CS 4395.001

xxxx and Ahmed Iqbal

Portfolio Chapter 8: N-grams Narrative

N-grams are a set of words that occur in succession to each other of a length n. Building a language model using n-grams is relatively simple. You find the probability of each n-gram in a larger body of text, and based on this, you can calculate the probability of the next word that would follow in a sentence.

Some applications of n-grams are language modeling, text completion, spell checking, and information retrieval.

Calculating the probability of a unigram is extremely straightforward. SInce the probability of an n-gram is just the amount of time that sequence appears in a body of text, you just need to count how many times each word appears and divide that by the total number of words in the text to get the probabilities of a unigram. For a bigram, it is a bit more complicated. You need to find the frequency of the bigram, and then divide that by the frequency of the first word of the bigram.

From this, it naturally follows that the source text that you use is extremely important as you will calculate probabilities based on the words in it. Thus, it is imperative that it is large and expansive in terms of the styles of writing in your chosen language. However, it is not possible to make the source text comprehensive of the entire language and all of it's n-grams, which means that when you try to test your language model, you may come across an n-gram that has a probability of zero even though it exists, and thus, clearly does not have a probability of zero.

The method to deal with this is called smoothing, which aims to avoid zero probabilities and reduce the effect that this has on the other probabilities. A simple example of this is Laplace smoothing, or add-1 smoothing. Essentially, you add 1 to the count of the n-gram and the number of unique words in the text to the denominator.

We can use language models for text generation by picking a starting word and then letting the model predict the next word. It will use the previous words in the sentence to predict the word that will come next. One of the major limitations of this approach is that its efficacy largely hinges on how good the training dataset that the model was trained on was. If the data wasn't very good or the model did not have access to a lot of computational resources, the results from the text generation may be lackluster. Another is that they may not have the proper context, because even though they generate text based on the previous words, that may not give the entire context of the sentence.

One way that a language model can be evaluated is by human inspection. If it were able to fool a human into thinking that the generated text was written by a human, it would be something close to passing the Turing test. Another way is called perplexity, which measures how well a model predicts a test dataset. The lower the perplexity score, the better the model is at predicting the data.

Google's N-gram viewer is a very interesting tool that lets users enter an n-gram, which Google then searches for in its massive catalog of books and then shows the user the n-grams frequency by time. It can present this information in the form of a graph, which shows the user the popularity and frequency of usage over the n-gram over time. Beyond being interesting, it is useful for researchers of language that want to study how language and vocabularies evolve over time. The example that I used is the bigram "stormy night", which returned this plot.