Spell Checker Program Description

There were several factors I needed to decide on when designing a spell checker. My first obstacle was calculating all the different permutations of a word and the weighted value for that change. I used the cost matrices provided by aclweb.org, read them in as a list of lists, with the indices being the ascii value, and the value being the cost. For each different edit operation, I performed it and if the word was in the dictionary, I calculated the percent chance that it was the correct word based upon the edit distance. I then kept all the correct word permutations for a word in a list, to use later when choosing the correct one.

The next big challenge was deciding which language model to use and implementing it. I decided on using the bigrams provided by the COCA n-gram. I read in the bigrams into a list, with the value, given word, and predicted word all represented. In order to choose the best replacement word, I took the list of correct word permutations for each word and then performed bigram calculations. I checked if the given word matched the previous word from the input text and if it did, then I checked if the current word matched the predicted word. If it did, then I added the chance calculated from the value to the chance the word had from edit distance. I then went through the list and selected the one with the greatest chance of being correct.

This is the basic functionality of the spell checker program, but there were a few other things I was able to do in order to improve efficiency and correctness. There are two possible types of errors: nonword errors and real word errors. I created a different method for fixing each one, since it is much easier to tell if a word is a nonword error. Also with real word errors, there is the chance that the word that the user typed is correct, while that is not the case with nonword errors. Having different methods for each type of error allowed me to be more specific and accurate.

Since I could assume there would be at max one error per sentence, I implemented a counter to keep track of errors. I prioritized fixing nonword errors, since they are much easier to catch.

There are several limitations to my spell checker that I would have liked to fix or improve upon. If a word is capitalized, then the checker automatically assumes it is a proper noun and spelled correctly. Ideally, I would have liked to have a dictionary of names and proper nouns to check against, but I could not find one. The program only calculates an edit distance of one. I would have liked to do two, but it took far too long to run the program. I didn’t properly implement a smoothing function for the noisy channel calculations, so the spell checker is unlikely to select a word it hasn’t seen before.

Writing this program has given me a new respect for automatic spell-checkers, a feature I had previously taken for granted. I have implemented only a few features found in basic word processors or web searches.