Justin Hardy

JEH180008

CS 4395.001

Dr. Mazidi

N-Grams Assignment Overview

I. Overview of N-Grams

In natural language processing, n-grams are sliding windows of text, and can be of any size. Commonly, n-grams are used as Unigrams (1), Bigrams (2), and Trigrams (3), but can be of any size N. In this assignment in particular, we'd used bigrams, and by extension unigrams, to carry out the task at hand.

N-Grams are commonly used to form a probabilistic language model by determining the probability of the sequence of words in the n-gram occurring. This is done by calculating the probability of each word in the sequence given the word(s) preceding it, and multiplying each of those probabilities together. Hence why n-grams are collected and evaluated in that way. In the real world, n-grams are used to create spell checkers & correctors, text compression, and language identification.

II. Overview of Language Models

As previously noted, language models are built off of n-grams. However, a source text is required to not only collect these n-grams, but also to have something to gather counts (for probability calculations) of these n-grams. Having a good source text is essential to building a strong language model, as a source text with low perplexity will help the language model evaluate the language with greater confidence.

One problem that is easy to identify is the fact that not all source text will be able to cover ever single n-gram, meaning that when we're evaluating text using a language model, we're likely to come across n-grams with count of zero in our model. Obviously, multiplying or dividing with zeros will cause the probability to be zero, resulting in computed results being undesirable. This is fixed by smoothing, which is an essential component of language models. It works by filling in these zero values with a small bit of probability mass, in order to fight this "sparsity problem," as described. I'd used Laplace smoothing for this assignment, which adds one to each n-gram count, and divides by the sum of the total number of that type of n-gram, plus the total number of unique n-grams of that type (AKA the vocabulary size).

Language models can be used for text generation, as well. One approach is the naïve approach, which simply takes a start word, and returns the n-gram with the higher probability; it gives the most common continuation to the start word. As one can imagine, text generation approaches work better with larger corpuses and with higher n-grams, but are limited solely by the fact that it is prone to generating text that may essentially be gibberish. Gibberish as in, it's writing coherent sentences, but what is being stated by it makes no sense.

Lastly, language models are often evaluated via a number of metrics. These metrics are either extrinsic, meaning that a predefined metric is used to evaluate the results of the model, or intrinsic, meaning that the results of the model are compared with the results of other models. Particular to intrinsic metrics, perplexity is often used to evaluate how well the model can predict on test data, which is then used to gauge the model's efficacy and confidence.

III. Examining Google's N-Gram Viewer

Google has an interesting feature on their books website that allows users to view the frequency of an inputted n-gram across all of the books they have record of. What's most interesting is that it displays this information by year, so if a particular text didn't exist at that point in time, it is not considered for that year.

In the example below, I inputted a couple of n-grams related to modern computer technology, to see the rise in the frequency of use in the terms in the late 1900s and the 2000s. The results are definitely very interesting, and I personally think it's really cool that google has designed this tool for the general public.