Natural Language Processing

Paper Code: CS-821/6
UG 8th Sem

Dr. Samit Biswas, *Assistant Professor*

Department of Computer Sc. and Technology

Indian Institute of Engineering Science and Technology, Shibpur

Email: samit@cs.iiests.ac.in

Plan for Today

- Similarity of two Words using Minimum Edit Distance

Minimum Edit Distance (MED)

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
```

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

- If each operation has cost of 1
    - Distance between these is 5.
- If substitutions cost 2 (Levenshtein)
    - Distance between themis 8.

MED Algorithm

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 . . . i]* and *Y[1 . . . 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.*

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

MED Algorithm

The Minimum Edit Distance Algorithm

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

  n ← LENGTH(source)
  m ← 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] ← D[i-1,0] + del-cost(source[i])
      for each column j from 1 to m do
          D[0,j] ← D[0,j-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] ← 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]
```

MED Table

The Minimum Edit Distance Table

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

N	9									
O	8									
I	7									
T	6									
N	5									
E	4									
T	3									
N	2									
I	1									
#	0	1	2	3	4	5	6	7	8	9
	#	E	X	E	C	U	T	I	O	N

The Minimum Edit Distance Table

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

N	9									
O	8									
I	7									
T	6									
N	5									
E	4									
T	3									
N	2									
I	1									
#	0	1	2	3	4	5	6	7	8	9
	#	E	X	E	C	U	T	I	O	N

$$D(i,j) = \min \begin{cases} D(i-1,j) + 1 \\ D(i,j-1) + 1 \\ D(i-1,j-1) + \begin{cases} 2; & \text{if } S_1(i) \neq S_2(j) \\ 0; & \text{if } S_1(i) = S_2(j) \end{cases} \end{cases}$$

MED Table

The Minimum Edit Distance Table

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

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

MED Algorithm - Backtrace for Computing Alignments

Backtrace for 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
- An optimal alignment is composed of optimal subalignments

MED Algorithm - Backtrace for Computing Alignments

Backtrace for Computing alignments

Base conditions:

$D(i,0) = i \qquad D(0,j) = j$

Termination:

$D(N,M)$ is distance

Recurrence Relation:

For each $i = 1 \ldots M$

For each $j = 1 \ldots N$

$$D(i,j) = \min \begin{cases} D(i-1,j) + 1 & \boxed{\text{deletion}} \\ D(i,j-1) + 1 & \boxed{\text{insertion}} \\ D(i-1,j-1) + \begin{cases} 2; & \text{if } X(i) \neq Y(j) \\ 0; & \text{if } X(i) = Y(j) \end{cases} & \boxed{\text{substitution}} \end{cases}$$

$$ptr(i,j) = \begin{cases} \text{LEFT} & \boxed{\text{insertion}} \\ \text{DOWN} & \boxed{\text{deletion}} \\ \text{DIAG} & \boxed{\text{substitution}} \end{cases}$$

Backtrace for Computing alignments

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**

References

References

- Daniel Jurafsky & James H. Martin, “*Speech and Language Processing*”, Prentice Hall
- Diana Maynard, Kalina Bontcheva, Isabelle Augenstein, “*Natural Language Processing for the Semantic Web*”, A Publication in the Morgan & Claypool Publishers series.
- Steven Bird, Ewan Klein and Edward Loper, “*Natural Language Processing with Python*”, By O‘Reilly.
- Christopher Manning and Hinrich Schtze, “*Foundations of Statistical Natural Language Processing*”, The MIT Press.

Thank You

Contacts:

Dr. Samit Biswas
Department of CST,
IIEST, Shibpur

samit@cs.iiests.ac.in
https://www.iiests.ac.in/IIEST/Faculty/cs-samit