Sentence Segmentation Program Description

When designing the sentence segmentation program, I first had to determine what makes up a sentence. I knew that every sentence has a subject and a verb, and ends with a terminator, such as a period, question mark, or exclamation point. Ultimately, I decided to focus on using the sentence terminator to divide words into sentences. The approach I took was to write the basic functionality first, and then add in restrictions and extra rules to deal with exceptions later.

In order to separate words into sentences, my program needed access to those words, so I wrote code that would read in lines from an input file, and then add each word into a list, formatting appropriately. I then wrote the basic function of adding words to a sentence until a terminator is reached. Once that happens, the whole sentence is added to a list of sentences to be written to the output file, and the process begins again. Now that the program worked in a rudimentary sense, I needed to make it run correctly in as many different cases as possible.

Since the program looks for a period to mark the end of a sentence, I knew there would be issues with abbreviations, such as “Mr.” and other abbreviations containing periods. I created a list of common abbreviations and upon encountering a word with a period in it, the program checks to see if the word is contained in the abbreviation list before terminating the sentence.

Another problem area I was aware of was concerning numbers with decimals, like “6.50” or similarly formatted numbers. My solution was to check that if a word contains a period, it is located at the last character of the word, otherwise the sentence will not be terminated. This keeps the program from ending the sentence at a period within a number. The one time this check doesn’t happen is if the word also contains the end of a quotation.

Quotes were my biggest obstacle when it came to designing the sentence segmentation program. They pose a challenge because a full quote may be a sentence, or there could be a beginning or ending outside the quotation marks that denotes who is speaking that is also part of the sentence. The solution I devised was to create a Boolean variable that keeps track of if the most recent quotation mark is open or closed. If a word contains a closing quotation mark, some special rules are applied. If the next word is capitalized, I assume it is the beginning of a new sentence. If not, then I assume the sentence does not end with the quote.

There are several limitations to my program that I would have liked to improve upon. Rarely, a sentence will end with an abbreviation, but my program will never end on an abbreviation. I believe my biggest issue to be with quotation marks. If there is a quote followed by a person’s name and the rest of the sentence, the program will split it into two sentences because the person’s name is capitalized. My list of abbreviations is not extensive, and an unknown abbreviation will cause issues. The program is also not designed to handle typos or misspellings in any way, and will not function properly if certain misspellings occur.

I believe the solution to many of these issues is to train a machine to know what a sentence is with a huge dataset, including many instances of unconventional spelling and grammar usage. Another would be to identify the subject and predicate of each sentence. However, more problems would arise from authors breaking the rules of traditional grammar with their writing. Creating this program has shown me that natural language processing is not exact, and can be very difficult or even impossible to get perfectly correct.