LANGUAGE AND COMPUTATION - HOMEWORK 2

INTRODUCTION

In this assignment, you'll implement a spell checker (non-word correction) and test it on some common misspellings. NOTE: the code you will be implementing in Part 2 will likely take several minutes (possibly more) to run. Since you will be testing and re-testing your program using various costs, you should allocate enough time to run the program multiple times.

PART 1 (20 PTS)

Implement the minimum edit distance algorithm using Levenshtein distance. You should make the min edit distance algorithm a function that takes two arguments (the two strings to be compared) and returns the distance; the main body of the code can just run the function taking in two strings from the command line and print the result. Make sure your algorithm works correctly before going on to the next part. Try it on the examples we did in class to make sure you get the same answer. You may find it useful to print out the table so you can check your program's behavior. Save your program as min_edit.py.

PART 2 (20 PTS)

Attached find three text files. One file, *dict.txt*, contains a list of real "words", some of which are actually multiple words like 'a lot'. The second file, *misspellings.txt*, contains a list of common misspelled words. Finally, a third file, *corrections.txt*, contains a list of the misspelled words' intended targets, in the same order. Examine the format of the third file. There are one or more target words separated by ', '. Modify your program (and save as spelling_correction.py) so that it reads in the dict.txt and the mispellings.txt files. It should be run on the command line like this:

```
>>> python spelling_correction.py dict.txt misspellings.txt
```

Your program should go through each of the misspelled words and print out *one* suggested correction followed by "\n". This correction should be one of the words that has the closest Levenshtein distance to the misspelled word. It's up to you how you choose between equally good choices. Save your output to a file. Your file should have one word per line. See TIPS below.

PART 3 (NOTHING TO TURN IN FOR THIS PART)

Find attached a python script called *score.py*. Run this script with the output of your program as the first argument and the *corrections.txt* file as the second argument. This code will print out the accuracy of your spell checker, that is, the proportion of misspellings for which your spell checker proposed one of the words in the target corrections. Make sure your output file is in the right format and you can successfully evaluate the accuracy of your spell checker.

PART 4 (40PTS)

Now for the fun part! Customize the costs used by the min edit distance algorithm any way you like to improve accuracy on the *misspellings.txt* file. I suggest you compare the output of your program to the intended targets and study your program's errors. Are there any regularities in the errors? Are there any common substitutions/deletions/insertions that you could give a lower (or higher) cost to? You may make as many changes as you like, but you should have *at least 5 changes* from the Levenshtein distance. Thus, there must be at least 5 modifications to how distance is computed, which could include pairs of letters that are assigned a different cost than in the original program or more substantial changes to the algorithm. If your improvements don't help, show your work and comment it out.

LANGUAGE AND COMPUTATION - HOMEWORK 2

WHAT TO TURN IN

Turn in your min_edit.py script and your final spelling_correction.py script after finishing Part 4 as attachments, with your name written as a comment at the top of both scripts. If you're not using python 2.7 be sure to write a comment to that effect at the top of your code. Also turn in answers (as a separate attachment (pdf, word, or plain text) to the questions below.

QUESTIONS (20 PTS)

Q1 (4pts)
Show the result when you use your min_edit program to find the distance between the following pairs of words: drive vs. brief, animal vs. mammal, elementary vs. education, python vs. perl

Q2 (2pts)
What is the accuracy of your spell checking script using the Levenshtein distance on the set of misspelled words?

Q3 (4pts)
In order to make the running time on this assignment feasible, I had to make the lexicon (the *dict.txt* file) very small. It has only 3139 words, whereas the lexicon you'll see in future problem sets has more than ten times as many words. How do you think the accuracy would be affected if the lexicon included all the words from the larger lexicon? Why?

Q4 (10pts)
a) What is the best accuracy you were able to get from your spell checker?
b) Explain all the improvements you made to the costs of spell checker. How did you decide on each cost? Were there any changes you made that you expected would help, but they didn't?
c) What changes made the most improvement? Why do you think they helped so much?

TIPS

First, as a sanity check, when you run spelling correction using Levenshtein distance, the accuracy should be in the ballpark of 65%. The exact accuracy you get will depend on some arbitrary choices you make about dealing with ties. For example, when I ran my code and used the alphabetically earliest word with the minimum edit distance, I got around 61% accuracy, whereas when I used the alphabetically last word with the minimum edit distance, I got around 66%.

Make sure you write your min-edit function so that it uses functions that you define for insertion, deletion, and substitution, like the pseudo code in the book does. Notice that the pseudo code uses the insertion and deletion functions in two places, you should do the same. Later in the assignment when you modify the cost functions, the changes in costs will be reflected both in the initialization of the first row and column AND in the construction of the subsequent cells.

Spelling errors happen for various reasons, and some errors depend on the phonological similarity of letters or sounds. If you'd like to try to improve performance by capturing some aspects of phonological similarity, you may find the phonological feature chart linked to below helpful for identifying related sounds:

http://idiom.ucsd.edu/~bakovic/grad-phonology/stuff/pdf/PhonChart_v1102.pdf