Andrei Palonpon, Justin Sacdalan, Yazan Homssi 11 / 05 / 2024
NLP1000

MP4 Post-Analysis

In our lives, we have been using a program which corrects spellings under our nose. This is shown in many places. When typing an analysis paper such as this one on Microsoft Word, we see red curving underlines below the supposed wrong spellings. If we are lazy in correcting spelling and grammar errors by yourself, we can use Grammarly to do it for us. However, in this mini project, we take an in-depth look at such programs by building a simple one ourselves. So, in this post-analysis, we are going discuss our implementation of a simple spelling-correction program. Also, we are going to answer some questions about spelling errors and our implementation's performance.

Our approach is like the one mentioned in the lectures. We first loaded the given text file count_1edit.txt which contains the spellings errors as well as their respective correction edits for our noisy channel model. Afterwards, we print the 10 sample errors to check if it works. Similarly, we downloaded the documents from the Gutenberg corpus as the basis for our language model. Although we could use other datasets, we decided that the Gutenberg corpus was suffice. After the download, we tokenize the words in those texts using nltk.corpus.gutenberg.words() method. Printing some information such as number of distinct tokens is also done for checking. With the extracted tokens, we then created our language model using a set in which for each word a ratio of its frequencies to the total number of tokens is computed. Before we created a noisy channel model, we first had had to prepare some functions is_word_correct(), edits1(), and error_probabilty(). The functions is_word_correct() and error_probability() are to check a validity of a given word and to find the error probability P (w|c). Among the functions, the edits1() function is the most interesting. It was derived from Peter Norvig's tutorial in developing a basic spelling corrector (https://norvig.com/spell-correct.html?fbclid=IwZXh0bgNhZW0CMTEAAR3hiizG-KyHv5YiOXEOJiBc_V5EG-GE0jxB12b3R60cSyZ72fZkpLv7ZOg_aem_Ee445IwIOaI-esMN3r8MWw), and it uses a unique approach to find words with 1 Damerau-Levenshtein edit distance away from the given word. It uses loops specifically list comprehension and slices to go through every possible word created through one of the four edit operations. Those words are stored in a set to be returned at the end. However, we noticed some of the words are not legitimate words, but this would be tackled later in implementing the noisy channel model. Lastly, we create the noisy channel model in the function spell_correct(). Inside the function, it is first checked whether a given word is a correct word. If it is, then the function prints that it is indeed a correct word. Afterwards, we find all the possible words with one edit distance with the edits1() function. However, as we said before, some of them are not real words. So, they are filtered out by checking our language model. As for each legitimate word, we then find its probability in the language model as well as the

probability of the error word given that correct word. We also do not forget to compute the probability of the possible correct word given the error word, P(c | w), by multiplying those two aforementioned probabilities, and lastly store it along with that information in a list. Once we are done with all the real correct candidates, we finally display them along with the edits and the probabilities.

The Gutenberg corpus may not be able to capture more recent vocabulary, so to be able to create a more comprehensive and representative language model, we could supplement the Gutenberg corpus with other text sources, such as web pages, social media platforms like Twitter, scientific publications, etc. The 'count_1edit.txt' file provides a good enough error model, but it may not cover all the types of spelling errors. Our model currently only considers errors within a Damerau-Levenshtein edit distance of 1. We could explore ways to incorporate errors with higher edit distances, maybe by using a larger and more diverse error corpus or by developing more advanced techniques for generating candidate corrections.

What our current model can do is that it is able to handle most spelling errors, which tend to be within a single edit distance. By leveraging the Damerau-Levenshtein distance and the noisy channel approach, our model can effectively identify and correct these types of errors. Although, the model's limitation is that it does not consider errors that involve two or more edits. This means that our model may struggle with more complex spelling mistakes, where the intended word is further removed from the misspelled version. In these cases, the model may fail to generate the correct candidate. To address this limitation, we could find ways to expand the candidate generation process, maybe by using a larger edit distance. We could also explore the use of contextual information, such as the surrounding words in a sentence, to better distinguish between similar-sounding or similar-looking words and identify the most appropriate correction.