CS 5/7391
Special Topics -- Natural Language Processing
Spring 2021

Programming Homework 1

Due: 2/22/21 (Mon) 11:59 pm. No extension

You are to implement the algorithm (using dynamic programming) to implement the editing distance algorithm. The details are as follows:

1. Implement the following function:

   editDistance(str1, str2, deleteCost = 1, defaultSubCost = 2, subCostDict= {})

   The function take two strings, str1, str2, and return the editing distance between the two strings. You should convert the two strings to lowercase before calculating the distance.

   The parameters are as follows:

   - *str1, str2*: the two strings to be compared
   - *deleteCost*: the cost of a deletion/insertion (we assume deletion cost and insertion cost are the same).
   - *defaultSubCost*: the default cost of a substitution (which can be affected by the subCostList dictionary, see below)
   - *subCostDict*: A dictionary storing the substitution cost between two characters. For this dictionary
     - key is a tuple of two characters. You can assume the first character is always before the second character.
     - Value is the cost for substitution for that character.
     - If a pair of characters is not in the subCostDict, then use defaultSubCost.

   For example, consider the following call:

   editDistance('going', 'kick', 2, 3 , d1)

   - *Where d1 = {('a', 'b'): 4, ('b', 'd'): 1, ('c', 'f'): 20, ('c', 'd'): 10, ('g', 'k'): 5.25}*

This will return the editing distance between the string ‘apple’ and ‘ability’, with the following cost:

- Insertion/deletion cost : 2
- Substitution cost : 3, except the following
  - From ‘a’ to ‘b’; and from ‘b’ to ‘a’ : 4
  - From ‘b’ to ‘d’ and from ‘d’ to ‘b’ : 1
  - From ‘c’ to ‘f’, and from ‘f’ to ‘c’: 20
  - From ‘c’ to ‘d’ and from ‘d’ to ‘c’: 10
  - From ‘g’ to ‘k’ and from ‘k’ to ‘g’: 5.25 (notice that non integer cost is allowed)

Then in this case, we can write a recursive formula as this:

Edit_distance(*(‘going’, ‘kick’) = min (Edit_distance(‘goin’, ‘kick’) + 2, Edit_distance(‘going’, ‘kic’) + 2, Edit_distance(‘goin’, ‘kic’ + 5.25))*

(Of course, you are to implement this using dynamic programming)

2. Once you have done the first part. You should implement a class call myDict, which store a list of words, and provide a function that match any given word to those in the dictionary that is ‘close’ (by editing distance). The details:

   Constructor
   - Read in a list of words, representing the words to be stored

   Methods
   - print(): print the list of the words
   - search(str, maxDistance, deleteCost, defaultSubCost, subCostList): return all words in the object that has editing distance <= maxDistance with str. The other parameters are the parameters to be passed to the editDistance() method. You should list the words in alphabetical order.

Notes:

- all words are to be converted to lower case before processing
- any non-alphabetical characters will not be in the subCostList array, and should be treated using default values (i.e. assign default cost for deletion/substitution)

What to hand in:

There will be two python files that is uploaded in Canvas: myDict.py and prog1.py. myDict.py is the file where you need to implement all your methods. Prog1.py is a small program that you can run and test if your program behave correctly.

You will need to upload a copy of the myDict.py that you have modified. You need to include your name in the comment section for the file.

You are to hand in this one and only one file. (I have updated Canvas to ensure only .py file is allowed)

Programming restrictions:

- Your program need to run on python 3.7 or after (preferably 3.8)
- You are only allowed to use numpy (in addition to the default modules that come with the python program).