11-411/611 Natural Language Processing

Homework 4 (Levenshtein Edit Distance)
Due on Oct 8th, 2020, 11:59 PM (Pittsburgh Time)

Problem Statement

In this assignment, you will write a program (in Python3) to compute the Levenshtein Edit Distance between two strings. A function for Levenshtein Distance is commonly used in Natural Language Processing applications, for example as one of the components of a spell checker.

Specifically, given a list of possibly misspelled words and a list of Dictionary words, you will have to find the word in the Dictionary with the smallest Levenshtein Distance for each element in the first list.

For the purpose of this assignment, in cases where there is a tie in the number of edit's with multiple strings, your program can choose any of them. In the basic setup for Levenshtein Distance, there are three operations that can be performed:

- insert (e.g, *ollege → college*)
- delete (e.g, *collegfe → college*)
- substitute (e.g, *cillege → college*)

You can refer to the Python handout package and implement the TODO code for both parts, details about these functions are written in the code.

For part 2, your function should take two file paths as input and return a list of integer distances:
dictionaryfile.txt - file that contains correctly spelled words. This file will contain multiple lines, one per word.
raw.txt - file that contains misspelled words, one per line.
For example, an element in your return list should be "1" if your misspelled word is "*ollege*" and the dictionary contains "*college*" which has the minimum distance with "*ollege*". You need to compare all the words in the dictionary until you find the minimum distance of the current word in raw.txt. Your return list should have the same length with raw.txt

Your code will be evaluated on the command line as follows:
ed = EditDistance()
ed.calculateLevenshteinDistance("colleg", "college")

student_res = task4("dictionary.txt","raw.txt")

Part 1 (60 Points)

Task 1

Implement Levenshtein Distance and have your program output to the output file (output1.txt) the corresponding word in the dictionary with the smallest Levenshtein distance, along with the distance (see above for output format), for the list of words given in raw.txt. Have your program run this task when the mode argument is 1.

Task 2

In the second task, you will add an additional operation to your Levenshtein Distance function, transposition. Transposition allows swapping between two adjacent characters in the string. For example, the distance between *college* and *ocllege* is only one, since we can swap the first two letters. Depending on how you restrict the allowable transpositions, the resulting distance is either called the Damerau-Levenshtein (DL) or Optimal String Alignment (OSA) distance . For this task, you are required to implement OSA only.

Task 3

You will now implement the Damerau-Levenshtein (DL) distance. The difference between the two algorithms consists in that the optimal string alignment algorithm computes the number of edit operations needed to make the strings equal under the condition that no substring is edited more than once, whereas the second one presents no such restriction.

Take for example the edit distance between CA and ABC. The Damerau–Levenshtein distance LD(CA,ABC) = 2 because CA → AC → ABC, but the optimal string alignment distance OSA(CA,ABC) = 3 because if the operation CA → AC is used, it is not possible to use AC → ABC because that would require the substring to be edited more than once, which is not allowed in OSA, and therefore the shortest sequence of operations is CA → A → AB → ABC.

Part 2 (30 Points)

Task 4

This task is to design your program such that finding the best word(s) in the dictionary is more efficient than the iterative approach of comparing the misspelled word to each of the words in the dictionary. (Hint: redefine the edit distance calculations for a string and a Trie, rather than two strings.) Include all four operations (insertion, deletion, substitution and transposition) in this task. We provide a dictionary of 50,000 Spanish words constructed from Reuters news reports (dictionaryfile.txt). We also provide five hundred possibly misspelled words (raw.txt). You will be given these two files and return a list of these distances. The raw.txt file we use for test will be different in Gradescope.

Submission

editdistance.py (part1)
task4.py(part2)

Do not tar these files and do not hand in your dictionary.txt or raw.txt

FAQ

(1) The cost of substitution to apply for all tasks in this assignment should be 1.
(2) The difference between DL and OSA: DL algorithm allows insertion, substitution, deletion and transposition, with no constraints. OSA allows them with the constraint that no sub-strings are edited more than once.
(3) You can apply any of the feasible algorithms to build up Trie in Task 4. Our test on Gradescope will impose a time limit (180 seconds) on your task 4 to ensure that your code is efficient and you are not solving the problem with brute force. **You will fail the test if your output is wrong or your code times out**.