Kevin Bookwalter

Methods and Analysis

The NgramTree class is the core of my methodology for this project. Letter predictions rely on ngram frequencies in my training data, which is a list of 333,333 English words. Frequent character patterns in this data are used to predict new characters. The NgramTree class allows me to store ngrams of any length. For a given word, I can store the frequency of every ngram in that word from n = 1 to n = the length of the word. The tree is structured where each node contains a value and a count, and the depth represents the length of the ngram. The value always contains the last character of an ngram, and the values of the ancestor nodes contain the previous characters in the ngram. The set of child nodes contains the characters that can follow the ngram, sorted by frequency. So, to find the frequency of the trigram “cat”, we traverse the tree from the root through successive child nodes with values “c”, then “a”, then “t”. The final node’s count is the frequency of the trigram “cat”. This could be extended further to find the frequency of “cats” by finding the count of the next child with value “s”.

Character predictions work by finding the 5 most likely next letters based on previous input. Predictions are all relative to the current word that is being typed, and are reset when a new word starts. As new characters are typed, we traverse through the tree down the path of previously typed characters. When we get to the last typed character, it’s 5 most frequent children are predicted as next characters. There is also a way to handle the case where a node does not have 5 children. In this case, the oldest character is cut off of the set of previous characters and we search for the children of this reduced set of previous characters, filling in the set of predictions up to 5. So, if we’re predicting the next character of the pattern “abc” and only get 3 predictions, we then fill in our prediction set with the 2 most frequent children of “bc”. Characters already in the prediction set will not be repeated, and this can continue all the way down to the monogram level if necessary.

There are 7 total files I tested my keyboard on. From the NLTK Gutenberg corpus, I used the King James Bible, Lewis Carroll’s *Alice in Wonderland*, Herman Melville’s *Moby Dick*, and Jane Austen’s *Sense and Sensibility*. I also sourced two other files from the NLTK corpora, including “overheard.txt”, which is transcribed conversations overhead on the street, and “switchboard.txt”, which is transcribed phone conversations. Finally, “testfile.txt” is the test file provided by Dr. McCoy. My dynamic keyboard ranged from 24.29% faster than the static keyboard on “switchboard.txt” to 36.06 % faster on “testfile.txt”, and was 27.47% faster on average. I think a strength of my method is that it worked well across a variety of text genres. This is likely because the training data, which is just a list of words, is genre-less. It is a good general-purpose data set to train on because it isn’t biased toward frequent words or phrases that occur in one specific genre.

My program seemed to perform the best on more plain texts. This is the case with “testfile.txt”, *Moby Dick*, and *Sense and Sensibility.* The language used in all of these texts is fairly simple and plain. My dynamic keyboard was at least 30% more efficient for all of these texts. These texts are all different genres, which indicates that my program uses a good general-purpose prediction model. My program struggled more on texts like the Bible and transcribed conversations. In the case of the Bible, this is likely due to the antiquated language that is used. The King James Bible was published in 1611, so a lot of the words used aren’t common anymore. My dynamic keyboard was 26.64% more efficient than a static keyboard on this text. The performance on transcribed conversations was lower than I expected. I think a significant factor in this is the more informal language that is used. These texts contain more slang and filler words that my program probably struggled to make predictions on. The results for “switchboard.txt” and “overheard.txt” were 24.3% and 25.42% improvements over the static keyboard, respectively.