Natural Language Processing Laboratory (CS 753)

Samit Biswas

samit@cs.iiests.ac.in

Department of Computer Science and Technology,
Indian Institute of Engineering Science and Technology, Shibpur

January 16, 2020

Minimum Edit Distance

Minimum Edit Distance

- minimum edit distance between two strings is defined as the minimum number of editing operations needed to transform one into the other.

  The editing operations like:
  - insertion
  - deletion
  - substitution

  Example: Representing the minimum edit distance between two strings as an alignment.

```
I N T E * N T I O N
| | | | | | | | | |
* E X E C U T I O N
d s s   i s
```

The Minimum Edit Distance Algorithm

- Given two strings, the source string X of length n, and target string Y of length m, we'll define $D[i, j]$ as the edit distance between $X[1 \ldots i]$ and $Y[1 \ldots j]$, i.e., the first i characters of X and the first j characters of Y. The edit distance between X and Y is thus $D[n, m]$.

The Minimum Edit Distance Algorithm

- **Dynamic programming:** A tabular computation of $D(n, m)$
- Solving problems by combining solutions to subproblems.
- Bottom-up
  - We compute D(i,j) for small i,j
  - And compute larger D(i,j) based on previously computed smaller values
  - i.e., compute D(i,j) for all i($0 < i < n$) and j ($0 < j < m$).

The Minimum Edit Distance Algorithm

- use dynamic programming to compute $D[n, m]$ bottom up, combining solutions to subproblems.

$$D[i,j] = \min \begin{cases} D[i-1,j] + \text{del-cost}(source[i]) \\ D[i,j-1] + \text{ins-cost}(target[j]) \\ D[i-1,j-1] + \text{sub-cost}(source[i], target[j]) \end{cases}$$

- assume the version of Levenshtein distance in which the insertions and deletions each have a cost of 1 and substitutions have a cost of 2

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

The Minimum Edit Distance Algorithm

function MIN-EDIT-DISTANCE(*source*, *target*) **returns** *min-distance*

$n \leftarrow$ LENGTH(*source*)
$m \leftarrow$ LENGTH(*target*)
Create a distance matrix *distance[n+1,m+1]*

\# *Initialization: the zeroth row and column is the distance from the empty string*
$D[0,0] = 0$
for each row i **from** 1 **to** n **do**
$D[i,0] \leftarrow D[i\text{-}1,0]$ + *del-cost*(*source*[i])
for each column j **from** 1 **to** m **do**
$D[0,j] \leftarrow D[0,j\text{-}1]$ + *ins-cost*(*target*[j])

\# *Recurrence relation:*
for each row i **from** 1 **to** n **do**
for each column j **from** 1 **to** m **do**
$D[i,j] \leftarrow$ MIN($D[i-1,j]$ + *del-cost*(*source*[i]),
$D[i-1,j-1]$ + *sub-cost*(*source*[i], *target*[j]),
$D[i,j-1]$ + *ins-cost*(*target*[j]))

\# *Termination*
return D[n,m]

The Minimum Edit Distance(MED) Algorithm

- Computation of MED between *intention* and *execution*:

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

Computing alignments

- Edit distance isn‘t sufficient
  - We often need to **align** each character of the two strings to each other
- We do this by keeping a “backtrace”
- Every time we enter a cell, remember where we came from
- When we reach the end
  - Trace back the path from the **lower right** corner to read off the alignment

Alignments and edit distance

These two problems reduce to one: find the optimal character alignment between two words (the one with the fewest character changes: the minimum edit distance or **MED**).

- Example:
  if all changes count equally, **MED(stall, table)** is 3

S	T	A	L	L		-
	T	A	L	L		deletion
	T	A	B	L		substitution
	T	A	B	L	E	insertion

- Written as an alignment:

S	T	A	L	L	-
d	|	|	s	|	i
	T	A	B	L	E

More Alignments

- There may be multiple best alignments. In this case, two:

S	T	A	L	L	-	S	T	A	-	L	L
d	|	|	s	|	i	d	|	|	i	|	s
	T	A	B	L	E		T	A	B	L	E

- And lots of non-optimal alignments, such as:

S	T	A	-	L	L
s	s	s	i	s	d
T	A	B	L	E	

Assignments:

1. Given two strings, the source string X of length n, and target string Y of length m, define $D[i,j]$ as the edit distance between $X[1 \dots i]$ and $Y[1 \dots j]$, i.e., the first i characters of X and the first j characters of Y. The edit distance between X and Y is thus $D[n,m]$. Write a program to compute the edit distance, $D[n,m]$ between X and Y using the MED algorithm as discussed in the class.
2. Choose a different path through the backpointers and reconstruct its alignment. How many different optimal alignments are there? Show all. Use your hand-computed results to check your code.

References

- Steven Bird, Ewan Klein, and Edward Loper, “Natural Language Processing with Python”, Published by O‘Reilly Media, Inc.