An n-gram is a sliding window over text that views n words at a time [1]. Unigrams are essentially alphabetical tokens where the sliding window views one word at a time, and a bigram has a sliding window that views two words at a time. Any number of words over three is simply called an n-gram. After analyzing text using things like word counts and parts of speech, N-grams are the next logical piece of the puzzle [1].

N-grams use the idea of context and patterns to form a probabilistic language model. The context is the provided corpus, and the patterns can be found using conditional probabilities from this corpus. For a bigram model, the model will learn the occurrence of every word and the occurrence of every two words [2]. The occurrence of each word is determined by the number of times that word appears in the corpus divided by the number of words in the corpus, and the occurrence of every two words in calculated by the number of times the two words appear in that order within the corpus divided by the number of times the first word appears in the corpus [1]. The model uses these occurrences to determine the most probable next word in the sequence. Since the probabilities are determined based of the training corpus, it is important that the training body of text is relevant to the application.

Since conditional probability involves multiplication and division, smoothing is necessary. If probability for a word that is not in the training corpus appears in application, the model would throw an error because division by zero isn't possible. Alternately, it might multiply the entire probability by 0 causing the total probability to be zero. In order to protect against this, smoothing is applied. Laplace smoothing is a simple type of smoothing where a one is added to the numerator and the count of the total vocabulary of the corpus, V, is added to the denominator [1]. The one in the numerator is to offset any potential division of 0, and V in the denominator balances it out [1].

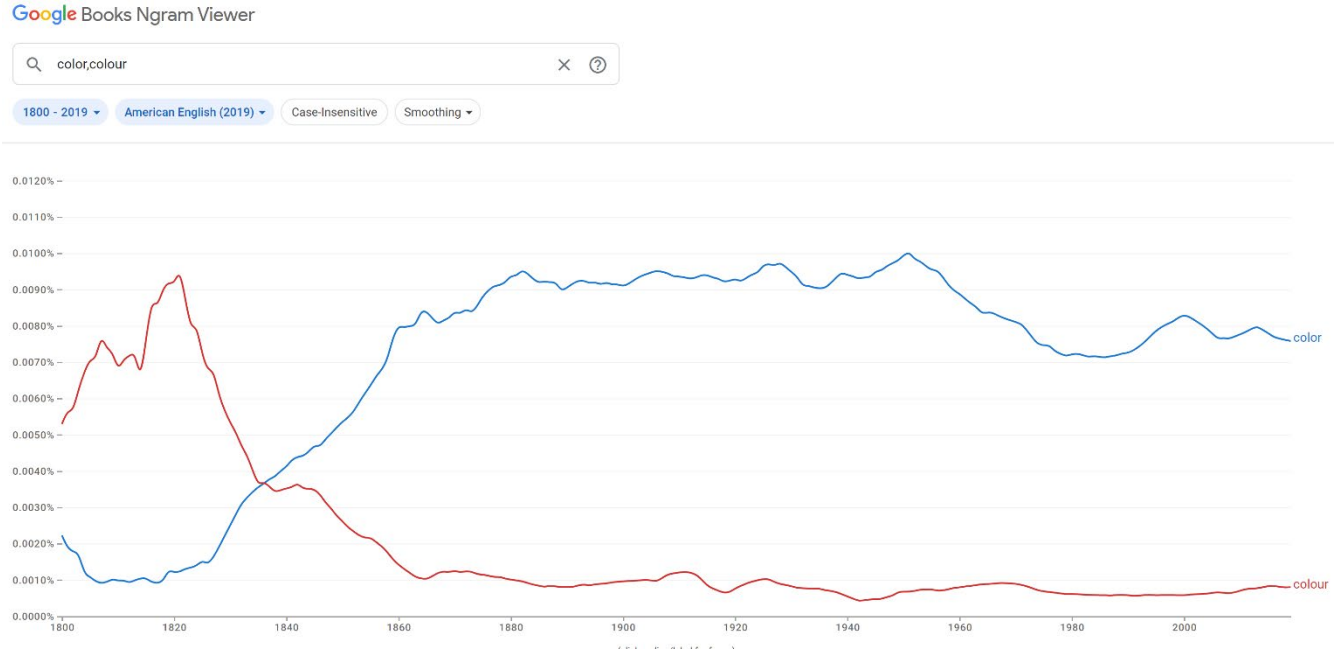
Many use cases for Natural Language Processing, or NLP, are also good use cases for n-grams. N-grams especially thrive, however, in situations where sequences of text need to be analyzed. A common example of this is suggested text [2]. An n-gram model could pick the most likely word to follow given the previous words. N-gram language models can also be useful in spell checkers and personal assistant bots [2]. For instance, if the bot is trained using a trigram model, it can distinguish queries that are three words long like "what's the temperature?" [2]. N-gram models are also used for text generation [1].

An application for text generation works in much the same way as text prediction. The n-grams are converted into probabilities first [1]. Probabilities are calculated for all n-grams from size n down to unigrams. Then the model takes a given word and uses these calculated probabilities to determine the most likely word to follow. These models are limited by the size of the corpus and the value of n. There

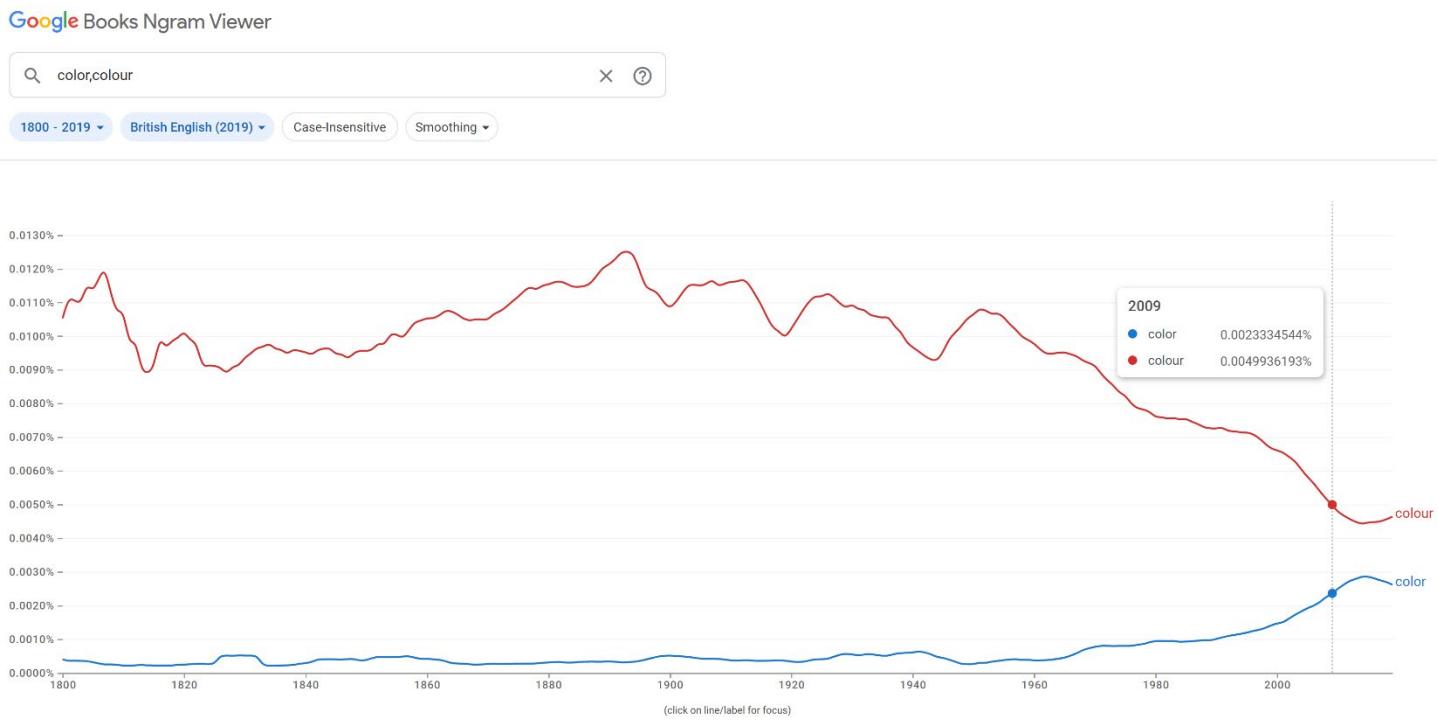
needs to be a lot of data to make a good model, and this data needs to be specific to the application of the language model. Also, higher values of n tend to work better for this type of task since the model accounts for more than just the preceding word. Unfortunately, the models are computationally very expensive.

One way to evaluate language models created using n-grams is through perplexity. Perplexity is a metric that quantifies how well a model predicted text in test data [1]. Usually, a small amount of data is set aside specifically for this task [1]. It is the inverse probability of seeing the observed words normalized by the number of words [1]. It is a branching factor, and a model with low perplexity means that the branching was less chaotic and therefore more favorable [1].

Google has created an n-gram viewer based on literature from 1800 to present day that is available online at https://books.google.com/ngrams/. It was trained on books of all kinds and allows for a user to select spans of various years as well as different languages. Smoothing can also be applied. As an example, two different spellings of color were examined in the American English corpus and case insensitive. It is interesting to see a sharp shift in preferred spellings around the year 1835.



The same data was examined for British English with much different results.



Google's n-gram viewer is an interesting tool to play around with and works blazingly fast.

N-grams are a very useful tool for making language models. They use conditional probabilities to predict word sequences of varying sizes and have a wide range of uses. Many uses of n-grams are relevant to day-to-day life.

References

[1] K. Mazidi, "8. N-gram Models," in Exploring NLP with Python: Building Understanding Through Code, First., 2019, p. 89- 97.

[2] S. Srinidhi, "Understanding word N-grams and n-gram probability in natural language processing," *Medium*, 09-Jan-2020. [Online]. Available: https://towardsdatascience.com/understanding-word-n-grams-and-n-gram-probability-in-natural-language-processing-9d9eef0fa058#:~:text=Well%2C%20in%20Natural%20Language%20Processing,grammar%20in%20a%20given%20sentence. [Accessed: 30-Sep-2022].