Minimum edit distance.[1]

Victor Kitov

v.v.kitov@yandex.ru

[1] 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:

```
i n t e n t i o n
                    ← delete i
n t e n t i o n
                    ← substitute n by e
e t e n t i o n
                    ← substitute t by x
e x e n t i o n
                    ← insert u
e x e n u t i o n
                    ← substitute n by c
e x e c u t i o n
```

- Optimzal path is found with dynamic programming.
- Main idea of dynamic programming: if path $X \to Y$ is optimal and is passes through Z then path $X \to Z$ 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	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**

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:

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

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