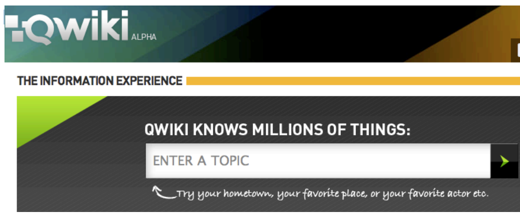


LANGUAGE MODELING: SMOOTHING

David Kauchak
CS159 – Spring 2011

some slides adapted from Jason Eisner

Admin

- Assignment 2 out
  - bigram language modeling
  - Java
  - Can work with partners
    - Anyone looking for a partner?
  - Due Wednesday 2/16 (but start working on it now!)
  - HashMap

Admin

- Our first quiz next Monday (2/14)
  - In-class (~30 min.)
  - Topics
    - corpus analysis
    - regular expressions
    - probability
    - language modeling
  - Open book
    - we'll try it out for this one
    - better to assume closed book (30 minutes goes by fast!)
  - 5% of your grade

Today

Today

- Take home ideas:
  - Key idea of smoothing is to redistribute the probability to handle less see (or never seen) events
    - Still must always maintain a true probability distribution
  - Lots of ways of smoothing data
  - Should take into account features in your data!
  - For n-grams, backoff models and, in particular, Kneser-Ney smoothing work well

Smoothing

What if our test set contains the following sentence, but one of the trigrams never occurred in our training data?

P(I think today is a good day to be me) =
P(I | <start> <start>) x
P(think | <start> I) x
P(today| I think) x
P(is| think today) x
P(a| today is) x
P(good| is a) x
…

If any of these has never been seen before, prob = 0!

Smoothing

P(I think today is a good day to be me) =
P(I | <start> <start>) x
P(think | <start> I) x
P(today| I think) x
P(is| think today) x
P(a| today is) x
P(good| is a) x
…

These probability estimates may be inaccurate. Smoothing can help reduce some of the noise.

Add-lambda smoothing

- A large dictionary makes novel events too probable.
- add $\lambda = 0.01$ to all counts

see the abacus	1	1/3	1.01	1.01/203
see the abbot	0	0/3	0.01	0.01/203
see the abduct	0	0/3	0.01	0.01/203
see the above	2	2/3	2.01	2.01/203
see the Abram	0	0/3	0.01	0.01/203
…			0.01	0.01/203
see the zygote	0	0/3	0.01	0.01/203
Total	3	3/3	203	

Vocabulary

- n-gram language modeling assumes we have a fixed vocabulary
  - why?
- Whether implicit or explicit, an n-gram language model is defined over a finite, fixed vocabulary
- What happens when we encounter a word not in our vocabulary (Out Of Vocabulary)?
  - If we don't do anything, prob = 0
  - Smoothing doesn't really help us with this!

Vocabulary

- To make this explicit, smoothing helps us with…

all entries in our vocabulary

see the abacus	1	1.01
see the abbot	0	0.01
see the abduct	0	0.01
see the above	2	2.01
see the Abram	0	0.01
…		0.01
see the zygote	0	0.01

Vocabulary

- and…

Vocabulary	Counts	Smoothed counts
a	10	10.01
able	1	1.01
about	2	2.01
account	0	0.01
acid	0	0.01
across	3	3.01
…	…	…
young	1	1.01
zebra	0	0.01

How can we have words in our vocabulary we've never seen before?

Vocabulary

- Choosing a vocabulary: ideas?
  - Grab a list of English words from somewhere
  - Use all of the words in your training data
  - Use some of the words in your training data
    - for example, all those the occur more than k times
- Benefits/drawbacks?
  - Ideally your vocabulary should represents words your likely to see
  - Too many words, end up washing out your probability estimates (and getting poor estimates)
  - Too few, lots of out of vocabulary

Vocabulary

- No matter your chosen vocabulary, you're still going to have out of vocabulary (OOV)
- How can we deal with this?
  - Ignore words we've never seen before
    - Somewhat unsatisfying, though can work depending on the application
    - Probability is then dependent on how many in vocabulary words are seen in a sentence/text
  - Use a special symbol for OOV words and estimate the probability of out of vocabulary

Out of vocabulary

- Add an extra word in your vocabulary to denote OOV (<OOV>, <UNK>)
- Replace all words in your training corpus not in the vocabulary with <UNK>
  - You'll get bigrams, trigrams, etc with <UNK>
    - p(<UNK> | "I am")
    - p(fast | "I <UNK>")
- During testing, similarly replace all OOV with <UNK>

Choosing a vocabulary

- A common approach (and the one we'll use for the assignment):
  - Replace the first occurrence of each word by <UNK> in a data set
  - Estimate probabilities normally
- Vocabulary then is all words that occurred two or more times
- This also discounts all word counts by 1 and gives that probability mass to <UNK>

Storing the table

How are we storing this table?
Should we store all entries?

see the abacus	1	1/3	1.01	1.01/203
see the abbot	0	0/3	0.01	0.01/203
see the abduct	0	0/3	0.01	0.01/203
see the above	2	2/3	2.01	2.01/203
see the Abram	0	0/3	0.01	0.01/203
…			0.01	0.01/203
see the zygote	0	0/3	0.01	0.01/203
Total	3	3/3	203	

Storing the table

- Hashtable
  - fast retrieval
  - fairly good memory usage
- Only store those entries of things we've seen
  - for example, we don't store V^3 trigrams
- For trigrams we can:
  - Store one hashtable with bigrams as keys
  - Store a hashtable of hashtables (I'm recommending this)

Storing the table: add-lambda smoothing

- For those we've seen before:

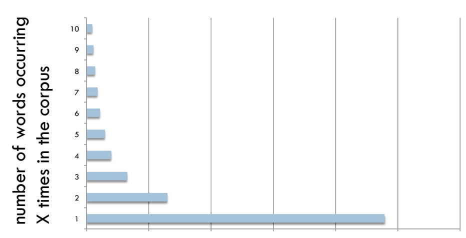
$$P(c \mid ab) = \frac{C(abc) + \lambda}{C(ab) + \lambda V}$$

- Unseen n-grams: p(z|ab) = ?

$$P(z \mid ab) = \frac{\lambda}{C(ab) + \lambda V}$$

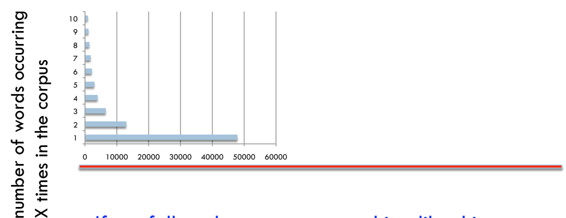
Store the lower order counts
(or probabilities)

How common are novel events?



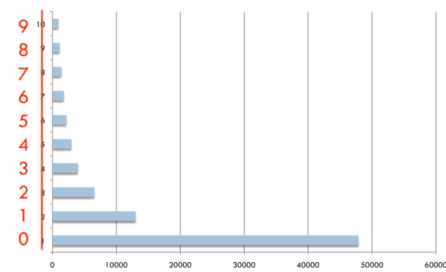
How likely are novel/unseen events?

How common are novel events?



If we follow the pattern, something like this…

Good-Turing estimation



Good-Turing estimation

- N_c = number of words/bigrams occurring c times
- Replace MLE counts for things with count c:

$$c^* = (c+1)\underbrace{\frac{N_{c+1}}{N_c}}_{\text{scale down the next frequency up}}$$

- Estimate the probability of novel events as:

$$p(unseen) = \frac{N_1}{Total\_words}$$

Good-Turing (classic example)

- Imagine you are fishing
  - 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)?

$$p(unseen) = \frac{N_1}{Total\_words} = \frac{3}{18}$$

Good-Turing (classic example)

- Imagine you are fishing
  - 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 next species is trout?

$$c^* = (c+1)\frac{N_{c+1}}{N_c} = 2 * \frac{1}{3} = 0.67$$

$$\boxed{\frac{0.67}{18}}$$

Good-Turing (classic example)

- Imagine you are fishing
  - 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 next species is perch?

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

N_4 is 0!

Nice idea, but kind of a pain to implement in practice

Problems with frequency based smoothing

- The following bigrams have never been seen:

p(X | San)   p(X | ate)

Which would add-lambda pick as most likely?

Which would you pick?

Witten-Bell Discounting

- Some words are more likely to be followed by new words

San	Diego Francisco Luis Jose Marcos	ate	food apples bananas hamburgers a lot for two grapes …

Witten-Bell Discounting

- Probability mass is shifted around, depending on the context of words
- If $P(w_i \mid w_{i-1},\ldots,w_{i-m}) = 0$, then the smoothed probability $P_{WB}(w_i \mid w_{i-1},\ldots,w_{i-m})$ is higher if the sequence $w_{i-1},\ldots,w_{i-m}$ occurs with many different words w_i

Witten-Bell Smoothing

- For bigrams
  - $T(w_{i-1})$ is the number of different words (types) that occur to the right of w_{i-1}
  - $N(w_{i-1})$ is the number of times w_{i-1} occurred
  - $Z(w_{i-1})$ is the number of bigrams in the current data set starting with w_{i-1} that do not occur in the training data

Witten-Bell Smoothing

- if $c(w_{i-1},w_i) > 0$

$$P^{WB}(w_i \mid w_{i-1}) = \frac{c(w_{i-1}w_i)}{N(w_{i-1}) + T(w_{i-1})}$$

$$\frac{\text{\# times we saw the bigram}}{\text{\# times } w_{i-1} \text{ occurred} \quad + \quad \text{\# of types to the right of } w_{i-1}}$$

Witten-Bell Smoothing

- If $c(w_{i-1},w_i) = 0$

$$P^{WB}(w_i \mid w_{i-1}) = \frac{T(w_{i-1})}{Z(w_{i-1})(N + T(w_{i-1}))}$$

Problems with frequency based smoothing

- The following trigrams have never been seen:

p(car | see the) p(zygote | see the)

p(cumquat | see the)

Which would add-lambda pick as most likely?
Good-Turing? Witten-Bell?

Which would you pick?

Better smoothing approaches

- Utilize information in lower-order models
- Interpolation
  - p*(z| x,y) = λ p(z | x, y) + μ p(z | y) + (1 - λ - μ)p(z)
  - Combine the probabilities in some linear combination
- Backoff

$$P(z \mid xy) = \begin{cases} \frac{C^*(xyz)}{C(xy)} & \text{if } C(xyz) > k \\ \alpha(xy)P(z \mid y) & \text{otherwise} \end{cases}$$

  - Often k = 0 (or 1)
  - Combine the probabilities by "backing off" to lower models only when we don't have enough information

Smoothing: Simple Interpolation

$$P(z \mid xy) \approx \lambda \frac{C(xyz)}{C(xy)} + \mu \frac{C(yz)}{C(y)} + (1 - \lambda - \mu)\frac{C(z)}{C(\bullet)}$$

- Trigram is very context specific, very noisy
- Unigram is context-independent, smooth
- Interpolate Trigram, Bigram, Unigram for best combination
- How should we determine λ and μ ?

Smoothing: Finding parameter values

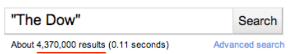
- Just like we talked about before, split training data into training and development
  - can use cross-validation, leave-one-out, etc.
- Try lots of different values for λ, μ on heldout data, pick best
- Two approaches for finding these efficiently
  - EM (expectation maximization)
  - "Powell search" – see Numerical Recipes in C

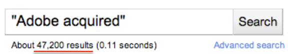
Smoothing: Jelinek-Mercer

- Simple interpolation:

$$P_{smooth}(z \mid xy) = \lambda \frac{C(xyz)}{C(xy)} + (1-\lambda) P_{smooth}(z \mid y)$$

- Should all bigrams be smoothed equally? Which of these is it more likely to start an unseen trigram?





Smoothing: Jelinek-Mercer

- Simple interpolation:

$$P_{smooth}(z \mid xy) = \lambda \frac{C(xyz)}{C(xy)} + (1-\lambda) P_{smooth}(z \mid y)$$

- Multiple parameters: smooth a little after "The Dow", more after "Adobe acquired"

$$P_{smooth}(z \mid xy) =$$
$$\lambda(C(xy)) \frac{C(xyz)}{C(xy)} + (1-\lambda(C(xy)) P_{smooth}(z \mid y)$$

Smoothing: Jelinek-Mercer continued

$$P_{smooth}(z \mid xy) =$$
$$\lambda(C(xy)) \frac{C(xyz)}{C(xy)} + (1-\lambda(C(xy)) P_{smooth}(z \mid y)$$

- Bin counts by frequency and assign λs for each bin
- Find λs by cross-validation on held-out data

Backoff models: absolute discounting

$$P_{absolute}(z \mid xy) = \begin{cases} \frac{C(xyz) - D}{C(xy)} & if\ C(xyz) > 0 \\ \alpha(xy) P_{absolute}(z \mid y) & otherwise \end{cases}$$

- Subtract some absolute number from each of the counts (e.g. 0.75)
  - will have a large effect on low counts
  - will have a small effect on large counts

Backoff models: absolute discounting

$$P_{absolute}(z \mid xy) = \begin{cases} \frac{C(xyz) - D}{C(xy)} & \text{if } C(xyz) > 0 \\ \alpha(xy) P_{absolute}(z \mid y) & \text{otherwise} \end{cases}$$

What is α(xy)?

Backoff models: absolute discounting

see the dog	1
see the cat	2
see the banana	4
see the man	1
see the woman	1
see the car	1

the Dow Jones	10
the Dow rose	5
the Dow fell	5

p(rose | the Dow) = ?

p(jumped | the Dow) = ?

p(cat | see the) = ?

p(puppy | see the) = ?

$$P_{absolute}(z \mid xy) = \begin{cases} \frac{C(xyz) - D}{C(xy)} & \text{if } C(xyz) > 0 \\ \alpha(xy) P_{absolute}(z \mid y) & \text{otherwise} \end{cases}$$

Backoff models: absolute discounting

see the dog	1
see the cat	2
see the banana	4
see the man	1
see the woman	1
see the car	1

p(cat | see the) = ?

$$\frac{2 - D}{10} = \frac{2 - 0.75}{10} = .125$$

$$P_{absolute}(z \mid xy) = \begin{cases} \frac{C(xyz) - D}{C(xy)} & \text{if } C(xyz) > 0 \\ \alpha(xy) P_{absolute}(z \mid y) & \text{otherwise} \end{cases}$$

Backoff models: absolute discounting

see the dog	1
see the cat	2
see the banana	4
see the man	1
see the woman	1
see the car	1

p(puppy | see the) = ?

α(see the) = ?

How much probability mass did we reserve/discount for the bigram model?

$$P_{absolute}(z \mid xy) = \begin{cases} \frac{C(xyz) - D}{C(xy)} & \text{if } C(xyz) > 0 \\ \alpha(xy) P_{absolute}(z \mid y) & \text{otherwise} \end{cases}$$

Backoff models: absolute discounting

see the dog	1
see the cat	2
see the banana	4
see the man	1
see the woman	1
see the car	1

p(puppy | see the) = ?

α (see the) = ?

$$\frac{\text{\# of } \textit{types} \text{ starting with “see the” * D}}{\text{count(“see the”)}}$$

For each of the unique trigrams, we subtracted D/count(“see the”) from the probability distribution

$$P_{absolute}(z \mid xy) = \begin{cases} \frac{C(xyz) - D}{C(xy)} & \text{if } C(xyz) > 0 \\ \alpha(xy) P_{absolute}(z \mid y) & \text{otherwise} \end{cases}$$

Backoff models: absolute discounting

see the dog	1
see the cat	2
see the banana	4
see the man	1
see the woman	1
see the car	1

p(puppy | see the) = ?

α (see the) = ?

$$\frac{\text{\# of } \textit{types} \text{ starting with “see the” * D}}{\text{count(“see the”)}}$$

$$reserved\_mass(\text{see the}) = \frac{6 * D}{10} = \frac{6 * 0.75}{10} = 0.45$$

$$P_{absolute}(z \mid xy) = \begin{cases} \frac{C(xyz) - D}{C(xy)} & \text{if } C(xyz) > 0 \\ \alpha(xy) P_{absolute}(z \mid y) & \text{otherwise} \end{cases}$$

distribute this probability mass to all bigrams that we backed off to

Calculating α

- We have some number of bigrams we're going to backoff to, i.e. those X where C(see the X) = 0, that is unseen trigrams starting with “see the”
- When we backoff, for each of these, we'll be including their probability in the model: P(X | the)
- α is the normalizing constant so that the sum of these probabilities equals the reserved probability mass

$$\sum_{X: C(\text{see the X}) == 0} p(\text{X} \mid \text{the}) = reserved\_mass(\text{see the})$$

Calculating α

- We can calculate α two ways
  - Based on those we haven't seen:

$$\alpha(\text{see the}) = \frac{reserved\_mass(\text{see the})}{\sum_{X: C(\text{see the X}) = 0} p(\text{X} \mid \text{the})}$$

  - Or, more often, based on those we do see:

$$\alpha(\text{see the}) = \frac{reserved\_mass(\text{see the})}{1 - \sum_{X: C(\text{see the X}) > 0} p(\text{X} \mid \text{the})}$$

Calculating α in general: trigrams

- Calculate the reserved mass

$$\text{reserved\_mass(bigram)} = \frac{\#\text{ of } \textit{types} \text{ starting with bigram} * D}{\text{count(bigram)}}$$

- Calculate the sum of the backed off probability. For bigram "A B":

$$1 - \sum_{X:C(A\ B\ X)>0} p(X \mid B) \qquad \sum_{X:C(A\ B\ X)=0} p(X \mid B)$$

either is fine in practice, the left is easier

- Calculate α

$$\alpha(A\ B) = \frac{reserved\_mass(A\ B)}{1 - \sum_{X:C(A\ B\ X)>0} p(X \mid B)}$$

1 – the sum of the bigram probabilities of those trigrams that we saw starting with bigram A B

Calculating α in general: bigrams

- Calculate the reserved mass

$$\text{reserved\_mass(unigram)} = \frac{\#\text{ of } \textit{types} \text{ starting with unigram} * D}{\text{count(unigram)}}$$

- Calculate the sum of the backed off probability. For bigram "A B":

$$1 - \sum_{X:C(A\ X)>0} p(X) \qquad \sum_{X:C(A\ X)=0} p(X)$$

either is fine in practice, the left is easier

- Calculate α

$$\alpha(A) = \frac{reserved\_mass(A)}{1 - \sum_{X:C(A\ X)>0} p(X)}$$

1 – the sum of the unigram probabilities of those bigrams that we saw starting with word A

Calculating backoff models in practice

- Store the αs in another table
  - If it's a trigram backed off to a bigram, it's a table keyed by the bigrams
  - If it's a bigram backed off to a unigram, it's a table keyed by the unigrams
- Compute the αs during training
  - After calculating all of the probabilities of seen unigrams/bigrams/trigrams
  - Go back through and calculate the αs (you should have all of the information you need)
- During testing, it should then be easy to apply the backoff model with the αs pre-calculated

Backoff models: absolute discounting

the Dow Jones	10
the Dow rose	5
the Dow fell	5

p(jumped | the Dow) = ?

α(the Dow) = ?

$$\frac{\#\text{ of } \textit{types} \text{ starting with "see the"} * D}{\text{count("see the")}}$$

$$reserved\_mass(\text{the Dow}) = \frac{3 * D}{20} = \frac{3 * 0.75}{20} = 0.115$$

$$\alpha(\text{the Dow}) = \frac{reserved\_mass(\text{see the})}{1 - \sum_{X:C(\text{the Dow } X)>0} p(X \mid \text{the})}$$

Backoff models: absolute discounting

$$\text{reserved\_mass} = \frac{\#\text{ of } \textit{types} \text{ starting with bigram} * D}{\text{count(bigram)}}$$

- Two nice attributes:
  - decreases if we've seen more bigrams
    - should be more confident that the unseen trigram is no good
  - increases if the bigram tends to be followed by lots of other words
    - will be more likely to see an unseen trigram

Kneser-Ney

- Idea: not all counts should be discounted with the same value

common
P(Francisco | eggplant) vs
P(stew | eggplant)
rarer

If we've never seen either, which should be more likely? why?

What would an normal discounted backoff model say?

What is the problem?

Kneser-Ney

- Idea: not all counts should be discounted with the same value

P(Francisco | eggplant) vs
P(stew | eggplant)

Problem:
- Both of these would have the same backoff parameter since they're both conditioning on eggplant
- We then would end up picking based on which was most frequent
- However, even though Francisco tends to only be preceded by a small number of words

Kneser-Ney

- Idea: not all counts should be discounted with the same value
- "Francisco" is common, so backoff/interpolated methods say it is likely
  - But it only occurs in context of "San"
- "Stew" is common in many contexts
- Weight backoff by number of contexts word occurs in

P(Francisco | eggplant) low
P(stew | eggplant) higher

Kneser-Ney

$$P_{absolute}(z \mid xy) = \begin{cases} \dfrac{C(xyz) - D}{C(xy)} & if\ C(xyz) > 0 \\ \alpha(xy) P_{absolute}(z \mid y) & otherwise \end{cases}$$

instead of the probability of the word/bigram occurring, use the probability of the

$$P_{absolute}(z \mid xy) = \begin{cases} \dfrac{C(xyz) - D}{C(xy)} & if\ C(xyz) > 0 \\ \alpha(xy) P_{CONTINUATION}(z \mid y) & otherwise \end{cases}$$

$P_{CONTINUATION}$

- Relative to other words, how likely is this word to continue (i.e. follow) many other words

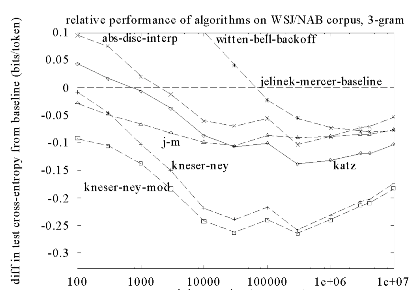
$$P_{CONTINUATION}(z \mid y) = \frac{\#\ \text{types ending with } yz}{\sum_{bc \in bigrams} \#\text{types ending with bigram } bc}$$

$$= \frac{|\{xyz : C(xyz) > 0\}|}{\sum_{bc \in bigrams} |\{abc : C(abc) > 0\}|}$$

Other language model ideas?

- Skipping models: rather than just the previous 2 words, condition on the previous word and the 3rd word back, etc.
- Caching models: phrases seen are more likely to be seen again (helps deal with new domains)
- Clustering:
  - some words fall into categories (e.g. Monday, Tuesday, Wednesday…)
  - smooth probabilities with category probabilities
- Domain adaptation:
  - interpolate between a general model and a domain specific model

Smoothing results



Language Modeling Toolkits

- SRI
  - http://www-speech.sri.com/projects/srilm/
- CMU
  - http://www.speech.cs.cmu.edu/SLM_info.html