University of Macau

CISC3025 – Natural Language Processing

Project #1, 2023/2024

(Due date: 5th **February, 2024**)

Project Rule

This is an **individual** course project. You are strongly recommended to commence work on each assignment task of the project soon after it is announced in class/UMMoodle. Students are free to discuss the project, but they are not permitted to share any code and report.

Problem Description

This assignment asks you to implement a *sequence comparison algorithm* (e.g., Levenshtein Distance). Given $x =$ "AACGCA" and $y =$ "GAGCTA", the objective is to match identical subsequences as far as possible through alignment. It can be seen as a way to transforming one sequence into the other with the *substitution*, *insertion*, and *deletion* of characters. The cost of operations is considered as:

- $Sub(a, a) = 0 \; for \; a \in \sum$;
- $Sub(a, b) = 2 \; for \; a, b \in \sum \; and \; a \neq b$;
- $Del(a) = Ins(a) = 1 \; for \; a \in \sum$.

In the following example, three operations are applied for aligning the two sequences, i.e., $Sub(A, G)$, $Del(C)$, and $Ins(T)$. Hence, the minimum cost for such transformation is 4.

$$Aligment = \begin{pmatrix} A & A & C & G & C & - & A \\ G & A & - & G & C & T & A \end{pmatrix}$$

The similarity of two sequences can be defined as the best score among possible alignment between them, i.e. the *minimum cost* or *minimum edit distance*. The computation of such alignment between two sequences can be efficiently solved by using *dynamic programming* approach based on *scoring matrix* (Table 1):

```
DynamicProgramming($x$, $m$, $y$, $n$)

1.  $T[-1,-1] \leftarrow 0$
2.  for $j \leftarrow 0$ to $n - 1$
3.      do $T[-1, j] \leftarrow T[-1, j - 1] + Ins(y_j)$
4.  for $i \leftarrow 0$ to $m - 1$
5.      do $T[i, -1] \leftarrow T[i - 1, -1] + Del(x_i)$
6.          for $j \leftarrow 0$ to $n - 1$
7.              do $T[i, j] \leftarrow min\{ T[i-1, j - 1] + Sub(x_i, y_j)$,
8.                                 $T[i-1, j] + Del(x_i)$,
9.                                 $T[i, j - 1] + Ins(y_j)\}$
10. return $T[m - 1, n - 1]$
```

A	6	↓5	↙4	←↓5	↓4	↙←↓5	↙4
C	5	↓4	↙←↓5	↓4	↙3	←4	←5
G	4	↙3	←↓4	↙3	←↓4	↙←↓5	↙←↓6
C	3	↙←↓4	↓3	↙←↓4	↙3	←4	←↓5
A	2	↙←↓3	↙↓2	↙←↓3	↙←↓4	↙←↓5	↙4
A	1	↙←↓2	↙1	←2	←3	←4	↙←5
#	0	1	2	3	4	5	6
$D(i,j)$	#	G	A	G	C	T	A

Table 1. Scoring matrix

More information regarding *dynamic programming* and *scoring matrix* can be found in Chapter 1 & Chapter 2 of [1] and [2].

Requirements

1. You are asked to implement the *dynamic programming* algorithm in Python. Input to the program are two strings and the *minimum cost* is output as the comparison result, followed by a possible *alignment* between the two strings.

   The following shows a scenario of the input and outputs:

   ```
   > AACGCA
   > GAGCTA
   The cost is: 4

   An possible alignment is:
   A A C G C - A
   | | | | | | |
   G A – G C T A
   ```

2. Extend your program to deal with sentence by taking words as the comparison units instead of letters.

   The following shows a scenario of the input and outputs:

   ```
   > I love natural language processing
   > I really like natural language processing course
   The cost is: 4

   An possible alignment is:
   I love    –   natural language processing   –
   |   |      |       |         |          |       |
   I really like natural language processing course
   ```

3. Write a function to compute the similarities between words in batch mode and store your results in a file.

   In the input file "word_corpus.txt", each row contains a word and a symbol, 'R', or 'H', indicating the correct Reference and the Hypothesis, respectively. Your program compares each hypothesis to the reference, and appends the minimum edit distance to the corresponding hypothesis in the output file, as shown in the following diagram. The number of the hypotheses for each reference may be **varied**. The name of the output file should be "word_edit_distance.txt".

```
Input file:

R satisfaction
H satisfacion
H satesfaction
H satisfation
H satiusfacson
.
.
.
```

```
Output file:

R satisfaction
H satisfacion 1
H satesfaction 2
H satisfation 1
H satiusfacson 4
.
.
.
```

4. Write a similar function to compute the similarities between *sentences* in batch mode "sentence_corpus.txt" and store your results in a file "sentence_edit_distance.txt". The References and Hypotheses are arranged in a similar way as in Requirement (3). Note, the number of hypotheses for each reference is **constant**.

The Starter Code

The starter code is in the edit_distance.py. To make it easier for you to do this project, we provide a starter code written in python. If you enter into the folder "Assignment#1" and execute the following command:

```
$python edit_distance.py -w 'word1' 'word2'
```

The program will execute the function word_edit_distance( ) to calculate the edit distance and the alignment, then output the result to the command line using the output_alignment function( ).

Similarly, you can use the following command to test your implemented sentence_edit_distance( ) function:

```
$python edit_distance.py -s 'sentence1' 'sentence2'
```

For Requirements (3) and (4), you can run the following command to specify the name of input and output files:

```
$python edit_distance.py -bw 'inputfile' 'outputfile'
```

```
$python edit_distance.py -bs 'inputfile' 'outputfile'
```

The output_alignment( ) function has been already implemented to show the alignments in a proper format.

Submissions

You need to submit the following materials:

1. Runnable program and source code;
2. A brief report containing the following contents:
   - **Introduction**: Clearly state the goal of your project. Explain why the project is both important and interesting in the context of NLP.
   - **Background**: Briefly introduce one or two fundamental NLP concepts that are central to your project.
   - **Approach & Challenges**: Summarize your methodological approach in one concise paragraph. Identify one significant challenge you encountered and describe how you addressed it.
   - **Results**: Summarize the outcomes of your project, highlighting the main findings.
   - **Conclusion**: Reflect briefly on what you learned from the project and what was accomplished.
3. The output files.

References

[1] C. Charras and T. Lecroq, *Sequence Comparison*. Université de Rouen. (https://www.researchgate.net/profile/Thierry_Lecroq/publication/2783325_Sequence_Comparison/links/09e415108d23e64eb7000000.pdf)

[2] http://ultrastudio.org/en/Dynamic%20programming%20table