Corpus sensitivity
  - Shakespeare vs Wall Street Journal
    - Very unnatural
- Ngrams
  - Unigram: little; bigrams: colloc;trigrams:phrase

Additional Issues in Good-Turing

- General approach:
  - Estimate of c^* for N_c depends on N_{c+1}

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

- What if $N_{c+1} = 0$?
  - More zero count problems
  - Not uncommon: e.g. fish example, no 4s

Modifications

- Simple Good-Turing
- Compute N_c bins, then smooth N_c to replace zeroes
  - Fit linear regression in log space
    - $\log(N_c) = a + b \log(c)$
- What about large c's?
  - Should be reliable
  - Assume $c^* = c$ if c is large, e.g $c > k$ (Katz: $k = 5$)
- Typically combined with other interpolation/backoff