Documentation by Jordan Frimpter, Henry Kim

N-Grams

N-grams are a series of sequential tokens in a string, where n is the number of tokens in the sequence. Given that words derive part of their meanings from the context of other words surrounding them, it can be helpful to examine language in the context of n-grams. An n-gram can be viewed as a sliding window of n words over a sequence of words. Unigrams consist of one word, bigrams consist of two consecutive words, trigrams consist of three consecutive words, etc.

N-grams can be used in spelling correction, machine translation, speech recognition, and typing suggestions [1]. In general, n-gram models are useful for making predictions about which words should follow others. One limitation of n-grams is that they do not necessarily combine words that are related by being part of the same phrase, so their ability to provide semantic insight is limited.

A probabilistic model of a language can be created by using n-grams to calculate the probabilities of one word following another. For example, the phrase “employees must wash their …” is more likely to be completed by “hands” than “bird.” N-gram language models can be made by observing a large corpus of words and calculating the probabilities of each n-gram, which is the conditional probability of observing the words of the n-gram given that the first words of the n-gram have been observed. This can be calculated by dividing the number of times the n-gram occurs in the corpus by the total number of words in the corpus.

> Probability of viewing an n-gram:
> $P(x) = v/z$, where $P(x)$ is the probability of seeing an n-gram x where x is an n-gram of size n, v is the number of times x appears in the training corpus, and z is the total number of n-grams of length n present in the training corpus.
>
> Probability of viewing an n-gram with Markov Assumption:
> $P(W) = \Pi_{k=1}{}^{n} P(w_k|w_{k-1})$ is the probability of seeing the n-gram *W* of length *n* if we make the Markov assumption and only consider the previous word when calculating the conditional probability of the next word [1].

Since the N-gram model is built by observing patterns, the corpus observed will make a significant difference in the characteristics of the model. A model trained on Tweets would use modern slang and abbreviations that a model trained on 1900’s literature would not. Another consideration in building an n-gram model is considering how to account for words and n-grams that were not seen in the corpus, a process known as ‘smoothing’. If no smoothing is used, an n-gram model calculates the probability of a sequence of words as zero if the model encounters a word or n-gram not present in the training corpus. One of the most common approaches to smoothing is Laplace Smoothing, where all words in a vocabulary are given at least 1 instance so the probability is very low instead of zero for words that do not appear in the training corpus but are still relevant to the problem.

Laplace Smoothing:

$P(W) = (b + 1)/(u + v)$ is a Laplace Smoothing version of the equation for the probability of a bigram W.

- variable b is the number of times the bigram appears in the corpus
- variable u is the number of times the first word of the bigram appears in the corpus
- variable v is the total size of the corpus vocabulary

In this method, $P(W)$ is never zero even if x does not appear in the training corpus.

Another smoothing method to ensure $P(W)$ is never zero is to Good-Turing smoothing which uses the probability of words only seen once in the corpora as the probability for words that were never seen in training.

N-gram models can be used for text generation by 'guessing' which word or words should appear next in a phrase based on the probabilistic training it has received from the n-grams it has studied. An n-gram model is limited by what it has seen during training and has no real context as to the meaning of the words it generates, only generating words that 'probably go together'. This makes the model poor for generating novel ideas in the way a human child might, but good for predicting patterns in common structures, such as autofilling a search bar.

One way to measure the quality of an n-gram model is with the intrinsic measure of perplexity. To measure perplexity, some data is set aside as test data and the model is evaluated by how well the model can predict the text set aside as test data.

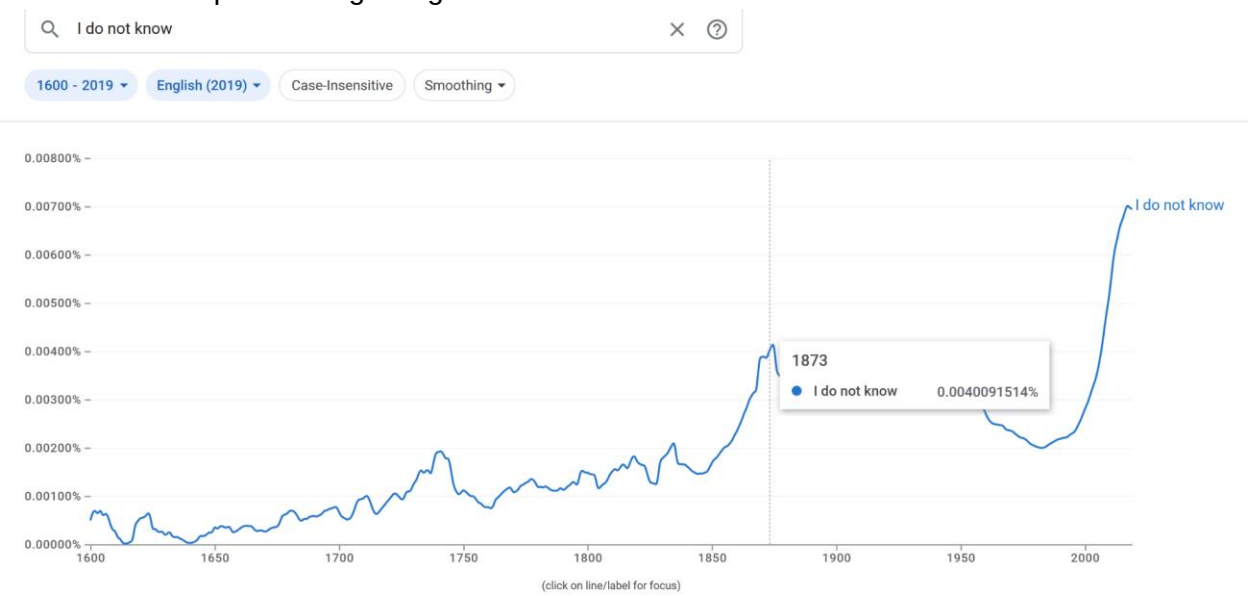
Formula for Perplexity:

$PP(W) = P(w_1 w_2 .. w_N)^{(-1/N)}$, where $P(w_1 w_2 .. w_N)$ is the probability seeing the sequence of words $w_1 w_2 .. w_N$ and *N* is the number of words in the sequence. A lower perplexity is considered better since it corresponds to less chaos in the data [1].

Google has an n-gram viewer that can be used to see statistics of n-gram use over time in the corpus of the Google Books library [2]. The tool will search over a specified timeframe and in a specified text corpus subset of the Google Books library for the appearance of the sought n-gram(s).

Documentation by Jordan Frimpter, Henry Kim

Here is an example of Google N-gram Viewer use:



References:

[1] K. Mazidi, Exploring NLP with Python, 2019.

[2] Google, Books Ngram Viewer https://books.google.com/ngrams/.