N-grams
L545
Markus Dickinson
Dept. of Linguistics, Indiana
Spring 2010

### N-grams: Motivation

An **n-gram** is a stretch of text *n* words long
- Approximation of language: information in *n*-grams tells us something about language, but doesn’t capture the structure
- Efficient: finding and using every, e.g., two-word collocation in a text is quick and easy to do

*N*-grams can help in a variety of NLP applications:
- Word prediction = *n*-grams can be used to aid in predicting the next word of an utterance, based on the previous *n* – 1 words
- Useful for context-sensitive spelling correction, approximation of language, ...

### Corpus-based NLP

**Corpus** (pl. corpora) = a computer-readable collection of text and/or speech, often with annotations
- We can use corpora to gather probabilities and other information about language use
  - We can say that a corpus used to gather prior information is **training data**
  - Testing data, by contrast, is the data one uses to test the accuracy of a method
- We can distinguish **types** and **tokens** in a corpus
  - type = distinct word (e.g., *like*)
  - token = distinct occurrence of a word (e.g., the type *like* might have 20,000 tokens in a corpus)

### Simple n-grams

Let’s assume we want to predict the next word, based on the previous context of *I dreamt I saw the knights in*
- What we want to find is the likelihood of *w*₇ being the next word, given that we’ve seen *w*₁,..., *w*₆
  - This is *P(w*₇|*w*₁,..., *w*₆)
  - But, to start with, we examine *P(*w*₁,..., *w*₇)*

In general, for *w*ᵢ, we are concerned with:

\[(1) \ P(*w*₁,..., *w*ᵢ) = P(*w*₁)P(*w*₂|*w*₁)...P(*w*ᵢ|*w*ᵢ₋₁)\]

But these probabilities are impractical to calculate: they hardly ever occur in a corpus, if at all.
- And it would be a lot of data to store, if we could calculate them.

### Unigrams

So, we can approximate these probabilities to a particular *n*-gram, for a given *n*. What should *n* be?
- Unigrams (*n* = 1):
  \[(2) \ P(*w*ᵢ|*w*₁,..., *w*ᵢ₋₁) \approx P(*w*ᵢ)\]
  - Easy to calculate, but we have no contextual information
  - The quick brown fox jumped
- We would like to say that *over* has a higher probability in this context than *lazy* does.

### Bigrams

**bigrams** (*n* = 2) are a better choice and still easy to calculate:

\[(4) \ P(*w*ᵢ|*w*₁,..., *w*ᵢ₋₁) \approx P(*w*ᵢ|*w*ᵢ₋₁)\]
\[(5) \ P(*over|The, quick, brown, fox, jumped) \approx P(*over|jumped)\]

And thus, we obtain for the probability of a sentence:

\[(6) \ P(*w*₁,..., *w*ᵢ) = P(*w*₁)P(*w*₂|*w*₁)...P(*w*ᵢ|*w*ᵢ₋₁)\]
Markov models

A bigram model is also called a **first-order Markov model** because it has one element of memory (one token in the past)

- Markov models are essentially weighted FSAs—i.e., the arcs between states have probabilities
- The states in the FSA are words

Much more on Markov models when we hit POS tagging ...

Bigram example

What is the probability of seeing the sentence *The quick brown fox jumped over the lazy dog*?

(7) \[ P(\text{The quick brown fox jumped over the lazy dog}) = P(\text{The|START})P(\text{quick|The})P(\text{brown|quick}) \ldots P(\text{dog|lazy}) \]

- Probabilities are generally small, so log probabilities are usually used

Q: Does this favor shorter sentences?
- A: Yes, but it also depends upon \( P(\text{END|lastword}) \)

Trigrams

If bigrams are good, then trigrams \( (n = 3) \) can be even better.

- Wider context: \( P(\text{know|did, he}) \) vs. \( P(\text{know|he}) \)
- Generally, trigrams are still short enough that we will have enough data to gather accurate probabilities

... which means we should talk about how to gather data

Training n-gram models

Go through corpus and calculate relative frequencies:

(8) \[ P(w_n|w_{n-1}) = \frac{C(w_{n-1}, w_n)}{C(w_{n-1})} \]

(9) \[ P(w_n|w_{n-2}, w_{n-1}) = \frac{C(w_{n-2}, w_{n-1}, w_n)}{C(w_{n-2}, w_{n-1})} \]

This technique of gathering probabilities from a training corpus is called **maximum likelihood estimation (MLE)**

Smoothing: Motivation

Let’s assume that we have a good corpus and have trained a bigram model on it, i.e., learned MLE probabilities for bigrams

- But we won’t have seen every possible bigram:
  - *lickety split* is a possible English bigram, but it may not be in the corpus

- This is a problem of **data sparseness** → there are zero probability bigrams which are actual possible bigrams in the language

To account for this sparseness, we turn to **smoothing** techniques → making zero probabilities non-zero

- adjust probabilities to account for unseen data

Know your corpus

We mentioned earlier about having training data and testing data

- It’s important to remember what your training data is when applying your technology to new data
  - If you train your trigram model on Shakespeare, then you have learned the probabilities in Shakespeare, not the probabilities of English overall
- What corpus you use depends on what you want to do later
Add-One Smoothing

One way to smooth is to add a count of one to every bigram:
- in order to still be a probability, all probabilities need to be normalized to sum to one
- so, we add the number of word types to the denominator
  - we added one to every type of bigram, so we need to account for all our numerator additions

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

\( V = \text{total number of word types in the lexicon (each one gets a count of one)} \)

Discounting

An alternate way of viewing smoothing is as discounting:
- Lowering non-zero counts to get the probability mass we need for the zero count items
- The discounting factor can be defined as the ratio of the smoothed count to the MLE count

\[ \text{Jurafsky and Martin show that add-one smoothing can discount probabilities by a factor of 10!} \]
- This is because too much of the probability mass is now in the zeros

That's way too much ...

Witten-Bell Discounting

\[ n > 2 \]

\[ \text{Idea: Use the counts of words you have seen once to estimate those you have never seen} \]
- Instead of simply adding one to every \( n \)-gram, compute the probability of \( w_{n-1}, w_n \) by seeing how likely \( w_{n-1} \) is at starting any \( n \)-gram.
- Words that begin lots of bigrams lead to higher “unseen bigram” probabilities
- Non-zero bigrams are discounted in essentially the same manner as zero count bigrams
  - Jurafsky and Martin show that they are only discounted by about a factor of one

\[ \text{Witten-Bell Discounting formula} \]

\[ p^*(w_n|w_{n-1}) = \frac{T(w_{n-1})}{Z(w_{n-1})[N(w_{n-1}) + T(w_{n-1})]} \]

- \( T(w_{n-1}) = \text{number of bigram types starting with } w_{n-1} \)
  - determines how high the value will be (numerator)
- \( N(w_{n-1}) = \text{no. of bigram tokens starting with } w_{n-1} \)
  - \( N(w_{n-1}) + T(w_{n-1}) \) gives total number of “events” to divide by
- \( Z(w_{n-1}) = \text{number of bigram tokens starting with } w_{n-1} \)
  - and having zero count
  - this just distributes the probability mass between all zero count bigrams starting with \( w_{n-1} \)

\[ \text{Good-Turing discounting} \]

\( \text{Good-Turing is based on a similar idea to Witten-Bell discounting:} \)
- use counts of more frequent \( n \)-grams to estimate less frequent \( n \)-grams

\[ \text{Good-Turing discounting:} \]

\[ \text{count}_{+} = (\text{count} + 1) \frac{N_c}{N_c} \]

where \( \text{count} = \text{MLE count of } N_c \), and \( N_c \) is the count of \( n \)-grams of frequency \( c \)
- Probabilities are readjusted based on these re-calculated counts
Kneser-Ney Smoothing (absolute discounting)

Good-Turing and Witten-Bell Discounting are based on using relative discounting factors
- Kneser-Ney simplifies this by using absolute discounting factors
- So, instead of multiplying by a ratio, we simply subtract some discounting factor

Class-based N-grams

Intuition: we may not have seen a word before, but we may have seen a word like it
- Never observed Shanghai, but have seen other cities
- Can use a type of hard clustering, where each word is only assigned to one class (IBM clustering)

\[
P(w_i|w_{i-1}) \approx P(c_i|c_{i-1}) \times P(w_i|c_i)
\]

POS tagging equations will look fairly similar to this...

Backoff models: Basic idea

Let's say we're using a trigram model for predicting language, and we haven't seen a particular trigram before.
- But maybe we've seen the bigram, or if not, the unigram information would be useful
- Backoff models allow one to try the most informative n-gram first and then back off to lower n-grams

Backoff equations

Roughly speaking, this is how a backoff model works:
- If this trigram has a non-zero count, we use that information
  \[
P(w_i|w_{i-2}w_{i-1}) = \alpha_2 P(w_i)
\]
- else, if the bigram count is non-zero, we use that bigram information:
  \[
P(w_i|w_{i-2}w_{i-1}) = \alpha_1 P(w_i|w_{i-1})
\]
- and in all other cases we just take the unigram information:
  \[
P(w_i|w_{i-2}w_{i-1}) = \alpha_2 P(w_i)
\]

Backoff models: example

Let's say we've never seen the trigram maples want more before
- But we have seen want more, so we can use that bigram to calculate a probability estimate.
- So, we look at \(P(\text{more}|\text{want})\) ...
- But we're now assigning probability to \(P(\text{more}|\text{maples}, \text{want})\) which was zero before \(\Rightarrow\) we won't have a true probability model anymore
  \[
  \Rightarrow \text{This is why } \alpha_1 \text{ was used in the previous equations, to assign less re-weight to the probability.}
  \]
In general, backoff models have to be combined with discounting models

Deleted Interpolation

Deleted interpolation is similar to backing off, except that we always use the bigram and unigram information to calculate the probability estimate

\[
P(w_i|w_{i-2}w_{i-1}) = \lambda_1 P(w_i|w_{i-2}w_{i-1}) + \lambda_2 P(w_i|w_{i-1}) + \lambda_3 P(w_i)
\]

where the lambdas (\(\lambda\)) all sum to one
- Every trigram probability, then, is a composite of the focus word's trigram, bigram, and unigram.
A note on information theory

Some very useful notions for n-gram work can be found in information theory. Basic ideas:

- **entropy** = a measure of how much information is needed to encode something
- **perplexity** = a measure of the amount of surprise of an outcome
- **mutual information** = the amount of information one item has about another item (e.g., collocations have high mutual information)

Take L645 to find out more ...