Assignment 2

Dhruvin Patel

Spell Corrector

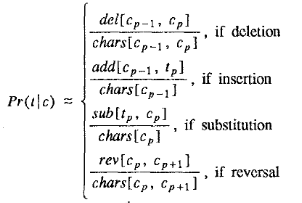
**Methods**

Language Model

For my language model, I chose to use COCA’s bigram corpus. From the frequencies of word pairs in the bigram file, I was able to calculate the unigram and character count for each character in a word. The bigram was made from the count of the first word and second word divided by the first word count. The unigrams were calculated by the number of times a word appears divided by all the word’s count. The character count was saved as the times a character and two characters appeared in the corpus. The language model is used throughout most of the project from checking if a word is spelled correct to getting the probability of a candidate word.

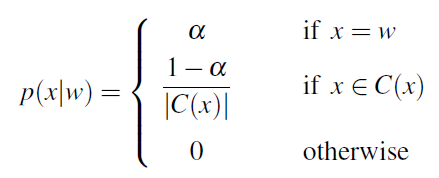
Edit Distance and Cost

To get the candidate list for an incorrect word, I used the edit distance algorithm to find words that are less than 2 edit distance from the incorrect word. The edit distance table is also used to find the changes in the candidate word to the incorrect word by backtracking. Using the changes from the candidate word to the incorrect word, I can calculate the conditional probability. To calculate the conditional probability, I used the same algorithm in *A Spelling Correction Program Based on a Noisy Channel Model*. Depending on the change such as insertion, deletion, substitution, and reversal, the algorithm will get the cost of the characters that were changed. The figure below shows how to calculate the conditional probability of each change to a candidate word. The cost for changing a character comes from the cost matrix that was provided in paper, which I smoothed with 1 plus smoothing.

****

After getting the conditional probability, I use the language model to get the bigram probability of the word previous to the current word and the probability of the word after the current word. I repeat these steps for all of the candidate words. Then, the program chooses the max probability from the candidate list to replace the incorrect spelling word.

Error Model and Real-World Spelling Errors



To fix real-world errors, I generate a candidate set will all the words in the sentence with the word original word include in the candidate set. For each word in the sentence, I replace one word with a candidate word and generate a list of candidate sentences. Using the formula above, if the candidate word is the original word in the sentence then I set the conditional probability to alpha. In my program, I set alpha to .95. Also, for any other candidate word, I take 1-α divide by the size of the candidate size of words then multiple by the conditional probability from incorrect spelling errors. .I assume that there is at least one error in a given sentence; therefore, I did not add the original sentence to the set. The algorithm will then choose the sentence with the highest probability.

**Limitations**

There are many limitations in my implementation. First, my real-world spelling error did not work correctly. It changes words that are already correct in the sentence. This is due to limitation on the dataset I used for my project. For example, the bigram probability of (as, well) is significantly less than the probability of (a, wall). However, if the input was just ‘Until there days’ it will replace there to three. Therefore, the program will try to maximums the sentence probability and choose to replace well to wall. Second, I used 1 plus smoothing in the cost matrices; however, most of the cost matrices have a lot of zeros. This makes it not ideal to use. Good-Turing would be a better implementation.

**Improvements**

If given more time, I believe that using a trigram may have better results on the real-world spelling errors. Also, using Good-Turing to smooth the cost matrices would be better implementation then 1 plus smoothing.