N-grams

L445 / L545

Dept. of Linguistics, Indiana University
Spring 2017
Morphosyntax

We just finished talking about morphology (cf. words)
  ▶ And pretty soon we’re going to discuss syntax (cf. sentences)

In between, we’ll handle words in context
  ▶ Today: n-gram language modeling (bird’s-eye view)
  ▶ Next time: POS tagging (emphasis on rule-based techniques)

Both of these topics involve approximating grammar
  ▶ Both topics are covered in more detail in L645
N-grams: Motivation

An **n-gram** is a stretch of text \(n\) words long

- Approximation of language: \(n\)-grams tells us something about language, but doesn’t capture 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
- Context-sensitive spelling correction
- Machine Translation post-editing
- ...

We are interested in how \(n\)-grams capture **local** properties of grammar
Corpus-based NLP

**Corpus** (pl. corpora) = a computer-readable collection of text and/or speech, often with annotations

- Use corpora to gather probabilities & other information about language use
  - **Training data**: data used to gather prior information
  - **Testing data**: data used to test method accuracy

- A “word” may refer to:
  - **Type**: distinct word (e.g., *like*)
  - **Token**: distinct occurrence of a word (e.g., the type *like* might have 20,000 token occurrences in a corpus)
Simple n-grams

Let’s assume we want to predict the next word, based on the previous context of *The quick brown fox jumped*

- Goal: find the likelihood of $w_6$ being the next word, given that we’ve seen $w_1, ..., w_5$
  - This is: $P(w_6|w_1, ..., w_5)$

In general, for $w_n$, we are concerned with:

\[
P(w_1, ..., w_n) = P(w_1)P(w_2|w_1)...P(w_n|w_1, ..., w_{n-1})
\]

or:

\[
P(w_1, ..., w_n) = P(w_1|\text{START})P(w_2|w_1)...P(w_n|w_1, ..., w_{n-1})
\]

Issues:

- Very specific $n$-grams that may never occur in training
- Huge number of potential $n$-grams
- Missed generalizations: often local context is sufficient to predict a word or disambiguate the usage of a word
Unigrams

Approximate these probabilities to $n$-grams, for a given $n$

- Unigrams ($n = 1$):

  $P(w_n | w_1, \ldots, w_{n-1}) \approx P(w_n)$

- Easy to calculate, but lack contextual information

  (3) The quick brown fox jumped

  - We would like to say that *over* has a higher probability in this context than *lazy* does
Bigrams

**bigrams** \((n = 2)\) give context & are still easy to calculate:

\[
P(w_n|w_1, \ldots, w_{n-1}) \approx P(w_n|w_{n-1})
\]

\[
P(over|The, quick, brown, fox, jumped) \approx P(over|jumped)
\]

The probability of a sentence:

\[
P(w_1, \ldots, w_n) = P(w_1|START)P(w_2|w_1)P(w_3|w_2)\ldots P(w_n|w_{n-1})
\]
Markov models

A bigram model is also called a **first-order Markov model**

- *First-order*: 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

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

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

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

Trigrams \((n = 3)\) encode more context

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

Go through corpus and calculate **relative frequencies**:

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

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

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

Assume: a bigram model has been trained on a good corpus (i.e., learned MLE bigram probabilities)

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

- Problem = **data sparsity** → zero probability bigrams that are actual possible bigrams in the language

**Smoothing** techniques account for this

- Adjust probabilities to account for unseen data
- Make zero probabilities non-zero
Language modeling: comments

Note a few things:

- Smoothing shows that the goal of $n$-gram language modeling is to be **robust**
  - vs. our general approach this semester of defining what is and what is not a part of a grammar

- Some robustness can be achieved in other ways, e.g., moving to more abstract representations (more later)

- Training data choice is a big factor in what is being modeled
  - Trigram model trained on Shakespeare represents the probabilities in Shakespeare, not of English overall
  - Choice of corpus depends upon the purpose
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 sum to one
- Thus: add 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

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

\(V = \text{total number of word types in the lexicon}\)
So, if *treasure trove* never occurred in the data, but *treasure* occurred twice, we have:

\[(11) \quad P^*(\text{trove}|\text{treasure}) = \frac{0+1}{2+V} \]

The probability won’t be very high, but it will be better than 0

- If the surrounding probabilities are high, *treasure trove* could be the best pick
- If the probability were zero, there would be no chance of appearing
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

$\Rightarrow$ Jurafsky and Martin show that add-one smoothing can discount probabilities by a factor of 10!

- Too much of the probability mass is now in the zeros

We will examine one way of handling this; more in L645
Witten-Bell Discounting

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 $w_{i-1}, w_i$ by seeing how likely $w_{i-1}$ is at starting any bigram.

- 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
  
  $\rightarrow$ Jurafsky and Martin show that they are only discounted by about a factor of one
Witten-Bell Discounting formula

\[(12) \text{ zero count bigrams:} \]
\[ p^*(w_i|w_{i-1}) = \frac{T(w_{i-1})}{Z(w_{i-1})(N(w_{i-1})+T(w_{i-1}))} \]

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

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

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

POS tagging equations will look fairly similar to this ...
Assume a trigram model for predicting language, where we haven’t seen a particular trigram before

- Maybe we’ve seen the bigram or the unigram
- 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, use that:

\[
\hat{P}(w_i|w_{i-2}w_{i-1}) = P(w_i|w_{i-2}w_{i-1})
\]  

- Else, if the bigram count is non-zero, use that:

\[
\hat{P}(w_i|w_{i-2}w_{i-1}) = \alpha_1 P(w_i|w_{i-1})
\]

- In all other cases, use the unigram information:

\[
\hat{P}(w_i|w_{i-2}w_{i-1}) = \alpha_2 P(w_i)
\]
Backoff models: example

Assume: never seen the trigram *maples want more* before

- If we have seen *want more*, we use that bigram to calculate a probability estimate \( P(\text{more}|\text{want}) \)
- But we’re now assigning probability to \( P(\text{more}|\text{maples, want}) \) which was zero before
  - We won’t have a true probability model anymore
  - This is why \( \alpha_1 \) was used in the previous equations, to assign less weight to the probability.

In general, backoff models are combined with discounting models

- Point for us: which pieces of (local) information are most relevant to making a decision?