Andrew Sen

An n-gram is a sliding window of size n over some text. N-grams are obtained by iterating over a text and taking all token sequences of length n. N-grams can be used to create a language model by providing probabilities for what words are most likely to follow a given sequence of words.

One use for n-grams is for text generation. Because n-grams form a probabilistic model for a language, when given a starting word or sentence, n-grams can be used to find the tokens that are most likely to follow that starting sequence. These tokens can be appended to one another to form a computer-generated sequence of text. This can be expanded to other purposes, such as autocomplete, speech recognition, and text compression. Essentially, any use case that can utilize information regarding what words are likely to follow other words can use a language model created via n-grams. Using multiple n-gram models that represent different types of text, one can detect the type of text being read based on which model's n-grams share the highest commonality with the text.

First, a list of unigrams is generated by tokenizing the input text. A list of bigrams is generated by taking all pairs of tokens in the list of unigrams. The probability of a distinct unigram appearing in the text is equal to the number of occurrences of that unigram divided by the total number of tokens in the text. The probability of a bigram appearing in the text is equal to the joint probability of both words in the bigram. This is found by taking the product of the probability of the first word in the bigram appearing and the probability of the second word appearing given that the first word has preceded it.

The source text is extremely important in the generation of a language model because any model based on that text can only reflect the type of language seen in that text. Different languages will of course result in different models, but even different dialects or contexts within the same language can produce drastically different probabilistic models.

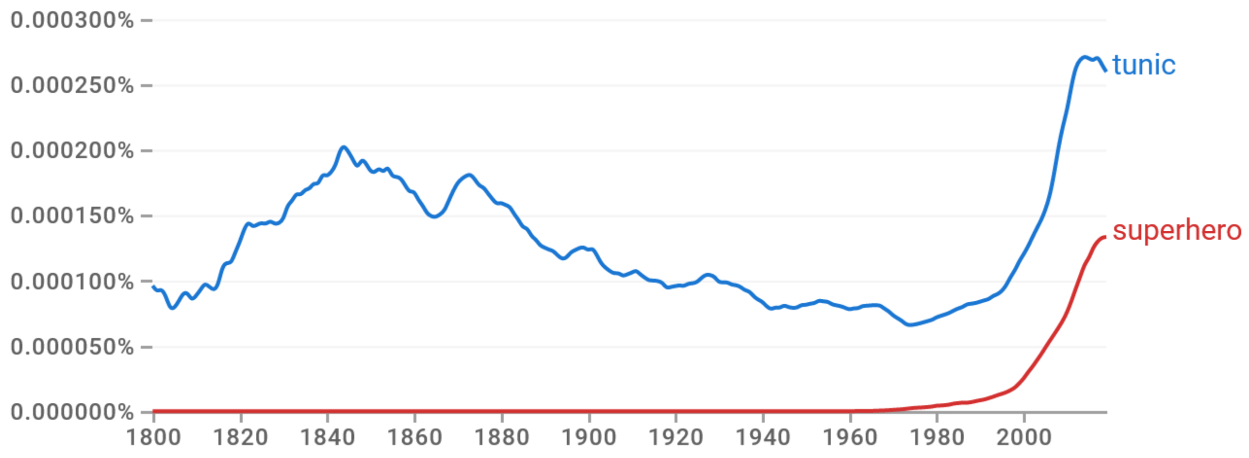
No text will include every possible n-gram, so some n-grams in the model will be given 0 probability of appearing. This is problematic when trying to use a model on novel texts because any unfamiliar sequence of words will cause the model to not be very useful. To make up for this, the data is "smoothed" to give n-grams with 0 probability some probability mass. A simple way of doing this is LaPlace smoothing, where instead of taking the appearance rate of an n-gram, the probability is instead found by adding 1 to the number of appearances and dividing by the total number of unigrams plus the size of the vocabulary.

A language model can be used for text generation by providing a start phrase and then constantly finding what n-gram is most likely to follow that phrase according to the model. This process can be continued until a word-limit has been reached or a certain token was produced, such as a period. The main limitation of this approach is that the resulting text is unlikely to make any sema ntic sense. Because an n-gram model only considers small groups of words, the generated text will mostly look like a series of random words rather coherent sentences.

A language model can be evaluated via extrinsic or intrinsic methods. Extrinsic evaluation involves human annotators manually evaluating a text. Intrinsic evaluation involves an internal metric that can be generated automatically. One example of an intrinsic metric is perplexity. Perplexity is the inverse probability of seeing the observed words normalized by the number of words. A lower perplexity indicates a better model.

Google's Ngram viewer allows the user to enter search phrases and will create a graph showing the frequencies of those phrases in printed sources over long periods of time.

Here's a graph showing the frequencies of "tunic" and "superhero"


tunic
superhero