11-411/611 Natural Language Processing

Homework 4 (Levenshtein Edit Distance)
Due on October 3th, 2019, 11:59 PM

Problem Statement

In this assignment, you will write a program (in Python3 / Java) to compute the Levenshtein Edit Distance between two strings. It 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 than 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 choose either Python or Java handout package and implement the TODO code for both part, 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 has the same length with raw.txt

Your code will be evaluated on the command line as follows:

- Python

ed = EditDistance()
distance = ed.calculateLevenshteinDistance("colleg","college")

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

- Java

```
Editdistance ed = new Editdistance();
int distance = ed.calculateLevenshteinDistance("collgfe","collge");

Optimization opt = new Optimization();
ArrayList<Integer> student_res = opt.task4("dictionary.txt","raw.txt");
```

Part 1 (70 Points)

Task 1

Implement Levenshtein Distance for task 1 including insert, delete, substitute. You should return an integer shows the distance between two strings.

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. Details about Damerau-Levenshtein (DL) distance can be found in Wikipedia.

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 file and return a list of these distances. The raw.txt file we use for test will be different in gradescope.

Submission

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

Java:
Editdistance.java (part 1)
Optimization.java (part 2)

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, and we may manually check your codes for this Task.