Python Spelling Corrector Application

Author: Raza Lamb

To examine how the performance of this spelling corrector, I ran it on the first 131 lines of the corrupted version of Jane Austen's *Sense and Sensibility*. From this output I was able to identify 4 different types of outcomes that the code produced, in comparison to the desired outcome. Below each of these cases are documented, along with a specific example, and if relevant, a potential solution.

Properly Corrected Errors

In this case, the corrector performed exactly as intended: first it identifies an error in the corrupted text, and identifies a potential correction. This correction was compared to the original Sense and Sensibility to confirm whether or not the correction was appropriate. Below are several cases in which the code performed as designed, with the corrupted word on the left and the corrected word on the right.

- "estete" → "estate"
- "thfs" → "this"
- "attacsment" → attachment
- "hveart" → "heart"

Missed Errors

Here, the corrector failed to identify corrupted words, and left them unchanged. Some examples are included below, with the corrupted word on the left, and the correct word on the right.

- "i" → "in"
- "Norlad" → "Norland"
- "nd" → "and"
- "o" → "of"
- "Daswood" → "Dashwood"
- "hi" → "him"

Within this case, there are two subcases: first, cases in which the corrupted word is in the dictionary of English words, so the code does not identify it as a mistake. In this case, in order to correct this behavior, I would need to add a grammatical component to the code, checking to see if the word grammatically fits in the flow of the sentence. Or, I could implement a word predictor that could help us choose the most likely word if there are multiple possibilities of replacement words like there are in this case.

The second case demonstrated above is proper nouns. The corrector is designed to ignore any capitalized words, because the dictionary is unlikely to contain these words, and I would end up doing more harm than good. However, in order to fix this, I would need to implement some form of Named-Entity Recognition (NER).

Improperly Corrected Errors

In this case, the corrector properly identified corrupted words, but did not implement the correct solution. I can identify these cases similarly to how I identified properly corrected errors. Below are several examples, with the incorrect word on the left, the word the corrector replaces it with on the right, and the correct word in parentheses.

- “inheritkr” → “inherited” (“inheritor”)
- “sorid” → “solid” (“sordid”)
- “wifl’s” → “will’s (“wife’s”)

While this error case is less frequent than the others, it still occurs. It mainly appears to occur due to the dictionary that I am using. The dictionary in use is modern, and organized by usage in modern language. My program would be more accurate to use a dictionary representing the english language in the time in which Sense and Sensibility was written. This type of error would also be helped through the addition of a grammatical component to the model, as mentioned earlier.

Error Introduction

This is the most frequent error the corrector produced. The program incorrectly identifies an uncorrupted word, and replaces it with a new word. A few examples are included below, with the uncorrupted word on the left and the word the corrector replaced it with on the right.

- “respectable” → “respective”
- “acquaintance” → “assistance”
- “nephew” → “new”
- “economically” → “economic”

These cases are almost entirely caused by the dictionary in use. Again, this dictionary is modern, while the text is not. Usually a more appropriate dictionary in this setting would resolve most of these errors.

Corner Cases

Having covered the main four categories of changes, last is one special case in which the program does not work as desired. The first example is words which are at the beginning of sentences. The program does not attempt to correct any word that is capitalized, which includes words at the beginning of sentences, in addition to proper nouns. One potential way to correct this would be to build code that can differentiate between proper nouns and words that begin a sentence, such as by building a regular expression that does well at generally recognizing names, or inserting code that recognizes when a word is starting a sentence. However, this proved to be complicated in this instance, because if I use periods as identifiers, the program will correct words that follow “Mrs.”, for example. Additionally, proper nouns can begin a sentence, which adds further complexity.