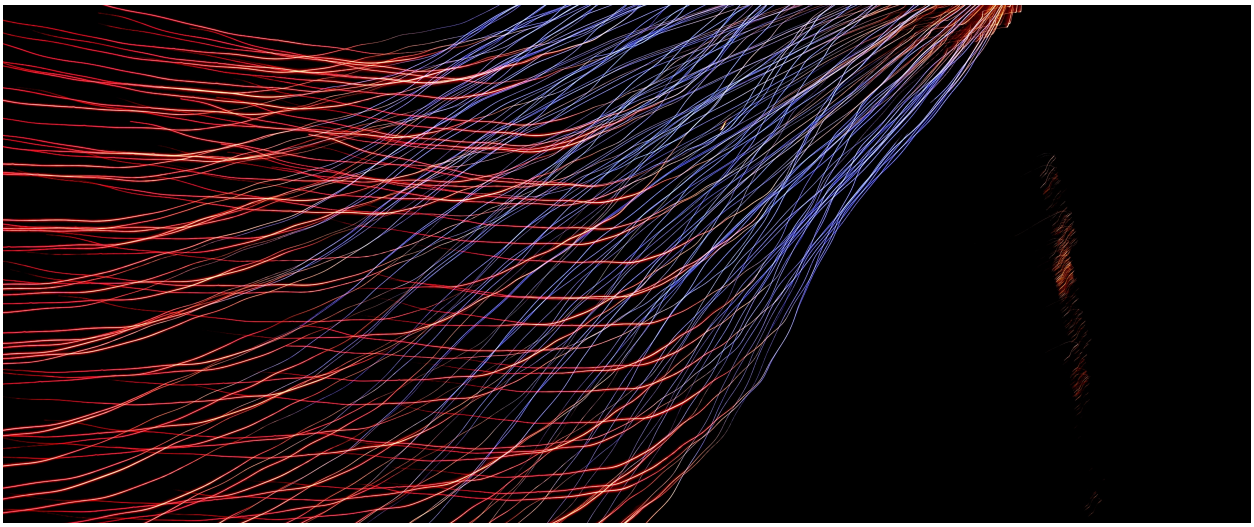


N-Grams

Written by Erik Rodriguez & Arielle Posadas

Source: Exploring NLP with Python by Dr. Mazidi

What are N-Grams?

N-grams are sequences of N contiguous items taken on a corpus of text where the letter N represents the number of items. They are used in Natural Language Processing to build language models.

N-gram example:

Take a look at the following sentence

> The quick brown fox jumps

A 2-gram aka ‘bigram’ of the following sentence would look something like this

- ‘The quick’
- ‘quick brown’
- ‘brown fox’
- ‘fox jumps’

Ways N-grams can be used in real-world applications :

- Classifying text: It can help us determine a specific type, such as in the case of language
- Search engine optimizations: Determine which searches are more common and optimize speed for quicker search retrievals, such as in auto suggestions.
- Spell corrections: It can help spell checkers determine commonly misspelled words

How are Language Models created?

To create a language model, you must first preprocess the text, tokenize it, count the frequency of each n-gram occurring, and use statistics to determine the probability of each n-gram occurring on any given text.

How are probabilities calculated for the Unigrams and Bigrams?

We can calculate the probability of a unigram with the formula:

$$P(w_i) = \frac{C(w_i)}{N}$$

It calculates the total number of that unigram occurring divided by the total number of words.

Meanwhile, the formula to calculate bigram probabilities is:

$$P(w_i w_{i+1}) = \frac{C(w_i w_{i+1})}{C(w_i)}$$

This calculates the total number of the bigram occurring divided by the total number of the first word occurring. For example, if the bigram is “red apple,” then it computes the count of “red apple” in the text divided by the count of “red” in the text.

The source text in building a language model and smoothing

While language models created through N-grams may be impressive and helpful. It is important to remember the source of the text a language model is trained upon. Since it is impossible to test all knowledge, language models could build potentially biased and inaccurate information. This is especially prevalent if you try to generate large pieces with a small sample size since repetition and lack of coherence will be highly apparent.

Smoothing

There may be a case where the probability of a given n-gram will result in zero if the count of a given token is since it will zero out the multiplication. This is not a desired result and will result in a sparsity problem because it is impossible to include every given word in our text. To counter this issue, we can use a technique called smoothing. There are different types of smoothing, but the simplest is called one-add smoothing, which consists of adding a count of 1 to all zeros and readjusting the probabilities to offset the counts.

Language models for text generation

N-grams can be used to generate text. If you start with a word as input and loop through the n-grams that have a higher probability of appearing with a starting word, This approach continues until you reach an n-gram that ends with a period.

Limitations

While the approach listed above may create some grammatically correct sentences, it can be heavily limited by its small size and simple algorithm for creating sentences. Usually, the smaller the n-gram, the worse the result. Using the NLTK generate function with a large text such as the inaugural text can generate a better result. The more significant limitation, however, is that this form of generation is based on probabilities and can still result in text that doesn’t make sense on a larger scale.

Perplexity: Ways to evaluate a language model

Extrinsic

It consists of having a group of human annotators check the result of text generation using metrics. The downside of this method is that it is expensive and highly time-consuming therefore, it isn’t used often.

Intrinsic

This method uses metrics to compare models. A common metric is called **Perplexity**

Perplexity measures how language predicts the text in a given test data using the following formula:

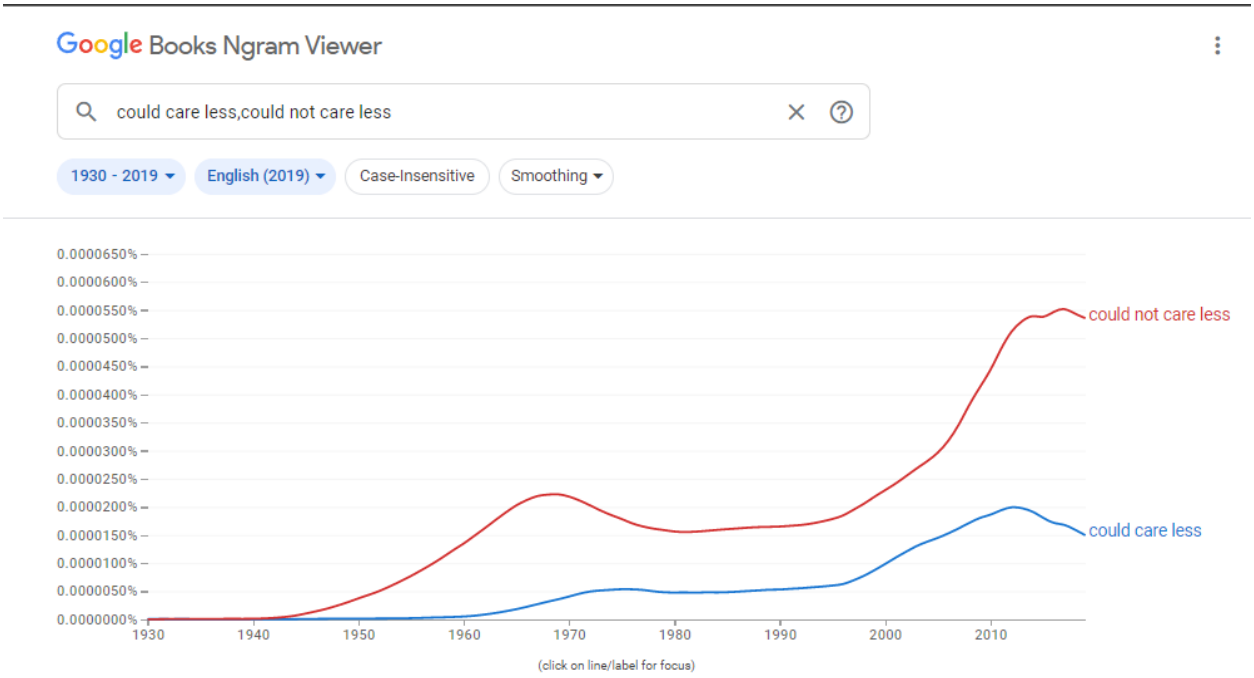
$$PP(W) = P(w_1 w_2 ... w_N)^{\frac{-1}{N}}$$

Which is the function that NLTK has under the hood to calculate the perplexity of any n-gram model.

Google N-gram viewer

Google N-gram viewer is a tool that feeds Google Books to provide the popularity of a sequence of text over a period of time. It also includes common misspellings and gibberish of words.

For example, a phrase often mistyped is “could care less” vs. “could not care less.”



Try Google N-gram viewer by yourself by clicking the link below.

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