Basics of Spelling Correction

Edit Distance and Levenshtein Distance

Dr. Sambit Praharaj, Assistant Professor (II), KIIT

1 Introduction to Spelling Correction

Spelling correction is the task of identifying and correcting misspelled words in text. It is widely used in spell checkers, search engines, and Natural Language Processing (NLP) systems.

Examples:

- speling $\rightarrow$ spelling
- recieve $\rightarrow$ receive

A fundamental idea behind spelling correction is that the correct word is usually the one *closest* to the misspelled word. This closeness is measured using **Edit Distance**.

2 Edit Distance

Edit Distance is defined as the minimum number of operations required to transform one string into another.

The allowed operations are:

1. **Insertion**: inserting a character
2. **Deletion**: deleting a character
3. **Substitution**: replacing one character with another

Among various edit-distance measures, the most commonly used is the **Levenshtein Distance**.

3 Levenshtein Distance

The Levenshtein Distance between two strings is the minimum number of insertions, deletions, and substitutions needed to convert one string into another.

Example:

$$\text{cat} \rightarrow \text{cut}$$

Only one substitution ($a \rightarrow u$) is required, so the distance is 1.

4 Dynamic Programming Formulation

Let:

$$dp[i][j] = \text{minimum edit distance between the first } i \text{ characters of word1}$$

$$\text{and the first } j \text{ characters of word2}$$

Base Cases

$$dp[0][j] = j \quad \text{(insert } j \text{ characters)}$$
$$dp[i][0] = i \quad \text{(delete } i \text{ characters)}$$

Recurrence Relation

If the characters match:

$$dp[i][j] = dp[i-1][j-1]$$

If the characters differ:

$$dp[i][j] = 1 + \min \begin{cases} dp[i-1][j] & \text{(deletion)} \\ dp[i][j-1] & \text{(insertion)} \\ dp[i-1][j-1] & \text{(substitution)} \end{cases}$$

5 Step-by-Step Numerical Example

Convert:

$$\text{kitten} \to \text{sitting}$$

Length of “kitten” = 6, length of “sitting” = 7.
The DP table size is $(6+1) \times (7+1)$.

DP Table Initialization

	∅	s	i	t	t	i	n	g
∅	0	1	2	3	4	5	6	7
k	1							
i	2							
t	3							
t	4							
e	5							
n	6							

Key Cell Computations

Cell (1,1): Convert `k` → `s`

$$dp[1][1] = 1 + \min(1, 1, 0) = 1$$

Cell (2,2): Convert `i` → `i` (match)

$$dp[2][2] = dp[1][1] = 1$$

Cell (3,3): Convert `t` → `t` (match)

$$dp[3][3] = dp[2][2] = 1$$

Cell (5,5): Convert `e` → `i`

$$dp[5][5] = 1 + \min(2, 2, 1) = 2$$

Cell (6,7): Convert `n` → `g`

$$dp[6][7] = 1 + \min(4, 2, 3) = 3$$

Final DP Table

	∅	s	i	t	t	i	n	g
∅	0	1	2	3	4	5	6	7
k	1	1	2	3	4	5	6	7
i	2	2	1	2	3	4	5	6
t	3	3	2	1	2	3	4	5
t	4	4	3	2	1	2	3	4
e	5	5	4	3	2	2	3	4
n	6	6	5	4	3	3	2	**3**

6 Final Result

The Levenshtein Distance between **kitten** and **sitting** is:

$$\boxed{3}$$

Interpretation

The optimal sequence of edits is:

1. Substitute `k` $\rightarrow$ `s`
2. Substitute `e` $\rightarrow$ `i`
3. Insert `g` at the end

7 Why Levenshtein Distance is Useful?

- Models common human spelling errors
- Simple and mathematically well-defined
- Widely used in spell checkers and NLP applications

8 Limitations

- All operations have equal cost
- Cannot handle transpositions efficiently
- Slow for large dictionaries without optimization

9 Conclusion

Levenshtein Distance provides a simple and effective way to perform spelling correction using dynamic programming. The final edit distance is obtained from the bottom-right cell of the DP table.