The Levenshtein (Edit) Distance and WER

The Definition

The WER is the Levenshtein distance of words.

Levenshtein distance is the number of edits (insertion, deletion, and substitution) we need to change string s into string t.

If the elements of strings are letters, the distance is often known as the *edit distance*.

If elements are words, the distance normalized by the number of words in the source string is known as *WER*.

The Properties

The range of Levenshtein distance is from 0 to ∞.

According to Wikipedia, when used as WER, the source string is the ground-truth, and the destination is the hypothesis.

The Calculation

The Levenshtein distance can be computed using a dynamic programming algorithm.

Consider that $s = [s^-, x]$ and $t = [t^-, y]$, the Levenshtein distance between $d^- = D\{s^-; t^-\}$ is the number of edits we need to make s^- to t^-. Basing on those edits, we can go on changing s into t in either of the following ways:

1. delete x from s and insert y
2. insert y into s and delete x
3. replace x by y, if $x \neq y$
4. nothing to do, if $x = y$

Where 1. introduces sums the weights of a deletion and an insertion to d^-; 2. does the same; 3. sums the weight of substitution; 4. adds zero. Anyway, we'd like to choose the minimum number of edits we need from these four cases, so we have

$$D\{s; t\} = \min\left(D\{s, t^-\} + w_I, D\{s^-; t\} + w_D, D\{s^-; t^-\} + \delta_{x;y} w_S\right)$$

where w_I, w_D, and w_S are the cost of insertion, deletion, and substitution; $\delta_{x;y} = 1$ if $x \neq y$, or 0 if $x = y$.

Boundaries cases

1. If s is empty (ϕ), $D\{\phi, t\} = |t|$ because we need to insert each character in t.
2. If t is empty (ϕ), $D\{s, \phi\} = |s|$ because we need to delete each character in s.

The Algorithm.

These boundary cases inspire us to fill in a $(|s| + 1) \times (|t| + 1)$ matrix, where each cell i, j saves the value $D\{s_{0:i}; t_{0:j}\}$. The upper-left cell is $D\{s_{0:0}; t_{0:0}\} = D\{\phi; \phi\} = 0$.

The algorithm is a two-level nested loop: the outer go through the diagonal, the inner one go through the current column and row to fill the matrix cells.

After the nested loop, we will see $D\{s; t\}$ in the bottom-right cell.

For more details about this algorith, please refer to https://people.cs.pitt.edu/~kirk/cs1501/Pruhs/Spring20