Minimum edit distance. - Victor Kitov

# Minimum edit distance.<sup>1</sup>

Victor Kitov

v.v.kitov@yandex.ru

<sup>&</sup>lt;sup>1</sup>With materials used from "Speech and Language Processing", D. Jurafsky and J. H. Martin.

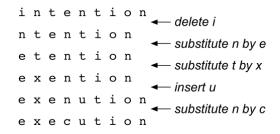
#### Introduction

- Need to estimate distances between strings
  - error correction:
    - user typed graffe
    - probably he meant giraffe
  - named entity recognition
    - Stanford President John Hennessy
    - Stanford University President John Hennessy
- Minimum edit distance between two strings the minimum number of editing operations (insertion, deletion, substitution) needed to transform one string into another.
  - each editing operation has cost 1
  - however we may assign different costs

### Example

Distance from [intention] to [execution] is 5.

• Optimal (minimum loss) conversion path:



- Optimzal path is found with dynamic programming.
- Main idea of dynamic programming: if path X → Y is optimal and is passes through Z then path X → Z should also be optimal (otherwise original path can be imporved).

## Definitions

Define

• X-input string, Y-target string.

• 
$$len(X) = n$$
,  $len(Y) = m$ 

- X[1..i]-substring, consisting of fisrt i X symbols.
- D(i,j) distance between X[1...i] and Y[1...j]
- Then distance between X and Y is D(n, m)

## Minimum edit distance calculation

- D(0,j) = cost of j instertions
- D(i,0) = cost of i deletions
- Recurrent recalculation:

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

#### Demo

	#	e	X	e	c	u	t	i	0	n
#	0	1	2	3	4	5	6	7	8	9
i	1	™<+↑ 2	⊼,←↑ 3	⊼,←↑ 4	⊼,←↑ 5	⊼,←↑ 6	<u>⊼</u> ←↑ 7	べ 6	← 7	$\leftarrow 8$
n	2	⊼,←↑ <b>3</b>	<u>∿</u> ←↑4	⊼,←↑ 5	⊼,←↑ 6	<u>∿</u> ,⊷† 7	<u>⊼</u> ←↑ 8	<u>↑</u> 7	⊼,←↑ 8	乀7
t	3	<u>∿</u> , ←† 4	⊼, <b>←</b> ↑ <b>5</b>	<u>⊼</u> ←↑ 6	⊼,←↑ 7	<u>⊼</u> ,←↑ 8	<u> </u>	$\leftarrow \uparrow 8$	⊼,←↑ 9	$\uparrow 8$
e	4	<b>べ 3</b>	← 4	⊼, <b>⊢ 5</b>	← <b>6</b>	← 7	$\leftarrow \uparrow 8$	⊼,←↑ 9	∿, ←↑ 10	↑ <b>9</b>
n	5	↑ 4	<u>∽</u> ←↑ 5	<u>∿</u> ←↑6	<u> ≺</u> ←↑ 7	⊼,⊷↑ <b>8</b>	<u>∿</u> ←↑9	∿~ 10	∿,←↑ 11	<u>∖</u> † 10
t	6	↑ <b>5</b>	<u>∿</u> ←↑6	<u> ≺</u> ←↑ 7	<u>⊼</u> ←↑ 8	<u>∿</u> ←↑ 9	<b>べ 8</b>	$\leftarrow 9$	← 10	← <u>↑</u> 11
i	7	↑ <b>6</b>	<u>~</u> ←↑ 7	<u>⊼</u> ←↑ 8	<u>~</u> ←↑9	∿~ 10	↑ <b>9</b>	K 8	$\leftarrow 9$	$\leftarrow 10$
0	8	↑ 7	$\stackrel{\scriptstyle \scriptstyle \nwarrow}{} \leftarrow \uparrow 8$	<u>∽</u> , ←↑ 9	∿, ←↑ 10	∿(→ 11	↑ 10	↑ <b>9</b>	× 8	$\leftarrow 9$
n	9	$\uparrow 8$	${\rm Ker}9$	<u>∽</u> ←↑ 10	∿,←↑ 11	∿,←↑ 12	↑ 11	↑ 10	↑ <b>9</b>	× 8

## Optimal path

- We can find optimal transformations path by
  - storing the optimal steps in each D(i,j) recalculation
  - after D(n,m) is found, backtrace the optimal path
- Optimal path is equivalent to alginement of 2 strings:

```
INTE*NTION
| | | | | | | | | |
*EXECUTION
dss is
```

d=deletion, s=substitution, i=insertion.