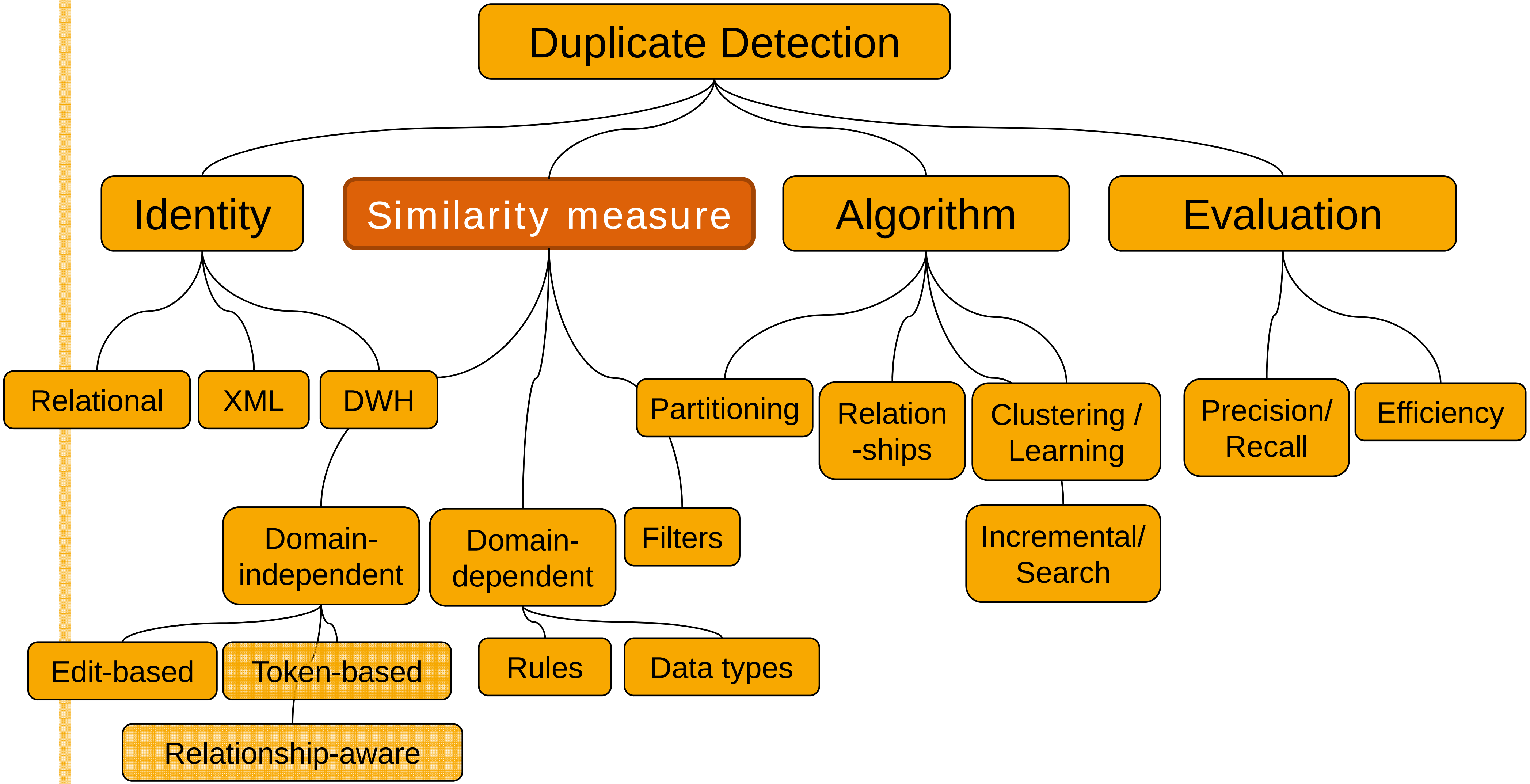
Similarity measures

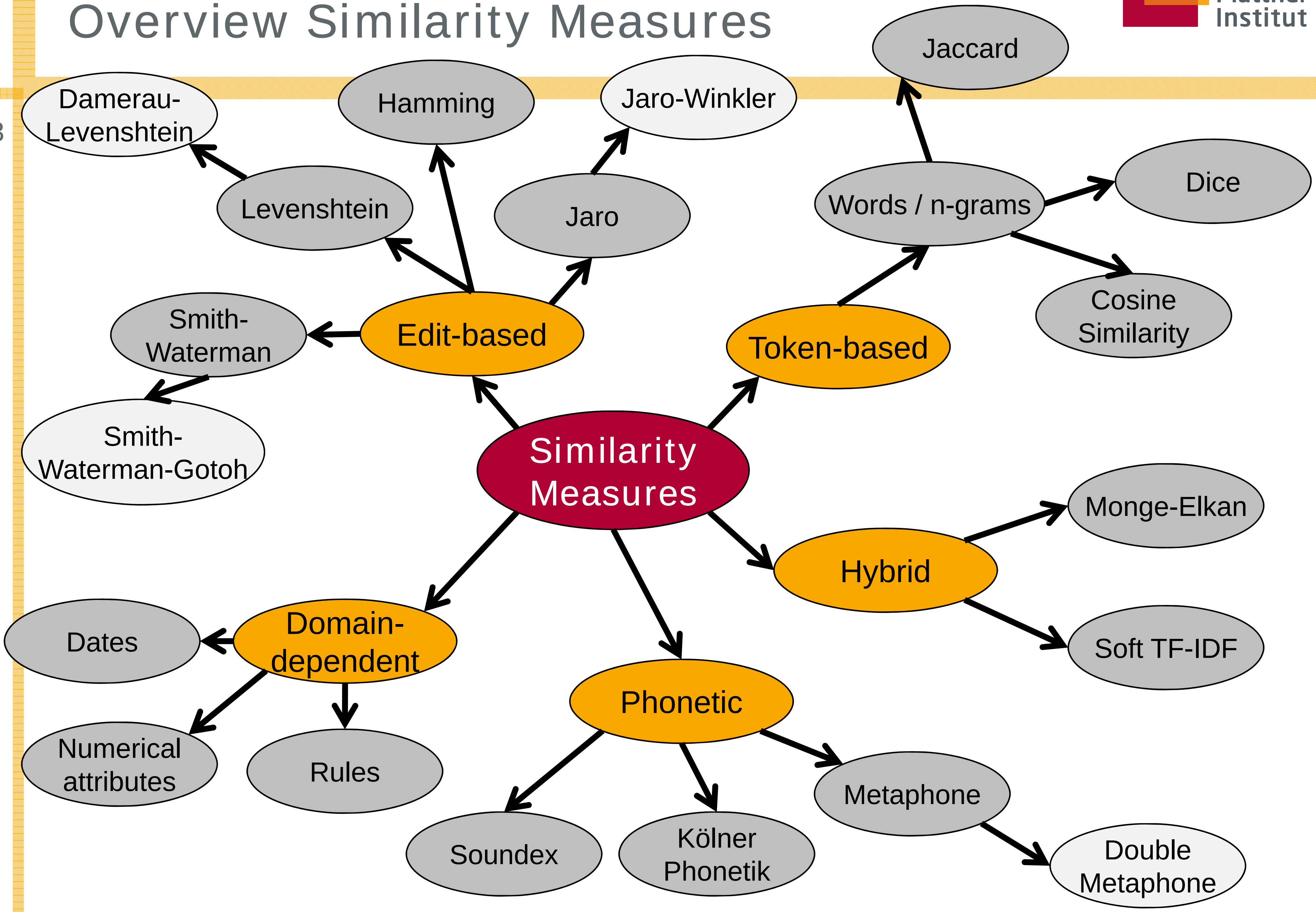
11.6.2013

Felix Naumann

Duplicate Detection – Research



Overview Similarity Measures



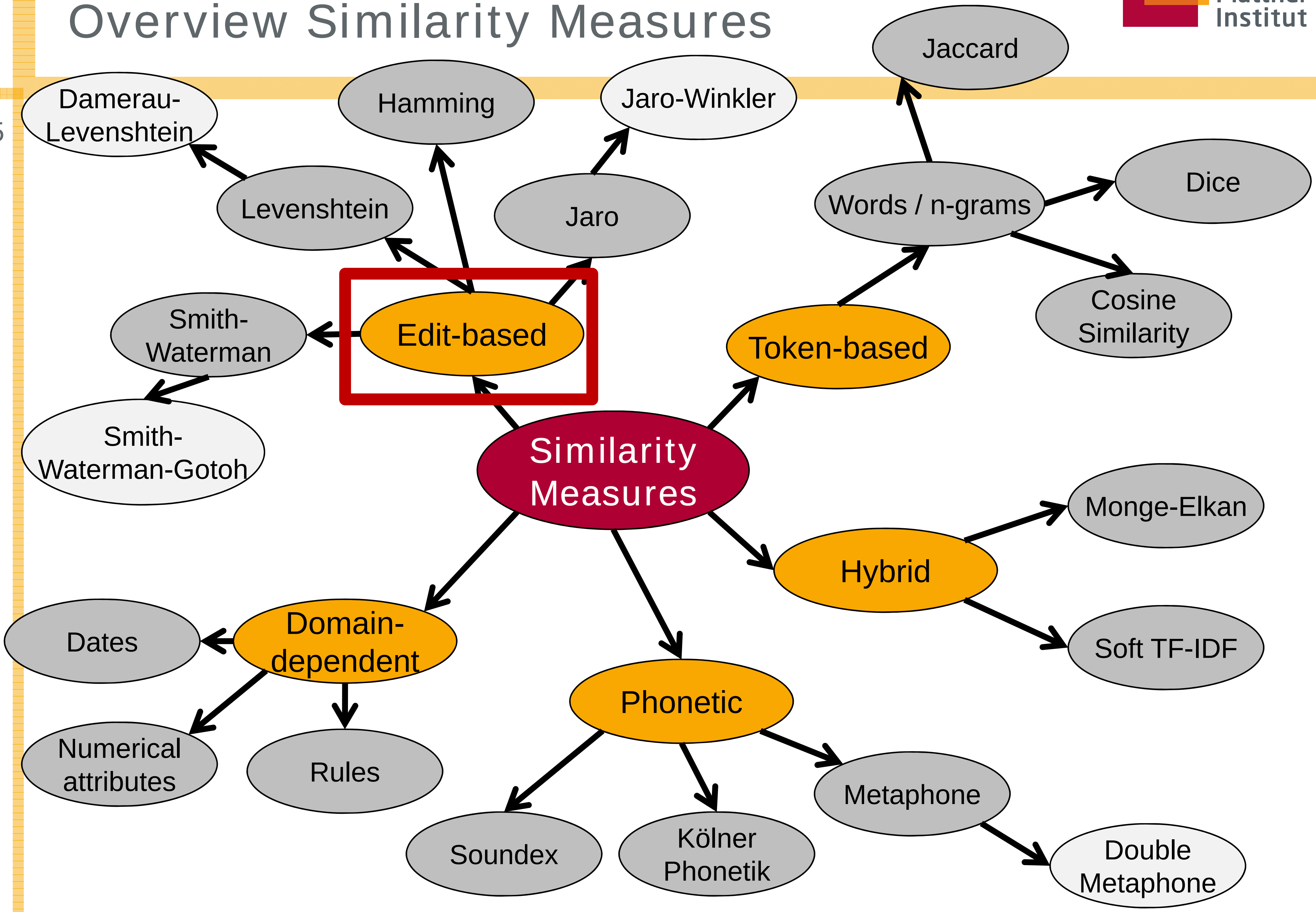
Similarity measures

- $sim(x,y)$
  - x and y can be strings, numbers, tuples, objects, images, …
- Normalized: $sim(x,y) \in [0,1]$
  - $sim(x,y) = 1$ for exact match
  - $sim(x,y) = 0$ for „completely different“ x and y.
  - $0 < sim(x,y) < 1$ for some approximate similarity

- Distance function / distance metric
  - Reflexive: $dist(x,x) = 0$
  - Positive: $dist(x,y) \geq 0$
  - Symmetric: $dist(x,y) = dist(y,x)$
  - Triangular inequation: $dist(x,z) \leq dist(x,y) + dist(y,z)$

- $sim(x,y) = 1 – dist(x,y)$
- $sim(x,y) = 1/dist(x,y)$

Overview Similarity Measures



Exact and truncated match

- $sim_{exact}(x, y) = \begin{cases} 1 \; if \; x = y \\ 0 \; if \; x \neq y \end{cases}$

- $sim_{trunc\_beg}(x, y) = \begin{cases} 1 \; if \; x[1{:}k] = y[1{:}k] \\ 0 \; if \; x[1{:}k] \neq y[1{:}k] \end{cases}$

- $sim_{trunc\_end}(x, y) = \begin{cases} 1 \; if \; x[k{:}n] = y[k{:}n] \\ 0 \; if \; x[k{:}n] \neq y[k{:}n] \end{cases}$

- $sim_{encode}(x, y) = \begin{cases} 1 \; if \; encode(x) = encode(y) \\ 0 \; if \; encode(y) \neq encode(y) \end{cases}$
  - E.g., with a phonetic encoding

Hamming distance

- Number of positions in which two strings (of equal length) differ
  - Minimum number of *substitutions* required to change one string into the other
  - Minimum number of *errors* that could have transformed one string into the other.
- Used mostly for binary numbers and to measure communication errors.
  - Hamming distance = number of 1s in *x* XOR *y*.
- $\text{dist}_{\text{hamming}}(\text{peter},\text{pedro}) = 3$

Edit distances

- Compare two strings based on individual characters
- Minimal number of edits required to transform one string into the other.
  - Edits: Insert, Delete, Replace (and Match)
  - Alternative: Smallest edit cost
  - Give different cost to different types of edits
  - Give different cost to different letters
- Naive approach: *editdistance*(Jones,Johnson)
  - DDDDDIIIIIII = 12
  - But: Not minimal!

- Levenshtein distance: Basic form
  - Each edit has cost 1

Levenshtein Distance

- Minimum number of character **insertions**, **deletions**, and **replacements** necessary to transform s_1 into s_2
- Compute transcript based on **dynamic programming** algorithm
  - Optimality principle: Best transcript of two substrings must be part of best overall solution

1. Initialize matrix *M* of size $(|s_1|+1) \times (|s_2|+1)$
2. Fill matrix: $M_{i,0} = i$ and $M_{0,j} = j$
3. Recursion: $M_{i,j} = \begin{cases} M_{i-1,j-1} & if\ x[i] = y[j] \\ 1 + \min(M_{i-1,j}, M_{i,j-1}, M_{i-1,j-1}) & else \end{cases}$
4. Distance: $LevenshteinDist(x, y) = M_{|x|,|y|}$

Levenshtein Similarity: $sim_{Levenshtein}(x, y) = 1 - \frac{LevenshteinDist(x,y)}{\max(|x|,|y|)}$

Levenshtein Distance

		J	O	N	E	S
	0	1	2	3	4	5
J	1					
O	2					
H	3					
N	4					
S	5					
O	6					
N	7					

		J	O	N	E	S
	0	1	2	3	4	5
J	1	0	1	2	3	4
O	2					
H	3					
N	4					
S	5					
O	6					
N	7					

		J	O	N	E	S
	0	1	2	3	4	5
J	1	0	1	2	3	4
O	2	1	0	1	2	3
H	3	2	1	1	2	3
N	4	3	2	1	2	3
S	5	4	3	2	2	2
O	6	5	4	3	3	3
N	7	6	5	4	4	4

$$M_{i,j} = \begin{cases} M_{i-1,j-1} & if\ x[i] = y[j] \\ 1 + \min(M_{i-1,j}, M_{i,j-1}, M_{i-1,j-1}) & else \end{cases}$$

Levenshtein Distance – Example

$$sim_{Levenshtein} = 1 - \frac{LevenshteinDist}{\max(|s_1|, |s_2|)}$$

s_1	s_2	Levenshtein Distance	$sim_{Levenshtein}$
Jones	Johnson	4	0.43
Paul	Pual	2	0.5
Paul Jones	Jones, Paul	11	0

Levenshtein discussion

- Complexity
  - Time: $O(|x| \cdot |y|)$ (fill in matrix)
  - Space: $O(\min(|x|, |y|))$
    - Trick: Store only two rows of the matrix
- Some properties
  - $0 \leq LevenshteinDist(x, y) \leq \max(|x|, |y|)$
  - $|\,|x| - |y|\,| \leq LevenshteinDist(x, y)$
    - Often: Compare only strings with similar lengths
- Other cost models
  - Insert, delete cost 1.0 but replace 0.5
    - change in string length is punished, e.g. for zip codes
  - Character based: OCR (m ≃ n, 1 ≃ l) or keyboard (a ≃ s) or brain (6 ≃ 9) or biology (a ≃ t)

Damerau–Levenshtein distance

- Similar to Levenshtein distance, but additionally considers transposed characters
- $M_{i,0} = i$ and $M_{0,j} = j$
- $M_{i,j} =$

$$\begin{cases} M_{i-1,j-1} & if\, x[i] = y[j] \\ 1 + min\begin{pmatrix} M_{i-1,j}, M_{i,j-1}, \\ M_{i-1,j-1}, \\ M_{i-2,j-2}\ if\ x[i] = y[j-1]\ and\ x[i-1] = y[j] \end{pmatrix} & else \end{cases}$$

s_1	s_2	Levenshtein Distance	Damerau-Levenshtein Distance	$sim_{Damerau\text{-}Levenshtein}$
Jones	Johnson	4	4	0.43
Paul	Pual	2	1	0.75
Paul Jones	Jones, Paul	11	11	0

Jaro similarity

- Specifically designed for names at US Census Bureau
- Search for common characters
- *m :* number of matching characters
  - Search range matching characters: $\frac{\max(|x|,|y|)}{2} - 1$
- *t* : number of transpositions
- $sim_{jaro} = \frac{1}{3}\left(\frac{m}{|x|} + \frac{m}{|y|} + \frac{m-t}{m}\right)$

Jaro similarity – Example

- $sim_{jaro} = \frac{1}{3}\left(\frac{m}{|x|} + \frac{m}{|y|} + \frac{m-t}{m}\right)$

s_1	**P**	**A**	**U**	**L**
s_2	**P**	**U**	**A**	**L**

$m = 4 \qquad t = \frac{2}{2} = 1$

$$sim_{jaro} = \frac{1}{3} \cdot \left(\frac{4}{4} + \frac{4}{4} + \frac{4-1}{4}\right) \approx 0.92$$

s_1	**J**	**O**	**N**	E	**S**		
s_2	**J**	**O**	H	**N**	**S**	O	N

$m = 4 \qquad t = \frac{0}{2} = 0$

$$sim_{jaro} = \frac{1}{3} \cdot \left(\frac{4}{5} + \frac{4}{7} + \frac{4-0}{4}\right) \approx 0.79$$

Winkler similarity

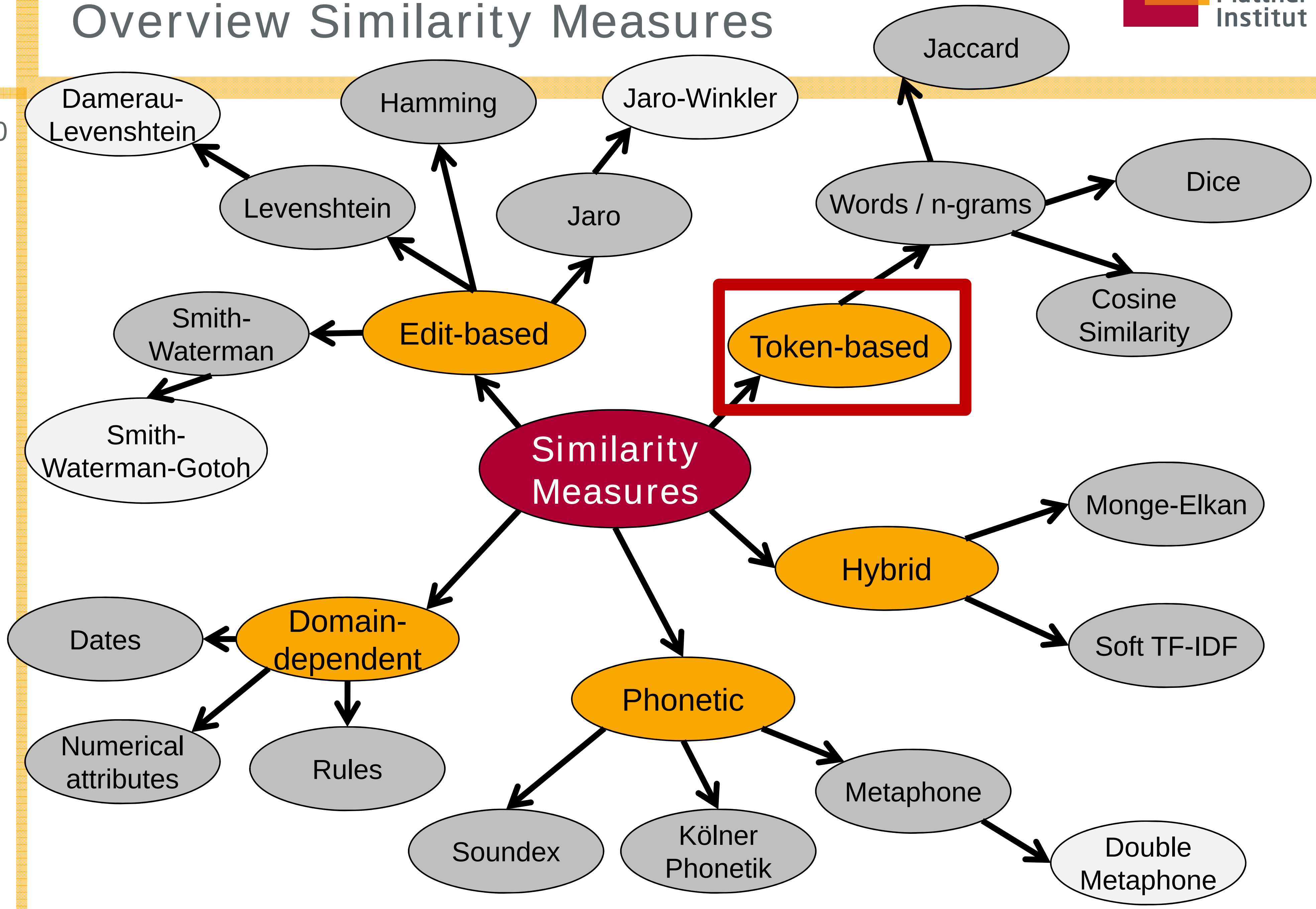
- Intuition 1: Similarity of first few letters is most important.
- Let *p* be the length of the common prefix of x and y.
- $sim_{winkler}(x,y) = sim_{jaro}(x,y) + (1 - sim_{jaro}(x,y))\frac{p}{10}$
  - = 1 if common prefix is ≥10

- Intuition 2: Longer strings with even more common letters
- $sim_{winkler\_long}(x,y) = sim_{winkler}(x,y) + (1 - sim_{winkler}(x,y))\frac{c-(p+1)}{|x|+|y|-2(p-1)}$
  - Where *c* is overall number of common letters
  - Apply only if
    - Long strings: $\min(|x|,|y|) \geq 5$
    - Two additional common letters: $c - p \geq 2$
    - At least half remaining letters of shorter string are in common: $c - p \geq \frac{\min(|x|,|y|)-p}{2}$

Comparison

String 1	String 2	c	t	p	c_{sim}	sim_{jaro}	$sim_{winkler}$	$sim_{winkler\_long}$
shackleford	**shackelford**	11	1	4	0	0.9697	0.9818	0.9886
nichleson	**nichulson**	8	0	4	0.3	0.9259	0.9556	0.9667
jones	**johnson**	4	0	2	0.3	0.7905	0.8324	0.8491
massey	**massie**	5	0	4	0.3	0.8889	0.9333	–
jeraldine	**geraldine**	8	0	0	0.3	0.9259	0.9259	0.9519
michelle	**michael**	6	0	4	0.3	0.8690	0.9214	0.9302

Overview Similarity Measures

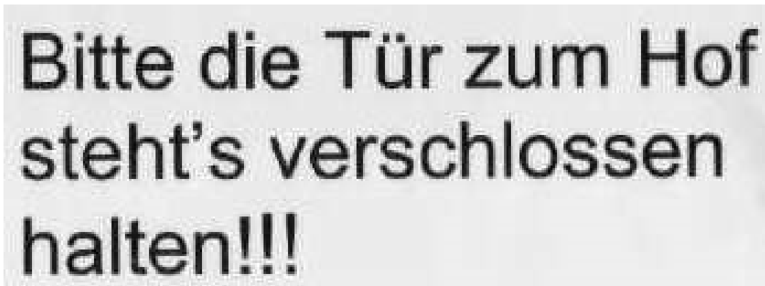


Tokenization

- Forming words from sequence of characters
- General idea: Separate string into tokens using some separator
  - Space, hyphen, punctuation, special characters
  - Usually also convert to lower-case
- Problems
  - Both hyphenated and non-hyphenated forms of many words are common
    - Sometimes hyphen is *not needed*
      - *e-bay, wal-mart, active-x, cd-rom, t-shirts*
    - Sometimes hyphens *should be considered* either as part of the word or a word separator
      - *winston-salem, mazda rx-7, e-cards, pre-diabetes, t-mobile, spanish-speaking*
  - Apostrophes can be a part of a word, a part of a possessive, or just a mistake
    - *rosie o'donnell, can't, don't, 80's, 1890's, men's straw hats, master's degree, england's ten largest cities, shriner's*
  - Numbers can be important, including decimals
    - *nokia 3250, top 10 courses, united 93, quicktime 6.5 pro, 92.3 the beat, 288358*
  - Periods can occur in numbers, abbreviations, URLs, ends of sentences, and other situations
    - *I.B.M., Ph.D., cs.umass.edu, F.E.A.R.*

Die Kapostropheum-Gruselgalerie – Kategorie „Völlig willenlos“

http://www.apostroph.de/



n-grams (aka q-grams)

- Split string into short substrings of length *n*.
  - Sliding window over string
  - *n*=2: Bigrams
  - *n*=3: Trigrams
  - Variation: Pad with *n* – 1 special characters
    - Emphasizes beginning and end of string
  - Variation: Include positional information to weight similarities
- Number of *n*-grams = |*x*| – *n* + 1
- Count how many n-grams are common in both strings

String	Bigrams	Padded bigrams	Positional bigrams	Trigrams
gail	ga, ai, il	⊙g, ga, ai, il, l⊗	(ga,1), (ai,2), (il,3)	gai, ail
gayle	ga, ay, yl, le	⊙g, ga, ay, yl, le, e⊗	(ga,1), (ay,2), (yl,3), (le,4)	gay, ayl, yle
peter	pe, et, te, er	⊙p, pe, et, te, er, r⊗	(pe,1), (et,2), (te,3), (er,4)	pet, ete, ter
pedro	pe, ed, dr, ro	⊙p, pe, ed, dr, ro, o⊗	(pe,1), (ed,2), (dr,3), (ro,4)	ped, edr, dro

Token-based Similarity Measures

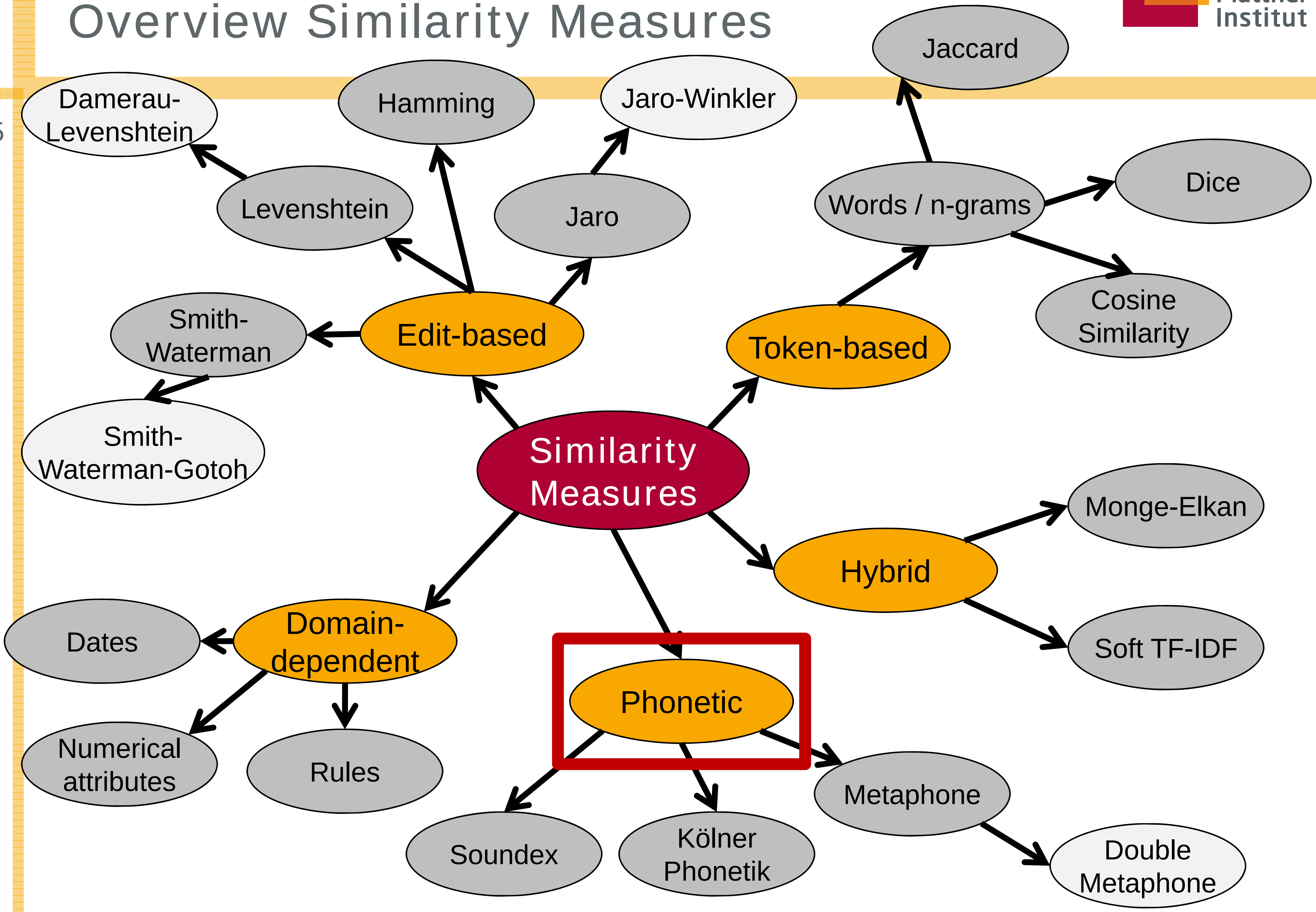
- Token similarity
  - Overlap coefficient: $sim_{overlap}(x,y) = \frac{|tok(x) \cap tok(y)|}{\min(|tok(x)|,|tok(y)|)}$
  - Jaccard coefficient:

$$sim_{jaccard}(x,y) = \frac{|tok(x)\ \cap\ tok(y)|}{|tok(x)| + |tok(y)| - |tok(x)\ \cap\ tok(y)|} = \frac{|tok(x)\ \cap\ tok(y)|}{|tok(x)\ \cup\ tok(y)|}$$

  - Dice's coefficient: $sim_{dice}(x,y) = \frac{2 \cdot |tok(x) \cap tok(y)|}{|tok(x)|+|tok(y)|}$
- Tokens („Paul Jones“)
  - Words / Terms („Paul“ „Jones“)
  - Padded n-grams (_P, Pa, au, ul, l_, _J, Jo, on, ne, es, s_)

s_1	s_2	*Jaccard*	*Dice*
Jones	Johnson	0.17	0.29
Paul	Pual	0.33	0.40
Paul Jones	Jones, Paul	0.77	0.87

Overview Similarity Measures



Soundex

- Soundex codes a last name based on the way a last name sounds
  1. Retain first letter of the name and drop all other occurrences of A, E, H, I, O, U, W, Y
  2. Replace consonants with digits
  3. Two adjacent letters with the same number are coded as a single number
  4. Continue until you have one letter and three numbers. If you run out of letters, pad with 0s.
- If a surname has a prefix, such as Van, Con, De, Di, La, or Le, code both with and without the prefix

Digit	Letters
1	B, F, P, V
2	C, G, J, K, Q, S, X, Z
3	D, T
4	L
5	M, N
6	R

- Example
  - PAUL: P400
  - PUAL: P400
  - JONES: J520
  - JOHNSON: J525

Jenkins, Jansen, Jameson

Soundex on WolframAlpha

Kölner Phonetik

- Like Soundex, but specialized for German last names
- Letters get different codes based on the context
- Code length is not restricted
- Multiple occurrences of the same code and „0“ are removed

- Example
  - PAUL: 15
  - PUAL: 15
  - JONES: 68
  - JOHNSON: 686

Letter	Context	Code
A, E, I, J, O, U, Y		0
H		-
B		1
P	not before H	1
D, T	not before C, S, Z	2
F, V, W		3
P	before H	3
G, K, Q		4
C	in the initial sound before A, H, K, L, O, Q, R, U, X	4
C	before A, H, K, O, Q, U, X but not after S, Z	4
X	not after C, K, Q	48
L		5
M, N		6
R		7
S, Z		8
C	after S, Z	8
C	in the initial sound, but not before A, H, K, L, O, Q, R, U, X	8
C	not before A, H, K, O, Q, U, X	8
D, T	before C, S, Z	8
X	after C, K, Q	8

Metaphone

- Improves on the Soundex algorithm
  - Knows variations and inconsistencies in English spelling and pronunciation

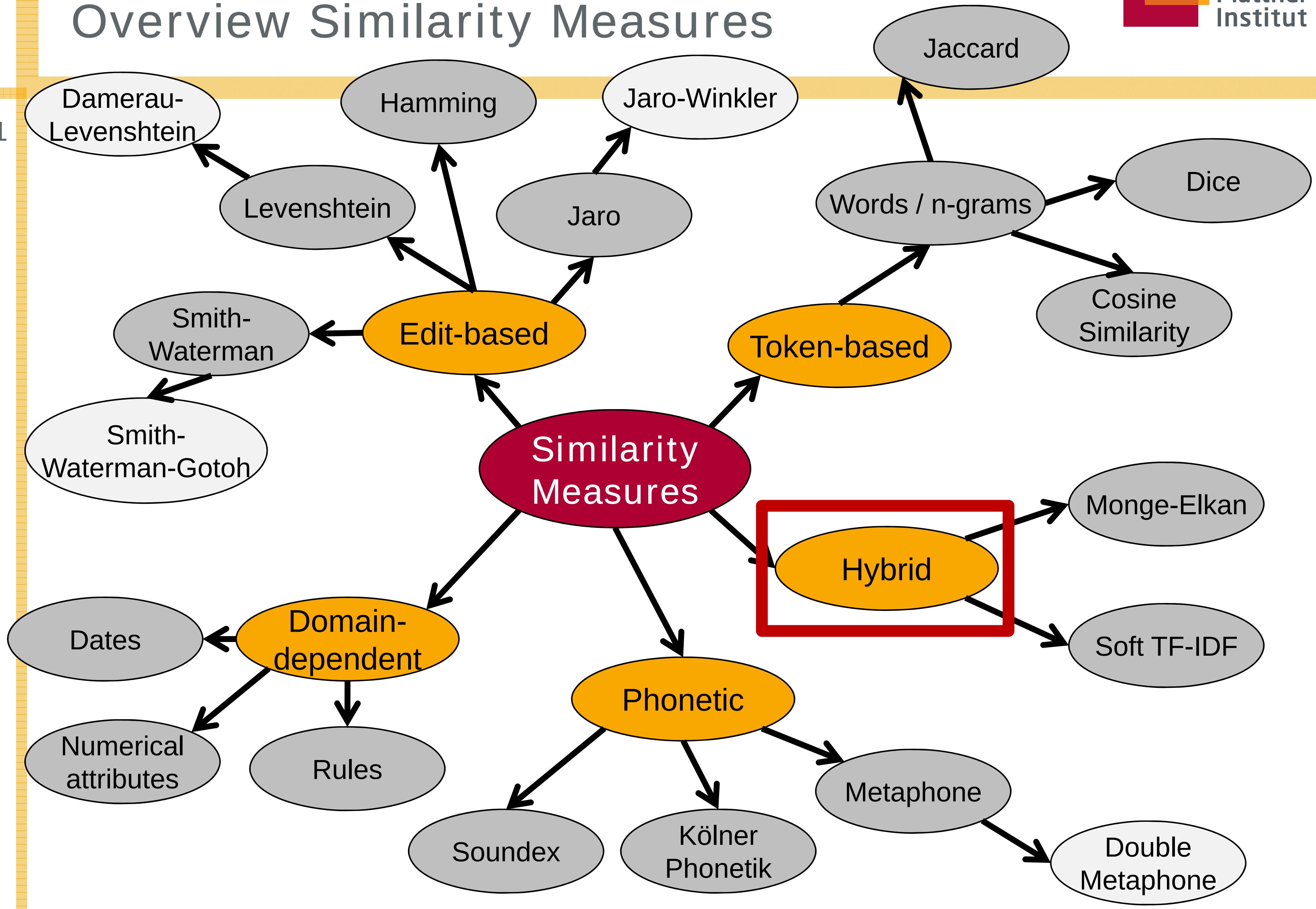
- Further improvements
  - Double Metaphone
    - Includes other languages: Slavic, Germanic, Celtic, Greek, French, Italian, Spanish, Chinese
    - Accuracy 89%
  - Metaphone 3
    - Accuracy over 99% (says author)

Original Metaphone Algorithm

- 16 consonant symbols 0BFHJKLMNPRSTWXY
  - '0' represents "th", 'X' represents "sh" or "ch“

1. Drop duplicate adjacent letters, except for C.
2. If the word begins with 'KN', 'GN', 'PN', 'AE', 'WR', drop the first letter.
3. Drop 'B' if after 'M' at the end of the word.
4. 'C' transforms to 'X' if followed by 'IA' or 'H' (unless in latter case, it is part of '-SCH-', in which case it transforms to 'K'). 'C' transforms to 'S' if followed by 'I', 'E', or 'Y'. Otherwise, 'C' transforms to 'K'.
5. 'D' transforms to 'J' if followed by 'GE', 'GY', or 'GI'. Otherwise, 'D' transforms to 'T'.
6. Drop 'G' if followed by 'H' and 'H' is not at the end or before a vowel. Drop 'G' if followed by 'N' or 'NED' and is at the end.
7. 'G' transforms to 'J' if before 'I', 'E', or 'Y', and it is not in 'GG'. Otherwise, 'G' transforms to 'K'.
8. Drop 'H' if after vowel and not before a vowel.
9. 'CK' transforms to 'K'.
10. 'PH' transforms to 'F'.
11. 'Q' transforms to 'K'.
12. 'S' transforms to 'X' if followed by 'H', 'IO', or 'IA'.
13. 'T' transforms to 'X' if followed by 'IA' or 'IO'. 'TH' transforms to '0'. Drop 'T' if followed by 'CH'.
14. 'V' transforms to 'F'.
15. 'WH' transforms to 'W' if at the beginning. Drop 'W' if not followed by a vowel.
16. 'X' transforms to 'S' if at the beginning. Otherwise, 'X' transforms to 'KS'.
17. Drop 'Y' if not followed by a vowel.
18. 'Z' transforms to 'S'.
19. Drop all vowels unless it is the beginning

Overview Similarity Measures



Monge-Elkan

- Hybrid: Token-based and internal similarity function for tokens
  - Find best match for each token

- $sim_{MongeElkan}(x,y) = \frac{1}{|x|}\sum_{i=1}^{|x|} \max_{j=1,|y|} sim'(x[i], y[j])$
  - $|x|$ is number of tokens in x
  - *sim'* is internal similarity function (e.g., Levenshtein)
- If strings contain just one token each
  - $sim_{MongeElkan}(x,y) = sim'(x,y)$

- Complexity: Quadratic in number of tokens

Monge-Elkan – Example

- $sim_{MongeElkan}(x,y) = \frac{1}{|x|}\sum_{i=1}^{|x|} \max_{j=1,|y|} sim'(x[i],y[j])$
- Peter Christen vs. Christian Pedro
  - sim_{jaro}(peter, christian) = 0.3741
  - sim_{jaro}(peter, pedro) = 0.7333
  - sim_{jaro}(christen, christian) = 0.8843
  - sim_{jaro}(christen, pedro) = 0.4417
- $sim_{MongeElkan}('peter\ christen','christian\ pedro') = \frac{1}{2}(0.7333 + 0.8843) = 0.8088$
- $sim_{MongeElkan}(x,y) \neq sim_{MongeElkan}(y,x)$

s_1 = „aaa xaa yaa“

s_2 = „aaa“

s_1 = „aaa xaa yaa“

s_2 = „aaa xxx yyy“

Extended Jaccard Similarity

- $sim_{jaccard}(x,y) = \frac{|tok(x) \cap tok(y)|}{|tok(x)|+|tok(y)|-|tok(x) \cap tok(y)|} = \frac{|tok(x) \cap tok(y)|}{|tok(x) \cup tok(y)|}$

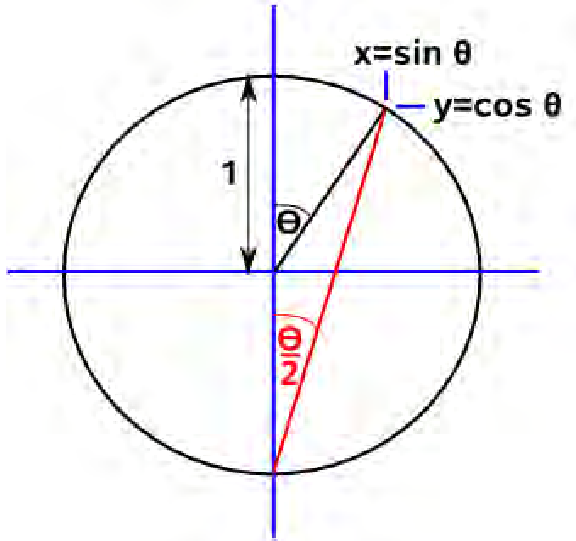
- If strings contain multiple words, choose words as tokens.
- Use internal similarity function to calculate similarity between all pairs of tokens.
  - Shared tokens:
    $S = \{(x_i, y_j) |\ x_i \in tok(x) \wedge y_j \in tok(y) : sim'(x_i, y_j) \geq \theta\}$
  - Unique tokens: $U_{tok(x)} = \{x_i | x_i \in tok(x) \wedge y_j \in tok(y) \wedge (x_i, y_j) \notin S\}$

- $sim_{jaccad\_ext}(x,y) = \frac{|S|}{|S|+|U_{tok(x)}|+|U_{tok(y)}|}$

Vector Space Model

- Each document ranked by distance between points representing query and document
- Popular measure: Cosine similarity
  - Cosine of angle between document and query vectors
  - Normalized dot-product

$$Cosine(D_i, Q) = \frac{\sum_{j=1}^{t} d_{ij} \cdot q_j}{\sqrt{\sum_{j=1}^{t} d_{ij}^2 \cdot \sum_{j=1}^{t} q_j^2}}$$



http://www.euclideanspace.com/maths/geometry/trig/derived/index.htm

Similarity Calculation – Example

- Consider three documents D_1, D_2, D_3 and query Q
  - D_1 = (0.5, 0.8, 0.3), D_2 = (0.9, 0.4, 0.2), D_3 = (0, 0.9, 0.1)
  - Q = (1.5, 1.0, 0)
- Vector space model reflects some term weights and number of matching terms (in contrast to Boolean retrieval)

$$\begin{aligned} Cosine(D_1, Q) &= \frac{(0.5 \times 1.5) + (0.8 \times 1.0)}{\sqrt{(0.5^2 + 0.8^2 + 0.3^2)(1.5^2 + 1.0^2)}} \\ &= \frac{1.55}{\sqrt{(0.98 \times 3.25)}} = 0.87 \\ Cosine(D_2, Q) &= \frac{(0.9 \times 1.5) + (0.4 \times 1.0)}{\sqrt{(0.9^2 + 0.4^2 + 0.2^2)(1.5^2 + 1.0^2)}} \\ &= \frac{1.75}{\sqrt{(1.01 \times 3.25)}} = 0.97 \end{aligned}$$

$Cosine(D_3, Q) = 0.55$

- But: How to assign term weights?

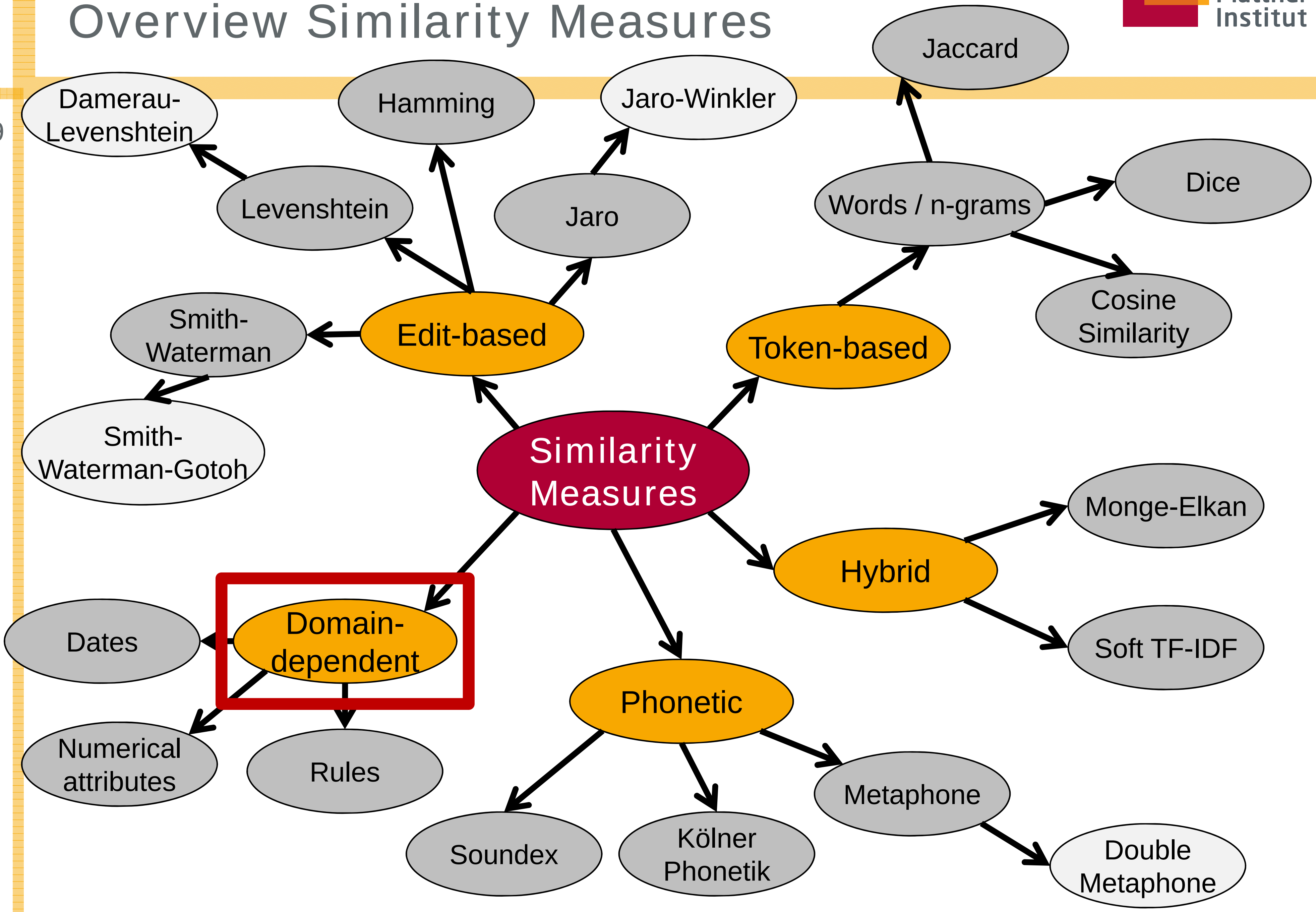
Term Weights – *tf.idf*

- Term frequency weight *tf* measures importance of term k in document *i*: $tf_{ik} = \frac{f_{ik}}{\sum_{j=1}^{t} f_{ij}}$
  - *log(*f_{ik}*)* to reduce this impact of frequent words
- Inverse document frequency *idf* measures importance in collection: $idf_k = log \frac{N}{n_k}$
  - Reflects “amount of information” carried by term
- *tfidf* by multiplying *tf* and *idf* with some heuristic modifications

SoftTFIDF

- Apply idea to records or values: Much shorter than documents
- $CLOSE(\theta, tok(x), tok(y))$ is set of tokens from *x* that have at least one sufficiently similar token in *y*.
- $sim_{softtfidf}(x, y) = \sum_{t \in CLOSE(\theta, tok(x), tok(y))} V\big(t, tok(x)\big) \cdot V(t, tok(y)) \cdot N(t, tok(y))$
  - $V\big(t, tok(x)\big)$ is TFIDF weight of token *t* in all tokens of *x*
  - $N\big(t, tok(y)\big) = \max(\{sim'\big(t, y_j\big) | y_j \in tok(y)\})$
    - Similarity of best matching token
- Soft: Tokens are considered a partial match if they get a good score using an internal similarity measure (CLOSE).
- Problem: Weights are calculated over entire database
  - Scan all data
  - Store weight for each unique token

Overview Similarity Measures



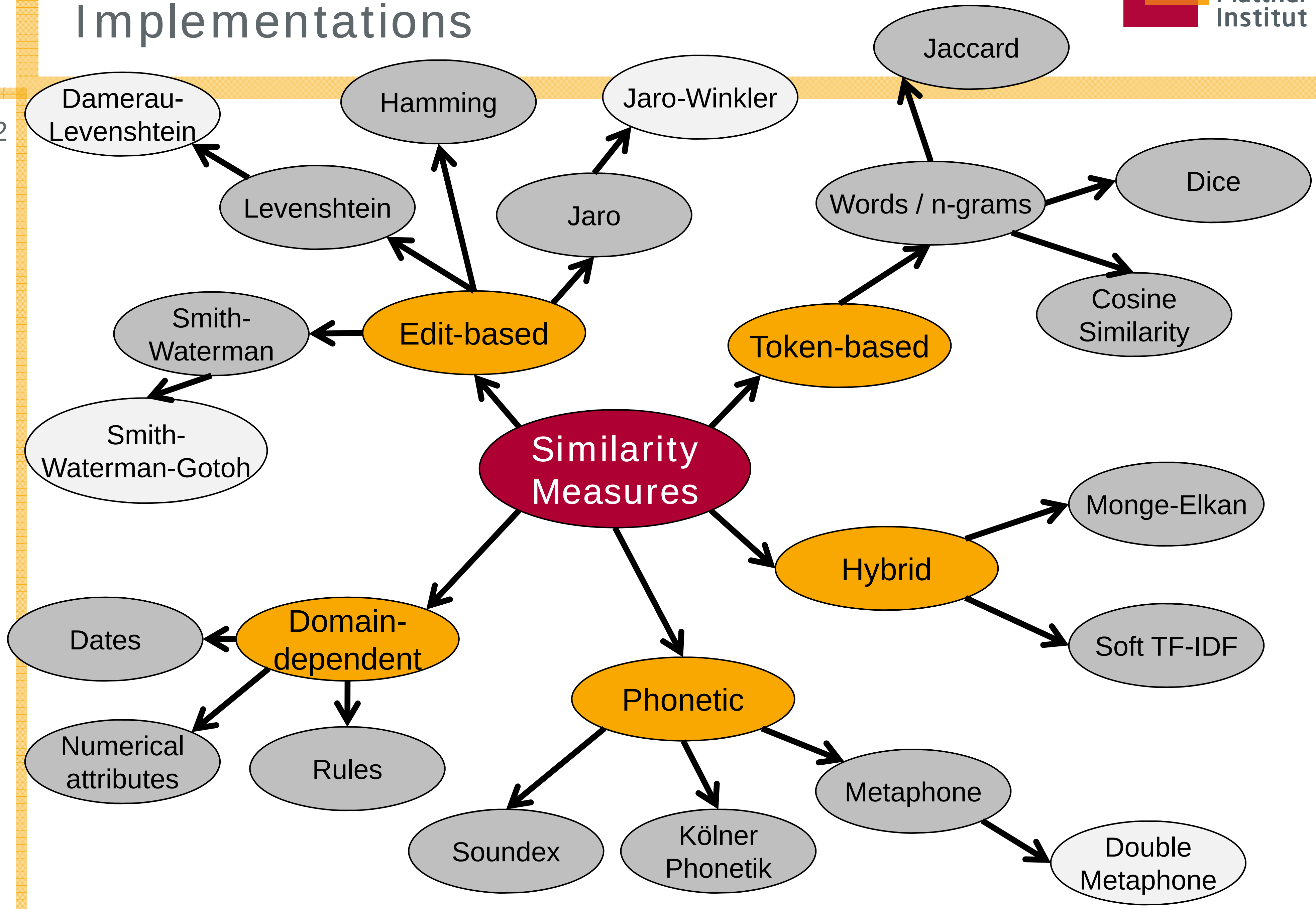
Numerical comparison

- $sim_{num\_abs}(n,m) = \begin{cases} 1 - \left(\frac{|n-m|}{d_{max}}\right) & if\ |n-m| < d_{max} \\ 0 & else \end{cases}$
  - Linear extrapolation between 0 and d_{max}
- Example:
  - d_{max} = \$1,000
  - $sim_{num\_abs}(2{,}000,\ 2{,}500) = 1 - \frac{500}{1{,}000} = 0.5$
  - $sim_{num\_abs}(200{,}000,\ 200{,}500) = 1 - \frac{500}{1{,}000} = 0.5$
- $sim_{num\_perc}(n,m) = \begin{cases} 1 - \left(\frac{pc}{pc_{max}}\right) & if\ pc < pc_{max} \\ 0 & else \end{cases}$
  - $pc = \frac{|n-m|}{\max(|n|,|m|)} \cdot 100$ is percentage difference
  - pc_{max} = 33%
  - $sim_{num\_perc}(2{,}000,\ \ 2{,}500) = 1 - \frac{20}{33} = 0.394$ because $pc = \frac{|2{,}000-2{,}500|}{2{,}500} \cdot 100 = 20$
  - $sim_{num\_perc}(200{,}000,\ \ 200{,}500) = 1 - \frac{0{,}25}{33} = 0.993$ because $pc = \frac{500}{200{,}500} \cdot 100 = 0{,}25\%$

Time and space comparisons

- Calculate difference in days and use sim_{num_abs}
- Special cases
  - Swapped day and month and both ≤ 12: Return some fixed similarity, e.g. 0.5
  - Single error in month could exceed d_{max}: Return some fixed similarity, e.g., 0.75
- Dates of birth can be converted to age (num days or num years)
  - Then apply numerical measures
  - $sim_{age\_perc}(n,m) = \begin{cases} 1 - \left(\frac{apc}{apc_{max}}\right) & if\ apc < apc_{max} \\ 0 & else \end{cases}$
  - $apc = \frac{|n-m|}{\max(|n|,|m|)} \cdot 100$ is percentage difference
- Geographical data: Compute distance based on some projection

Implementations



Similarity function packages

- SecondString
  - Java classes
  - All basic string comparisons
  - MongeElkan, SoftTFIDF
  - Similarity learner
  - http://sourceforge.net/projects/secondstring/
- SimMetrics
  - Java package
  - All basic string comparisons
  - Long sequences: Needleman-Wunsch, Smith-Waterman, Smith-Waterman-Gotoh
  - http://sourceforge.net/projects/simmetrics/
- Geographiclib for geographic similarity
  - http://geographiclib.sourceforge.net