

Lecture :Similarity measures for textual data

Jaro Distance
N gram Distances
LCS Distance

Date: Monday _09/10/2023

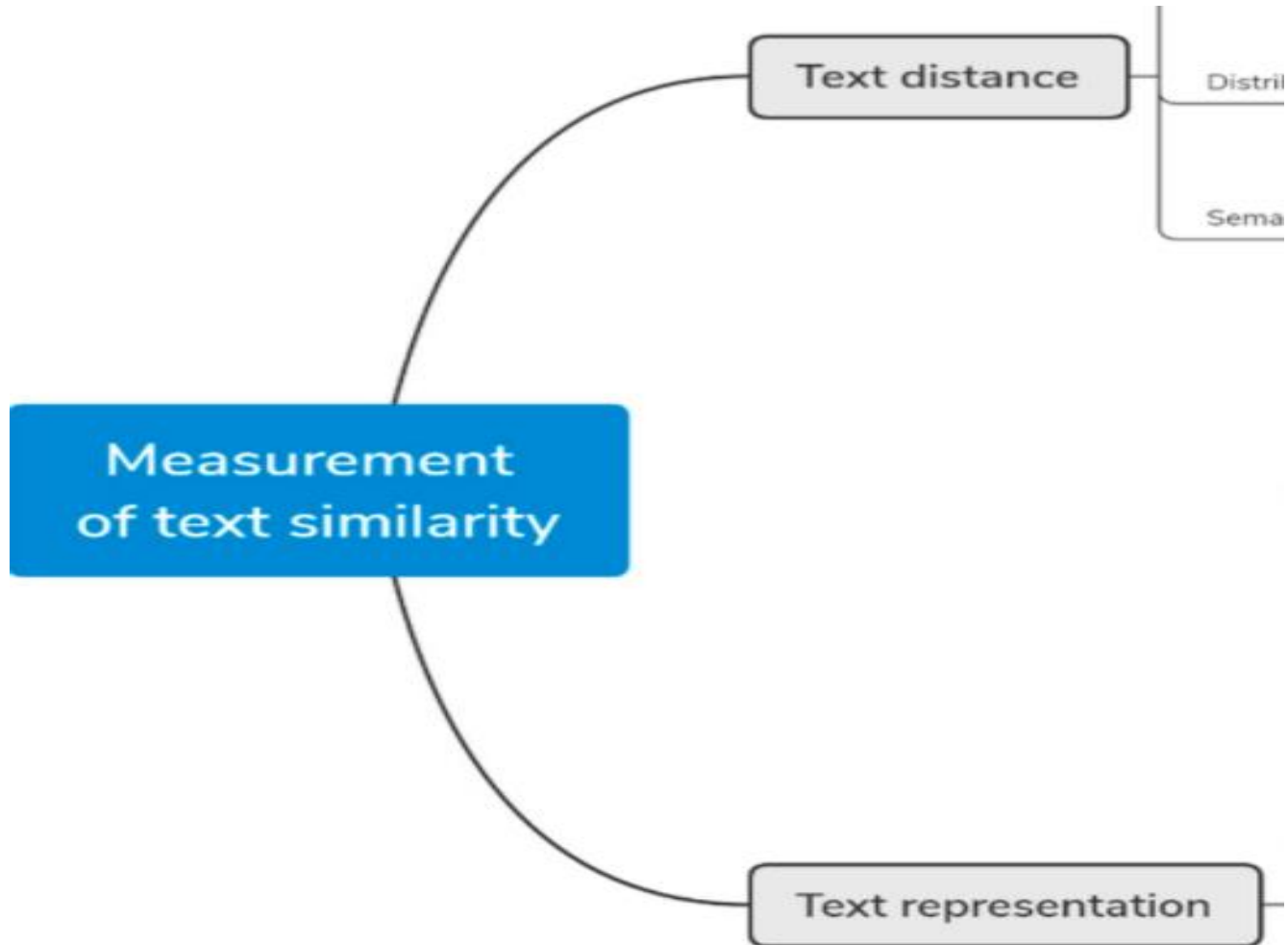
Similarity features of Text

Algorithm	Comparison by	Case Sensitive	Sequence Matters?	Result
Cosine Similarity / Distance	word	✓	✗	Number (0 to 1)
Hamming Distance (Same length input strings only)	character	✓	✓	Count of substitutions
Jaccard Similarity / Distance	character	✓	✗	Number (0 to 1)
Levenshtein Distance	character	✓	✓	Count of edits
Longest Common Subsequence	character	✓	✓	Common String

Text Similarity and computations

- Text similarity has to determine how ‘close’ two pieces of text are both in surface closeness [**lexical similarity**] and meaning [**semantic similarity**].
- The distance measures from one text to another text.

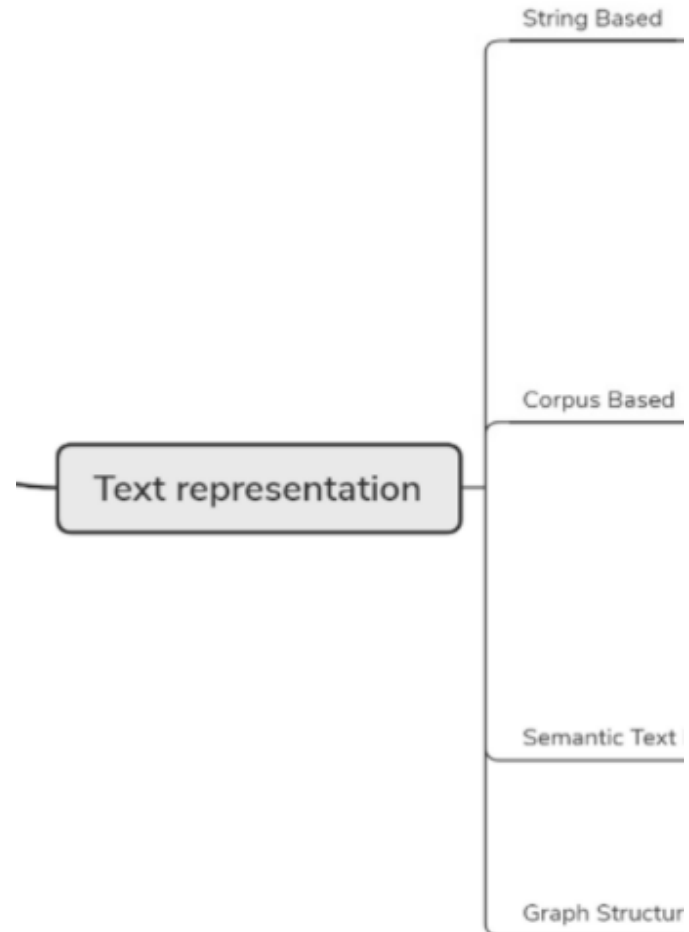
Similarity measures for textual data



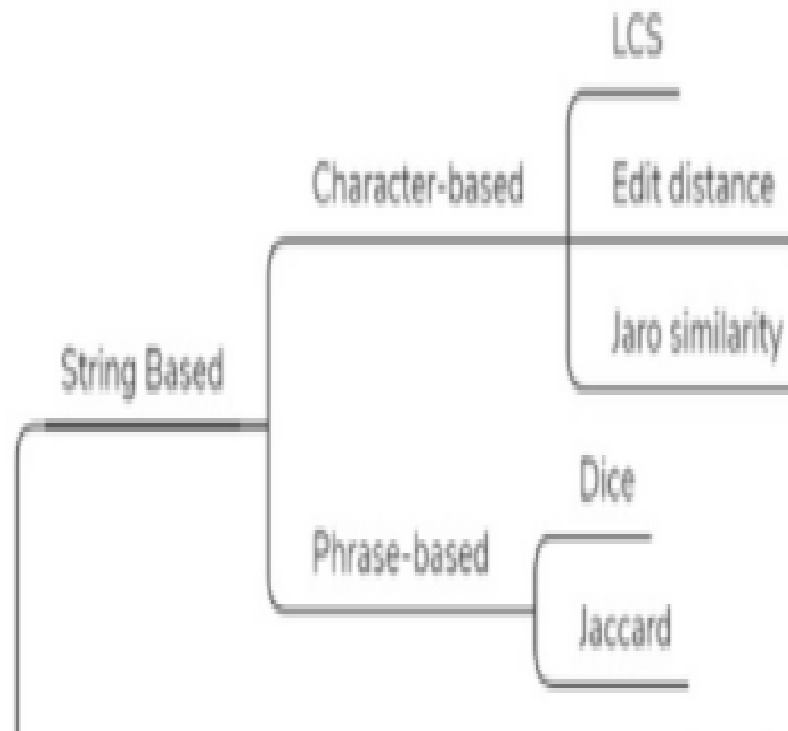
Text Distances



Text Representation



String Based



Text representation

- text matching, and graph-structure-based Text as numerical features that can be calculated directly.
- Texts can be similar in two ways lexically and semantically.
- The words that make up the text are similar lexically if they have a similar character sequence.
- Words are similar semantically if they have the same thing, are opposite of each other, used in the same way, used in the same context, and one is a type of another.
- Lexical similarity is introduced in this survey though different measurements of text representation, semantic similarity is introduced through the string-based method, corpus-based method, semantic d method.

Character-Based Text Representation

- A character-based similarity calculation is based on the similarity between characters in the text to express the similarity between texts.
- Three algorithms will be introduced:
 - LCS (longest common substring),
 - editing distance
 - Jaro similarity,

Jaro Distances

Jaro similarity

- The Jaro distance is a measure of edit distance between two strings; its inverse, called the *Jaro similarity*, is a measure of two strings' similarity: the higher the value, the more similar the strings are.
- The score is normalized such that **0** equates to no similarities and **1** is an exact match.

JARO Similarity Definition

- The Jaro similarity d_j of two given strings $S1$ and $S2$ is

$$d_j = \begin{cases} 0 & \text{if } m = 0 \\ \frac{1}{3} \left(\frac{m}{|s_1|} + \frac{m}{|s_2|} + \frac{m-t}{m} \right) & \text{otherwise} \end{cases}$$

Where:

- m is the number of *matching characters*;
- t is half the number of *transpositions*.

JARO Similarity

- Two characters from S1 and S2 respectively, are considered *matching* only if they are not farther apart than $\left\lfloor \frac{\max(|s_1|, |s_2|)}{2} \right\rfloor - 1$ characters.
- Each character of S1 is compared with all its matching characters in S2 .
- Each difference in position is half a *transposition*; that is, the number of transpositions is half the number of characters which are common to the two strings but occupy different positions in each one



Example

- Given 2 statements
 - Julie loves me more than Linda loves me
 - Jane likes me more than Julie loves me
- Find how similar these texts are, purely in terms of word counts (and ignoring word order)

SOLUTION

me	2	2
Jane	0	1
Julie	1	1
Linda	1	0
likes	0	1
loves	2	1
more	1	1
than	1	1

- The two vectors are, again:

- a: [2, 0, 1, 1, 0, 2, 1, 1]

b: [2, 1, 1, 0, 1, 1, 1, 1]

- The cosine of the angle between them is about 0.822.
- These vectors are 8-dimensional. A virtue of using cosine similarity is clearly that it converts a question that is beyond human ability to visualise to one that can be. In this case you can think of this as the angle of about 35 degrees which is some 'distance' from zero or perfect agreement.

Jaro distance

- **Jaro Similarity** is the measure of similarity between two strings. The value of Jaro distance ranges from 0 to 1. where **1 means the strings are equal** and **0 means no similarity between two strings. It checks sequence similarity**
- Jaro distance measure is based on the **number or order of the characters between two strings which are common;**

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

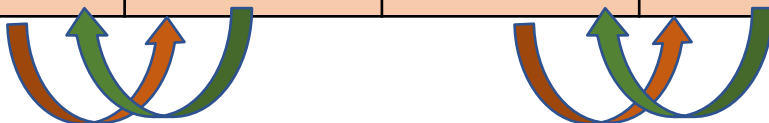
where:

- **m** is the number of matching characters
- **t** is half the number of transpositions
- **|s1|** and **|s2|** is the length of string **s1** and **s2** respectively
- **Condition for matching characters and getting number of transpositions:**
Two characters from s1 and s2 are **considered matching** only if they are the same and not farther than characters apart. $\left\lfloor \frac{\max(|s1|, |s2|)}{2} \right\rfloor - 1$
- For example, in comparing **CRATE** with **TRACE**, only 'R' 'A' 'E' are the **matching characters**, i.e. **m=3**. Although 'C', 'T' appear in both strings, they are farther apart than 1 (the result of (5/2)floor - 1. Therefore, t=0 .
- Each character of s_1 is compared with all its matching characters in s_2. The number of matching (but different sequence order) characters divided by 2 defines the number of transpositions.

EXAMPLE 1

- Let **s1="arnab"**, **s2="raanb"**, Calculate Jaro distance.
- Number of matching characters= **5** . Since Not farther than $(5/2)\text{floor} - 1$.

a	r	n	a	b
r	a	a	n	b



- But the order is not the same, so the number of characters which are **not in order** is **4**, so the **number of transpositions/2** is $4/2=2$.
- Therefore Jaro similarity can be calculated as follows:
Jaro Similariy = $(1/3) * \{(5/5) + (5/5) + (5-2)/5\} = \mathbf{0.86667}$
- **Jaro Distance = $1-0.86667=0.13333$**

EXAMPLE 2

- Given S1=WINKLER and s2=WELFARE

W	I	N	K	L	E	R
W	E	L	F	A	R	E

- Matching characters=WLER and WLRE

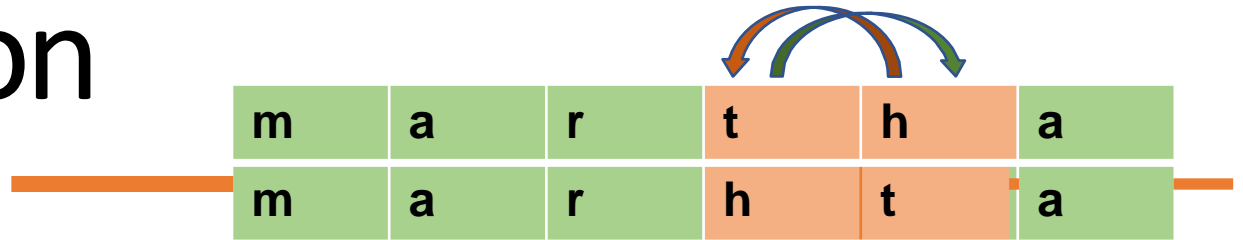
W	L	E	R
W	L	R	E

- M=4, S1=7 S2=7
- Transposition=2/2=1
- Jaro Similarity= $(1/3) * \{(4/7) + (4/7) + (4-1)/4\} = 53/(3*28) = 53/84 = 0.63095$

EXAMPLE 2

1. Given the strings $s1 = \text{"**martha**"}$ and $s2 = \text{"**marhta**"}$.
Calculate Jaro Similarity.
2. Given the strings $s1 = \text{"**DWAYNE**"}$ and $s2 = \text{"**DUANE**"}$.
Calculate Jaro Similarity.
3. Given the strings $s1 = \text{"**CRATE**"}$ and $s2 = \text{"**TRACE**"}$.
Calculate Jaro Similarity.

Solution



1. $m = 6$

$t = 2/2 = 1$ (2 couples of non matching characters, the 4-th and 5-th) { t/h ; h/t }

$|s1| = 6$ and $|s2| = 6$

• **Jaro Similarity** = $(1/3) (6/6 + 6/6 + (6-1)/6) = 17/18 = 0.944$

D	w	A	y	N	E
D	u	A	N	E	

2. $m=4$

$t=0$. (In DwAyNE versus DuANE the matching letters are already in the same order D-A-N-E, so no transpositions are needed.) $|s_1|=6$ and $|s_2|=5$

C	R	A	T	E
T	R	A	C	E

• **Jaro Similarity** = $(1/3) * (4/6 + 4/5 + (4-0)/4) = 37/45 = 0.822$

3. $m = 3$

$t = 0$ (only 'R' 'A' 'E' are the **matching characters**, i.e. $m=3$. Although 'C', 'T' appear in both strings, they are farther apart than 1 (the result of $(5/2)\text{floor} - 1$. Therefore, $t=0$. }

$|s1| = 5$ and $|s2| = 5$

• **Jaro Similarity** = $(1/3) * (3/5 + 3/5 + (3-0)/3) = 11/15 = 0.733333$

N-gram Distances

n-Gram edit distance

- An ***n*-gram** is a **contiguous sequence of *n* items** from a given sample of text or speech. The items can be phonemes, syllables, letters, words or base pairs according to the application.
- An n-gram can be obtained by splitting a string in sequences with the length n, ie substrings of length N are named "N-grams".
- **Example:** string "abcde"
 - **bigrams** are: ab, bc, cd, and de
 - **trigrams** will be: abc, bcd, and cde
 - **4-grams** will be abcd, and bcde.
- **There are different variants of N-gram distance:**
 - **Dice N-gram Matching Method**
 - **Generalized N-gram Matching for string matching**
 - **Bigram method**
 - **Trigram Method**

EXAMPLE 1

- Let $s1 = \text{PROGRAMMER}$, $s2 = \text{PROGRAMMING}$. $N(s1) = 10$ and $N(s2) = 11$, $\max\{N(s1), N(s2)\} = 11$. Calculate the 4 n-gram distances.

Compute 4-gram similarity

- To calculate the 4-gram distances (also known as 4-gram similarity) between two strings, you can follow these steps:
 - Break both strings, s_1 and s_2 , into sets of 4-grams (substrings of length 4).
 - Calculate the Jaccard similarity coefficient between these two sets of 4-grams.
 - The 4-gram distance is then computed as $1 - \text{Jaccard Similarity}$

Compute 4-grams from each string:

- 4-grams of s1:
 - {"PROG", "ROGR", "OGRM", "GRAM", "RAMM", "AMME", "MMER", "MERR"}
- 4-grams of s2:
 - {"PROG", "ROGR", "OGRM", "GRAM", "RAMM", "AMMI", "MMIN", "MING"}

Calculate the Jaccard similarity

- Intersection: {"PROG", "ROGR", "OGRM", "GRAM", "RAMM"}
- Union: {"PROG", "ROGR", "OGRM", "GRAM", "RAMM", "AMME", "MMER", "MERR", "AMMI", "MMIN", "MING"}

N gram through Formula Jaccard

- Jaccard Similarity =
(Size of Intersection) / (Size of Union)
- $5 / 11$
- ≈ 0.4545
- Now, calculate the 4-gram distance:
- 4-gram Distance = $1 - \text{Jaccard Similarity}$
- $= 1 - 0.4545$
- ≈ 0.5455
- So, the 4-gram distance between the strings "PROGRAMMER" and "PROGRAMMING" is approximately 0.5455.
- This value indicates the dissimilarity between the two strings based on their 4-gram similarity.

Longest Common Subsequences(LCS) Distances

LCS Problem

- The longest common subsequence (LCS) problem is the problem of finding the longest subsequence common to all sequences in a set of sequences (often just two sequences).
- It differs from the longest common substring problem: unlike substrings, subsequences are not required to occupy consecutive positions within the original sequences.
- The longest common subsequence problem is a classic computer science problem, the basis of data comparison programs such as the diff utility, and has applications in computational linguistics and bioinformatics.
- It is also widely used by revision control systems such as Git for reconciling multiple changes made to a revision-controlled collection of files

Example

- consider the sequences (ABCD) and (ACBAD).
- They have 5 length-2 common subsequences:
 - (AB), (AC), (AD), (BD), and (CD);
- 2 length-3 common subsequences: (ABD) and (ACD); and no longer common subsequences.
- So (ABD) and (ACD) are their longest common subsequences

LCS

- The LCS problem has an optimal substructure: the problem can be broken down into smaller, simpler subproblems, which can, in turn, be broken down into simpler subproblems, and so on,
- until, finally, the solution becomes trivial.
- LCS in particular has overlapping subproblems: the solutions to high-level subproblems often reuse solutions to lower level subproblems.
- Problems with these two properties are amenable to dynamic programming approaches, in which subproblem solutions are memoized, that is, the solutions of subproblems are saved for reuse.

DP Algorithm

LCS-Length(X, Y)

1. $m = \text{length}(X)$ // get the # of symbols in X
2. $n = \text{length}(Y)$ // get the # of symbols in Y
3. for $i = 1$ to m $c[i,0] = 0$ // special case: Y[0]
4. for $j = 1$ to n $c[0,j] = 0$ // special case: X[0]
5. for $i = 1$ to m // for all X[i]
6. for $j = 1$ to n // for all Y[j]
7. if ($X[i] == Y[j]$)
8. $c[i,j] = c[i-1,j-1] + 1$
9. else $c[i,j] = \max(c[i-1,j], c[i,j-1])$
10. return c

Longest Common

Subsequence Algorithm

- A subsequence of a string S , is a set of characters that appear in left - to-right order, but not necessarily consecutively.
- **Example:** Consider **ACTTGCG**
 - ACT, ATTC, T, ACTTGC are all subsequences. TTA is not a subsequence. There are 2^n subsequences of string S of length n .
- A common subsequence of two strings is a subsequence that appears in both strings. A longest common subsequence is a common subsequence of maximal length.

Example

$S1 = \text{AAACCGTGAGTTATTCTAGAA}$

$S2 = \text{CACCCTAAGGTACCTTGGTTC}$

ALGORITHM

LCS – Length(X, Y)

```
m = length[X]      n = length[Y]
for i = 1 to m
    c[i, 0] = 0
for j = 0 to n
    c[0, j] = 0
for i = 1 to m
    for j = 1 to n
        if xi == yj
            c[i, j] = c[i - 1, j - 1] + 1
            B[i, j] := 'D' or ↖
        else if (c[i - 1, j] ≥ c[i, j - 1])
            c[i, j] = c[i - 1, j]
            B[i, j] := 'U' or ↑
        else
            c[i, j] = c[i, j - 1]
            B[i, j] := 'L' or ←
    return c
and B
```

SEQUENCE RETRIEVAL

Algorithm: Print-LCS (B, X, i, j)

```
    if i = 0
    and j = 0

    return
    if B[i, j] = 'D' or ↖
        Print-LCS(B, X, i-1, j-1)
        Print(xi)
    else if B[i, j] = 'U' or ↑
        Print-LCS(B, X, i-1, j)
    else
        Print-LCS(B, X, i, j-1)
        //for 'L' or
        ←
```

EXAMPLE 1

- we have two strings
- **$X = ABCBDAB$** and **$Y = BDCABA$**

To find the longest common subsequence.
Following the algorithm
LCS-Length- Table-
Formulation

		j	0	1	2	3	4	5	6
			y_j						
				B	D	C	A	B	A
0	x_i		0	0	0	0	0	0	0
1	A		0	↑	↑	↑	↖	←	↖
2	B		0	↖	↖	←	↑	↖	←
3	C		0	↑	↑	↖	←	↑	↑
4	B		0	↖	↑	↑	↑	↖	←
5	D		0	↑	↖	↑	↑	↑	↑
6	A		0	↑	↑	↑	↖	↑	↖
7	B		0	↖	↑	↑	↑	↖	↑

EXAMPLE 2

X= innovation and **Y= tionwagon** **LCS=inaon**

	Yj	T	I	O	N	W	A	G	O	N
Xi	0	0	0	0	0	0	0	0	0	0
I	0	0	↑	↖	←	←	←	←	←	←
N	0	0	↑	↑	↖	←	↑	←	←	↖
N	0	0	↑	↑	↑	↖	↑	↑	↑	↖
O	0	0	↑	↑	↖	↑	↑	↑	↖	↑
V	0	0	↑	↑	↑	↑	↑	↑	↑	↑
A	0	0	↑	↑	↑	↑	↖	←	↑	↑
T	0	↖	↑	↑	↑	↑	↑	↑	↑	↑
I	0	↑	↖	↑	↑	↑	↑	↑	↑	↑
O	0	↑	↑	↖	←	←	↑	↑	↖	↑
N	0	↑	↑	↑	↖	←	←	←	←	↖



Thank
You