Iran University of Science and Technology, Computer Department

NLP Homework 2, Part 4

Vahid Kharazi

November 27, 2015

1 Support Transposiotion

Levenshtein edit distance algorithm can support insert, delete and substitution.

$$\operatorname{lev}_{a,b}(i,j) = \begin{cases} \max(i,j) & \text{if } \min(i,j) = 0, \\ \min \begin{cases} \operatorname{lev}_{a,b}(i-1,j) + 1 \\ \operatorname{lev}_{a,b}(i,j-1) + 1 \\ \operatorname{lev}_{a,b}(i-1,j-1) + 1_{(a_i \neq b_j)} \end{cases} & \text{otherwise.} \end{cases}$$

By adding a simple rule after above equation, it can support transposition:

```
D[i][j] = min(D[i][j], D[i-2][j-1]) + cost  if(a[i-1]==b[j-2] and a[i-2]==b[j-1])
```

For the space complexity we need to store a $n * m$ matrix($O(nm)$). We add an operation(check, minimum and update) after basic Levenshtein algorithm. since we have direct access to matrix we don't have any change in time complexity($O(nm)$).

There is three possible minimum edit distance:

1. If we can convert $a[1:i]$ to $b[1:j-1]$ by k operation, so we will can simply convert it to $b[1:j]$ with $k+1$ operations.

2. If we can convert $a[1:i-1]$ to $b[1:j]$ by k operation, so we will can do it on $a[1:i-1]$ and them delete $a[i]$($k+1$ operations, delete is one).

3. If we can convert $a[1:i-1]$ to $b[1:j-1]$ by k operation, so we will can do substitute $a[1:i]$ to $b[1:j](k+2$ operations, 2 is cost of substitution).

For the added rule we have:

1. If we can convert $a[1:i-1]$ to $b[1:j-2]$ or $a[1:i-2]$ to $b[1:j-1]$ by k operation, so we can do it with two substitution(cost of transposition).

The numbers of operation to convert $a[1:1-m]$ to $b[1:n-1]$. so we can say $D[m][n]$ is the final result.