KATHMANDU UNIVERSITY

Dhulikhel, Kavre

COMP 473 Speech and Language Processing

Assignment 1: Minimum Edit Distance

Submitted to:

Dr. Bal Krishna Bal

Associate Professor

Department of Computer Science and Engineering

Submitted by:

Ayush Kumar Shah (44)

Date: 12th April, 2019

1. Fill in the table with minimum edit distance values and backtrace pointers.

n	9	8	9	10	11	12	11	10	9	8
o	8	7	8	9	10	11	10	9	8	9
i	7	6	7	8	9	10	9	8	9	10
t	6	5	6	7	8	9	8	9	10	11
n	5	4	5	6	7	8	9	10	11	10
e	4	3	4	5	6	7	8	9	10	9
t	3	4	5	6	7	8	7	8	9	8
n	2	3	4	5	6	7	8	7	8	7
i	1	2	3	4	5	6	7	6	7	8
#	0	1	2	3	4	5	6	7	8	9
	#	e	x	e	c	u	t	i	o	n

Table 1: Computation of minimum edit distance between intention and execution using Levenshtein distance with cost of 1 for insertions or deletions, 2 for substitutions.

n	9	↓ 8	↙←↓9	↙←↓10	↙←↓11	↙←↓12	↓11	↓ 10	↓9	↙**8**
o	8	↓ 7	↙←↓ 8	↙←↓9	↙←↓10	↙←↓11	↓10	↓9	↙**8**	←9
i	7	↓ 6	↙←↓ 7	↙←↓8	↙←↓9	↙←↓10	↓9	↙**8**	←9	←10
t	6	↓ 5	↙←↓ 6	↙←↓7	↙←↓8	↙←↓9	↙**8**	←9	←10	←↓11
n	5	↓ 4	↙←↓ 5	↙←↓6	↙←↓7	↙←↓**8**	↙←↓9	↙←↓10	↙←↓11	↙↓10
e	4	↙ 3	← 4	↙←**5**	←**6**	←7	←↓8	↙←↓9	↙←↓10	↓9
t	3	↙←↓ 4	↙←↓ **5**	↙←↓6	↙←↓7	↙←↓8	↙7	←↓8	↙←↓9	↓8
n	2	↙←↓ **3**	↙←↓ 4	↙←↓5	↙←↓6	↙←↓7	↙←↓8	↓7	↙←↓8	↙7
i	**1**	↙←↓ 2	↙←↓ 3	↙←↓4	↙←↓5	↙←↓6	↙←↓7	↙6	←7	←8
#	**0**	1	2	3	4	5	6	7	8	9
	#	e	x	e	c	u	t	i	o	n

Table 2: Computation of minimum edit distance between intention and execution with alignment (minimum edit path) using a backtrace, starting at the 8 in the upper-right corner and following the arrows back. The sequence of bold cells represents one possible minimum cost alignment between the two strings.

So, one possible resulting alignment is

i	n	t	e	*	n	t	i	o	n
e	*	x	e	c	u	t	i	o	n

2. Explain the process

Dynamic programming algorithms for sequence comparison work by creating a distance matrix with one column for each symbol in the target sequence and one row for each symbol in the source sequence (i.e., target along the bottom, source along the side). For minimum edit distance, this matrix is the edit-distance matrix. Each cell edit-distance[i,j] contains the distance between the first i characters of the target and the first j characters of the source. Each cell can be computed as a simple function of the surrounding cells; thus starting from the beginning of the matrix it is possible to fill in every entry. The value in each cell is computed by taking the minimum of the three possible paths through the matrix which arrive there:

$$D[i,j] = \min \begin{cases} D[i-1,j]+1 \\ D[i,j-1]+1 \\ D[i-1,j-1]+\begin{cases} 2; & \text{if } source[i] \neq target[j] \\ 0; & \text{if } source[i] = target[j] \end{cases} \end{cases}$$

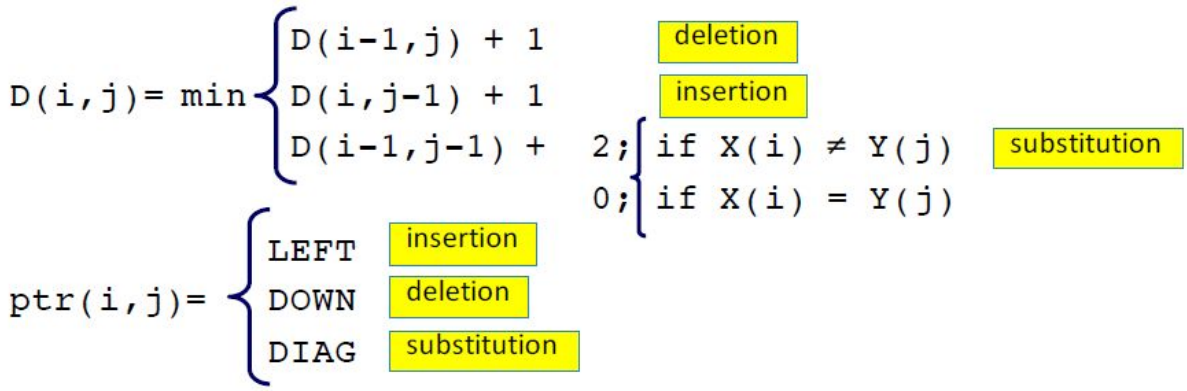
The Table 1 shows the results of applying the algorithm to the distance between intention and execution assuming the version of Levenshtein distance in which the insertions and deletions each have a cost of 1 (ins-cost(·) = del-cost(·) = 1), and substitutions have a cost of 2 (except substitution of identical letters has zero cost).
Example:

D[1,1]=min{D[0,1]+1=2, D[1,0]+1=2, D[0,0]+2=2}=2
D[1,2]=min{D[0,2]+1=3, D[1,1]+1=3, D[0,1]+2=3}=3
….
D[9,9]=min{D[8,9]+1=10, D[9,8]+1=10, D[8,8]+0=8}=8

In order to extend the edit distance algorithm to produce an alignment, we can start by visualizing an alignment as a path through the edit distance matrix. Table 2 shows this path with the boldfaced cell. Each boldfaced cell represents an alignment of a pair of letters in the two strings. If two boldfaced cells occur in the same row, there will be an insertion in going from the source to the target; two boldfaced cells in the same column indicates a deletion.

The computation proceeds in two steps. In the first step, we augment the minimum edit distance algorithm to store backpointers in each cell. The backpointer from a cell points to the previous cell (or cells) that were extended from in entering the current cell. Some cells have multiple

backpointers, because the minimum extension could have come from multiple previous cells. In the second step, we perform a backtrace. In a backtrace, we start from the last cell (at the final Backtrace row and column), and follow the pointers back through the dynamic programming matrix. Each complete path between the final cell and the initial cell is a minimum distance alignment.



Example:

D[1,1]=min{D[0,1]+1=2, D[1,0]+1=2, D[0,0]+2=2}=2
So, backpointers for cell [1,1] are ↙←↓
D[1,2]=min{D[0,2]+1=3, D[1,1]+1=3, D[0,1]+2=3}=3
So, backpointers for cell [1,2] are ↙←↓
….
D[9,9]=min{D[8,9]+1=10, D[9,8]+1=10, D[8,8]+0=8}=8.
So, backpointers for cell [9,9] are ↙