N-gram Narrative

N-grams are a method of parsing over a corpus similar to tokenization in the sense that they break down the body into smaller, more easily parsed segments, but the number of words included in each segment is of size n rather than just one (ex. “Sphinx of black quartz, judge my vow” bigram, or n-gram with n=2, would be {Sphinx of, of black, black quartz, …}). To leverage n-grams to construct a language model, you can find the probability of a sequence of words occurring together, calculate these probabilities for all n-grams, and use maximum likelihood estimation for a generative language model. N-grams seem most appropriate for simpler predictive text tasks, for example the predictive suggestions given by your phone’s keyboard or in this Microsoft Word document (typing Microsoft Word automatically suggested document upon entering ‘d’). N-grams also appear useful for serving as components of more advanced techniques; given that it essentially constructs a probability distribution of a given corpus, one could imagine it serving as a prior in the expectation-maximization algorithm.

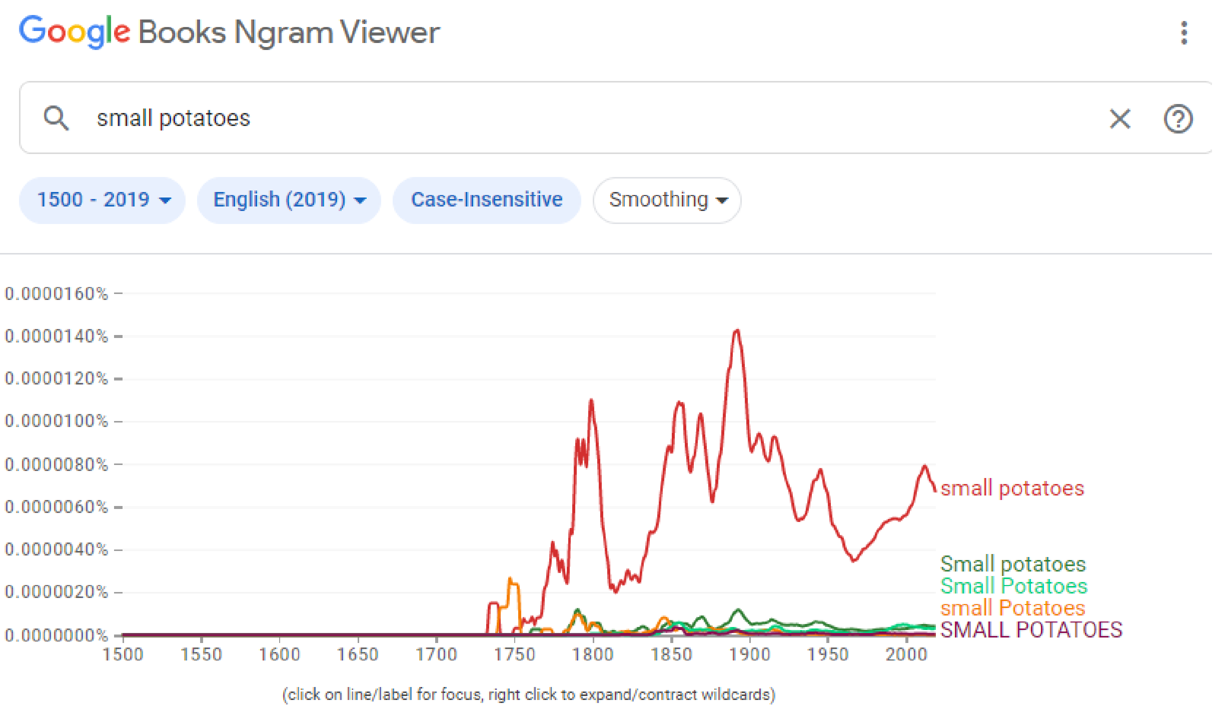
Probabilities are calculated using the product rule of probability- the probability of two non-independent events occurring together is equal to the probability of event one multiplied by the probability of event two given that event one occurred (to simplify the computational requirements on n-grams larger than n=2, the Markov assumption is made — the probability of a word occurring is only dependent on the previous word — despite knowing this is not how language works). These probabilities can be uncovered simply by taking counts of word and n-gram occurrences: the count of a given word divided by the number of tokens in the corpus multiplied by the count of the n-gram of interest divided by the count of the initial given word. Because these probabilities are only uncovered from the given input corpus, it is critical to keep in mind domain specificity and that the input size is sufficient to represent realistic probabilities of words being strung together; a model will only be able to spit out permutations of its input, so if a chatbot engineered solely to give advice about career guidance were asked about cooking advice for example, even if it were to generate some sort of output it would likely be total nonsense, and if one were to train it using a limited body of text it simply could not have enough diversity to generate both meaningful and novel text.

One issue with this method of probabilistic modeling is that for any words that are not seen in the training corpus, their probabilities will be assigned zero, which would make it impossible for the model to respond to any input containing even one word not previously observed. To get around this issue, there are methods of introducing some minimum nonzero probability that acts as a floor value, preventing any probabilities calculated from being strictly zero called smoothing. The simplest method is to simply add one to the count at calculation time, called LaPlace smoothing, but many more complicated methods of smoothing have been written to not much avail.

For the purposes of language generation, probabilistic models are able to capture some sense of the descriptive rules used in fluent speech without having to have them be input explicitly. This allows for the generation of sentences that could feel more natural to a native speaker, but also implies that the models are essentially 'guessing' what should come next. Even if an event has a high likelihood to happen, that does not mean that it will always happen (even if it should), so these types of models can never operate in a perfect manner.

To evaluate the performance of models, the ideal method would be to have human annotators judge and evaluate the output of the model, but this can be both costly in time and money and infeasible if the domain of the NLP application is large enough (e.g. ChatGPT). An intrinsic measure easily calculated from the output of a model is the perplexity: the inverse probability of seeing the given output normalized by the total number of words. Generally, a lower perplexity value is ideal as most of the things people need to communicate uses relatively common and basic language, so the observation of an unusually diverse range of words would indicate that the model could potentially be pulling words out inappropriately.

The Google Books Ngram Viewer is an incredible tool that allows you to examine the popularity of given words or phrases dating back to as early as the 1500's. In it, you can specify the exact year range of interest, as well as which corpus to search over, case sensitivity, and the degree of smoothing preferred.



Something interesting of note in this example is how for a brief period around the 1750's, the capitalization of only potatoes is more popular than any other usage- why?