Distance metrics

(AI notes)

Distance metrics are used in supervised and unsupervised learning to calculate similarity in data points. They improve the performance, whether that is for classification tasks or clustering. The four types of distance metrics are *Euclidean Distance*, *Manhattan Distance*, *Minkowski Distance* and *Hamming Distance*.

Hamming Distance

- between two integers: the number of bits that are different at the same position in both numbers.

Example:

```
Input: n1 = 9, n2 = 14
Output: 3

9 = 1001, 14 = 1110

No. of Different bits = 3

Input: n1 = 4, n2 = 8

Output: 2
```

- between two strings (of equal length): the number of positions at which the corresponding character is different.

Example:

```
Input : str1[] = "geeksforgeeks", str2[] = "geeksandgeeks"
Output : 3
Explanation : The corresponding character mismatch are
highlighted.
"geeksforgeeks" and "geeksandgeeks"

Input : str1[] = "1011101", str2[] = "1001001"
Output : 2
Explanation : The corresponding character mismatch are
highlighted.
"1011101" and "1001001"
```

Jaro Similarity

- is the measure of similarity between two strings;
- the value of Jaro distance ranges between 0 and 1, where 1 means the strings are equal and 0 means no similarity between the two strings.

Example:

```
Input: s1 = “CRATE”, s2 = “TRACE”;
Output: Jaro Similarity = 0.733333
```

```
Input: s1 = “DwAyNE”, s2 = “DuANE”;
Output: Jaro Similarity = 0.822222
```

$$Jaro\ similarity\ = \begin{cases} 0,\ \text{if m=0} \\ \frac{1}{3}\left(\frac{m}{|s1|} + \frac{m}{|s2|} + \frac{m-t}{m} \right),\ \text{for m!=0} \end{cases}$$

m - the number of matching characters

t - half the number of transpositions

|s1| and **|s2|** - the lengths of strings s1 and s2 respectively.

- **Jaro-Winkler Similarity**

- is a string metric measuring edit distance between two strings;
- differs from Jaro Similarity when the prefix of the strings matches;
- uses a prefix scale **‘p’** which gives a more accurate answer when the strings have a common prefix up to a defined maximum length l.

Example:

```
Input: s1 = “DwAyNE”, s2 = “DuANE”;
Output: Jaro-Winkler Similarity =0.84
```

```
Input: s1=”TRATE”, s2=”TRACE”;
Output: Jaro-Winkler similarity = 0.906667
```

Sw = Sj + P * L * (1 - Sj)

Sj - jaro similarity

Sw - jaro-winkler similarity

P - the scaling factor (0.1 by default)

L - the length of the matching prefix up to a maximum of 4 characters

```
s1=”arnab”, s2=”aranb”.
```

The Jaro Similarity of the two strings is **0.9(3)**.

The length of the matching prefix is 2 and we take the scaling factor as 0.1.

=> The Jaro-Winkler Similarity: 0.9(3) + 0.1 * 2 * (1 - 0.9(3)) = **0.946667.**

- **Levenshtein Distance**

- is a measure of the similarity between two strings, which takes into account the number of *insertion*, *deletion* and *substitution* operations needed to transform one string into the other.

Example:

Let’s see an example that there is **String A: “kitten”** which need to be converted in **String B: “sitting”** so we need to determine the minimum operation required

kitten → sitten (substitution of “s” for “k”)

sitten → sittin (substitution of “i” for e”)

sittin → sitting (insertion of “g” at the end).

In this case it took three operations to do this, so the levenshtein distance will be 3.

Applications of Levenshtein Distance:

- Autocorrect Algorithms;
- Data cleaning;
- Data clustering and classification.

- **Longest Common Subsequence (LCS) Distance**

- is the minimum cost of operations (insertions and deletions) required to transform str1 into str2.

Ground truth - a reference source or a correct data set with which the results are compared.

Bibliography:

https://www.geeksforgeeks.org/hamming-distance-between-two-integers/
https://www.geeksforgeeks.org/hamming-distance-two-strings/
https://www.geeksforgeeks.org/jaro-and-jaro-winkler-similarity/
https://www.geeksforgeeks.org/introduction-to-levenshtein-distance/