Final Project Report

by Tracy Cui and Tina Tao

Summary:

This project is aiming to add a language model of POS tagging into the SpellChecker program. Our program is able to first detect the incorrect words in the text and then replace the incorrect words with the correct words with the highest probability according to the pos tagging. The text could be both a sentence or a list of sentences.

Reference/Sources:

- We used Python built-in Spell Checker (pyspellchecker) to recognize the incorrect words and to generate a list of possible words for each incorrect word
    - Use "pip install pyspellchecker" to download the package
- Corpora used:
    - We used Brown corpus from NLTK as our training set for probability calculation. This corpus contains lists of list of words with tags
    - We used the Holbrook-tagged.dat corpus as our training set. This corpus contained sentences with incorrect words and their corrections (https://www.dcs.bbk.ac.uk/~roger/corpora.html)

Methods:

Final1.py -- building training model

1. We used Brown corpus from NLTK to train the model and generate Aprob -- a dictionary of dictionaries of transition probabilities, Bprob -- a dictionary of dictionaries of emission probabilities, and Tcount -- a dictionary of all the possible tags. After running the program, final1.py, the training data will be saved as final_model.dat, which can be used for our program directly.

final.py

1. getlines(file) -- this function will read the Holbrook-tagged.dat corpus and remove all unuseful information like <ERR> and punctuations in it. In addition, we also create a dictionary to store the correct words with the key as the sentence index and value as a list of correct words as a reference in the latter part. Thus, this function returns a list of lists of words (text) and a dictionary(errordic).
2. prob(W) -- this function takes a list of words (a sentence) and generates the max probability using Viterbi.
3. findError(sentence) -- this function takes a list of words(a sentence) and generates a list of index which represent the positions of the incorrect words. (In this function, we used the function unknow() in pyspellchecker, which generates all the incorrect words a list of words.)
4. get_possibles(sentence) -- this function takes a list of words (a sentence) and generates a list of lists of words (list of sentences), which includes all the combinations of sentences after replacing the incorrect words with possible correct candidates. (In this function, we used the function candidates() in pyspellchecker, which generate possible correct spellings for one incorrect word.)
5. correction(sentence) -- This function is used to correct words in one sentence. This function takes a list of words(a sentence). It generates the most possible correction for the sentences as a list of words and a list of corrected words according to POS tagging probability. In this function, we called get_possibles(sentence) to generate a list of possible sentences and then for each possible sentence, we called prob(W) to calculate its probability according to POS tagging Viterbi probability. During this process, we will have a variable to keep track of the maximum probability and a list to store the most possible sentence.
6. correction_text(sentences) -- This function is used to correct words in a list of sentences. It takes a list of lists of words (list of sentences) and generates a list of lists of corrected sentences and a dictionary with the key of sentence index and the value of a list of corrected words in each sentence.

7. accuracy(hypothesis, real, wordcount) -- this function is used to test for accuracy for our program. It takes two dictionaries -- one for correct words from the corpus from function getlines(file); one for correct words from our model from function correction_text(sentences) plus one variable wordcount. Wordcount is the number of incorrect words. It will generate a float representing our accuracy rate.

Problems:

1. When we are trying to find the errors in the sentence, there are some words that start with capital letters, and those words might be recognized as incorrect words. In order to deal with this problem, we will only consider words that are formed by lower letters.
2. We didn't know how to generate a combination of all the possible sentences when there are multiple incorrect words in the sentence, because there might be multiple candidates for each incorrect words. In order to solve this problem, we asked for help from Professor Exley and came up with a recursive algorithm, which we replace one incorrect word every round and recurse the function until there are no incorrect words.
3. We noticed that Holbrook-tagged corpus also contains correct words that combine two words together, for example, "some times" will be corrected as "sometimes". We tried to think about possible ways to solve it, but we couldn't find a good way, so we decided to ignore them.
4. We were stuck-up by how to decide our success rate. For calculating the accuracy of our program, we first thought about creating one list for correct reference words from the corpus and another list for the words corrected from our program, but we found out that because of the problem 3, we couldn't recognize all the incorrect words, so the indexes won't match. Because of that, we decided to create two dictionaries where the key is the position of the sentence and the value is a list of corrected words in that sentence. Then, we will compare the two lists with the same index to get our correct words. As for the denominator of the accuracy, we used the number of incorrect words we found using our program instead of the number of the corpus to avoid counting the cases in problem 3.

Final Results:

We calculate the accuracy of our program by dividing the number of matched correct words by the number of recognized incorrect words from our program. From this, we ran the whole text and got the accuracy of 0.18, which is very low. We suspect the possible errors in the Viterbi calculation, and most importantly, there is a significant amount of corrections are combining two words together as we mentioned in problem 3. Also, the accuracy is changing because the correct words coming from the POS tagging probability are different every time. It takes a long time for our program to run through the whole text, which means our program is not efficient enough.

Further Improvements:

1. There are several cases of incorrect words in the testing text that our spell-check model cannot recognize. If the error involves two words (for example, “some” “time” convert to “sometime” or vice versa), or if the word is correct in spelling however grammatically wrong,(eg. go to goes) the spellchecker cannot recognize it. For further improvement, we could consider to include the correction of these words.
2. When selecting the most possible word, we are just considering the word that gets a max Viterbi possibility result for the sentence. However, we notice that for spellchecker, the distance of correction also matters. For further improvement, we could change our way of calculation of prob to make it considers the Viterbi possibility and the distance at the same time.
3. For accuracy calculation, we could also keep track of the best correct words using the pyspellchecker using a dictionary or simply a list, then we can compare our result with the pyspellchecker. In this way, we might be able to get higher accuracy.