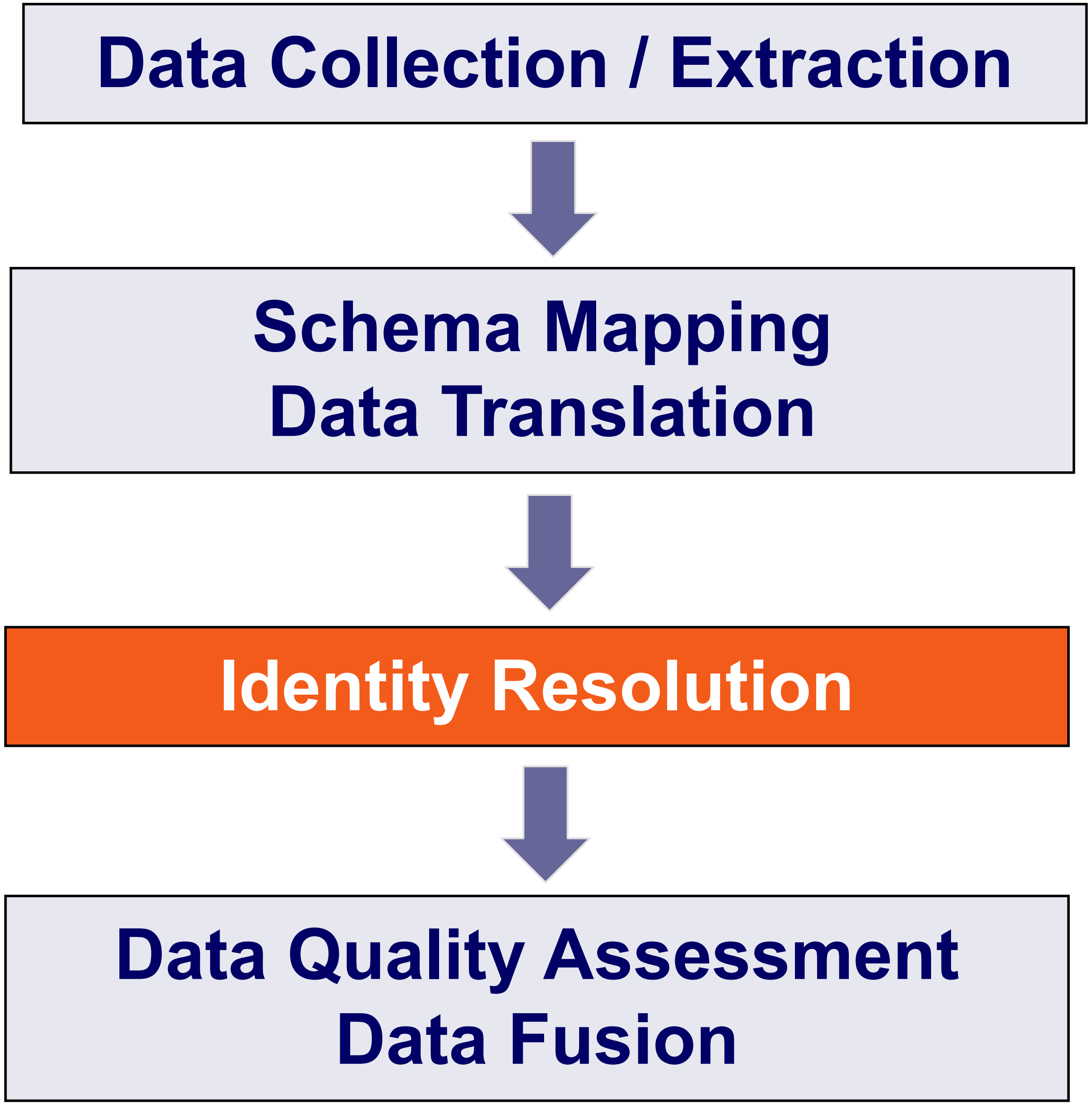
Web Data Integration

Identity Resolution

The Data Integration Process



Outline

1. Introduction
2. Entity Matching
3. Blocking
4. Evaluation
5. Similarity Measures – In Detail
6. Learning Matching Rules
7. Combining Schema and Entity Matching

1. Introduction

Goal of Identity Resolution:
Find all records that refer to the same real-world entity.

DB1	CID1243	Chris Miller	12/20/1982	Bardon Street; Melville	2 sales
DB2	344278	Christian Miller	2/20/1982	7 Bardon St., Melvile	24 sales
DB2	427859	Chris Miller	12/14/1973	Bardon St., Madison	23 sales

– The problem appears whenever

1. a single data source is cleaned (deduplicated)
2. data from multiple sources is integrated

Negative Effects of Duplicates in Single Data Source

1. Unnecessary memory consumption
2. Inconsistencies between records (after updates)
3. Queries give you wrong results
   - Number of customers != SELECT COUNT(*) FROM customer
4. You just see parts and not the whole
   - wrong assessment of customer value for CRM
   - customers that exceed credit limits are not recognized
   - multiple mailings of same catalog to same household
   - …

Ironically, “Identity Resolution” has many Synonyms

Duplicate detection

Record linkage

Data matching

Deduplication

Reference matching

Object identification

Entity resolution

Doubles

Householding

Entity clustering

Fuzzy match

Object consolidation

Identity uncertainty

Approximate match

Match

Hardening soft databases

Merge/purge

Reference reconciliation

Mixed and split citation problem

The Two Central Challenges of Identity Resolution

- **Challenge 1: Representations of the same real-world entity are not identical**
  - fuzzy duplicates

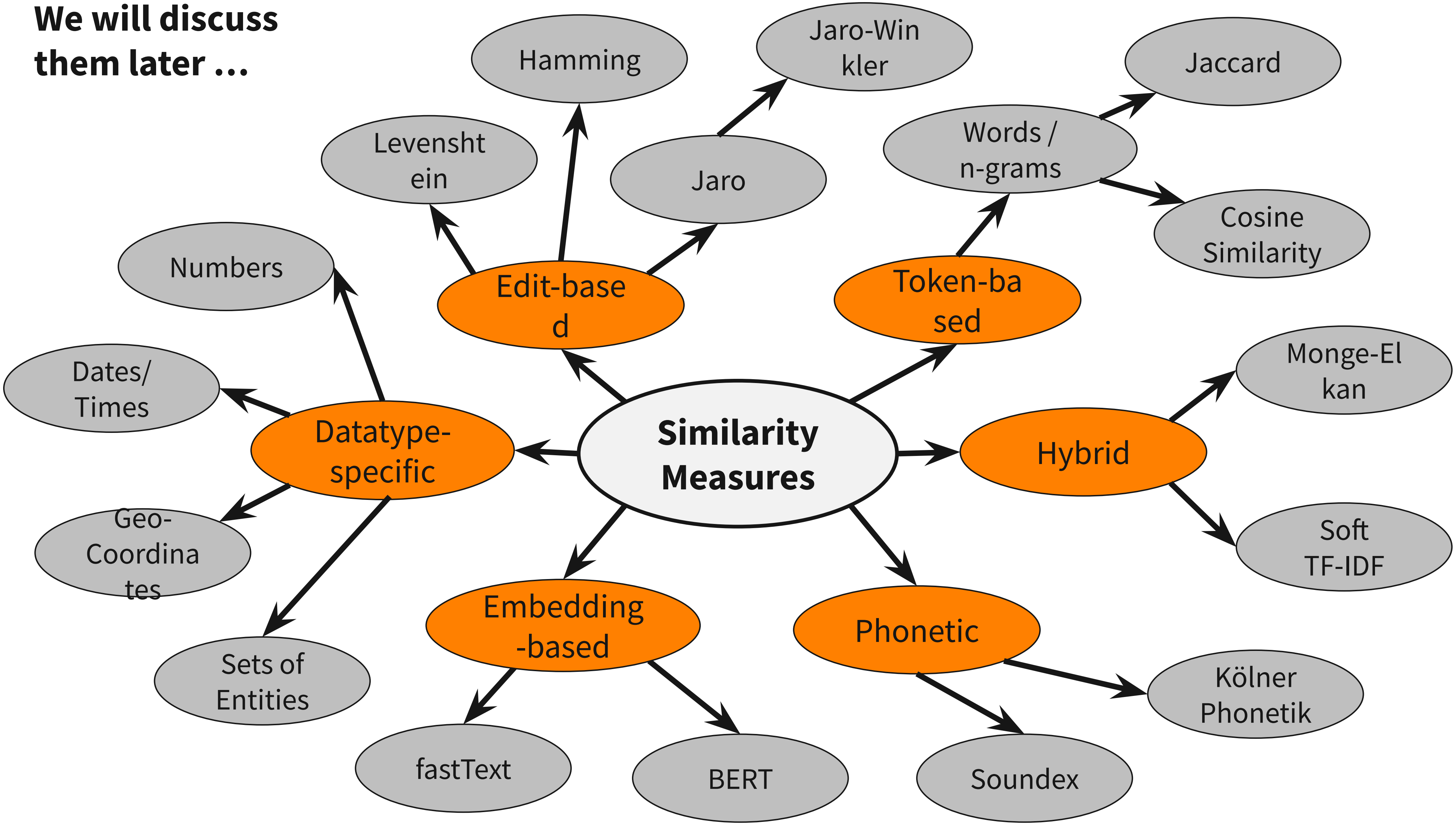
Chris Miller	12/20/1982	Bardon Street; Melville
Christian Miller	2/20/1982	7 Bardon St., Melvile
Chris Miller	12/14/1973	Bardon St., Madison

- Solution: Entity Matching
  - compare multiple attributes of the records using attribute-specific similarity measures
- Questions:
  1. Which attributes are relevant for the comparison?
  2. What is the right similarity measure for each attribute?
  3. How to combine the similarity scores of different attributes into a matching decision?

488941	britney spears	29	britent spears
40134	brittany spears	29	brittnany spears
36315	brittney spears	29	britttany spears
24342	britany spears	29	btiney spears
7331	britny spears	26	birttney spears
6633	briteny spears	26	breitney spears
2696	britteny spears	26	brinity spears
1807	briney spears	26	britenay spears
1635	brittny spears	26	britneyt spears
1479	brintey spears	26	brittan spears
1479	britanny spears	26	brittne spears
1338	britiny spears	26	btittany spears
1211	britnet spears	24	beitney spears
1096	britiney spears	24	birteny spears
991	britaney spears	24	brightney spears
991	britnay spears	24	brintiny spears
811	brithney spears	24	britanty spears
811	brtiney spears	24	britenny spears
664	birtney spears	24	britini spears
664	brintney spears	24	britnwy spears
664	briteney spears	24	brittni spears
601	bitney spears	24	brittnie spears
601	brinty spears	21	biritney spears
544	brittaney spears	21	birtany spears
544	brittnay spears	21	biteny spears
364	britey spears	21	bratney spears
364	brittiny spears	21	britani spears
329	brtney spears	21	britanie spears
269	bretney spears	21	briteany spears
269	britneys spears	21	brittay spears
244	britne spears	21	brittinay spears
244	brytney spears	21	brtany spears
220	breatney spears	21	brtiany spears
220	britiany spears	19	birney spears
199	britnney spears	19	brirtney spears
163	britnry spears	19	britnaey spears
147	breatny spears	19	britnee spears
147	brittiney spears	19	britony spears
147	britty spears	19	brittanty spears
147	brotney spears	19	britttney spears
147	brutney spears	17	birtny spears
133	britteney spears	17	brieny spears
133	briyney spears	17	brintty spears
121	bittany spears	17	brithy spears

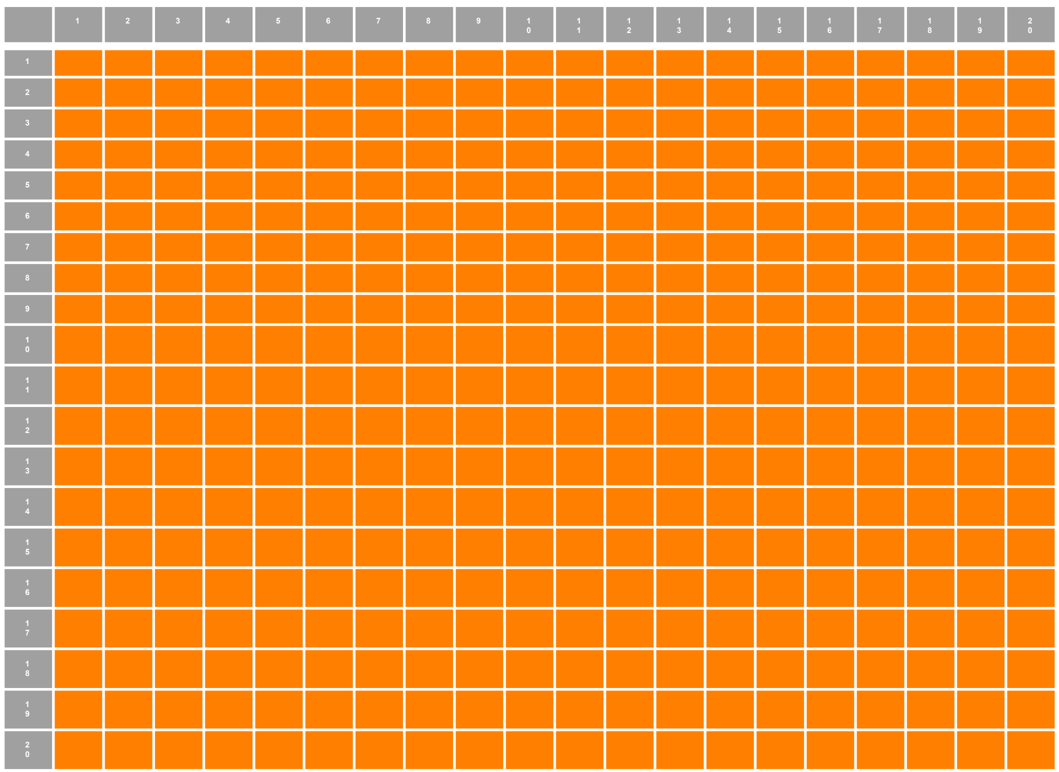
A Wide Range of Similarity Measures Exists

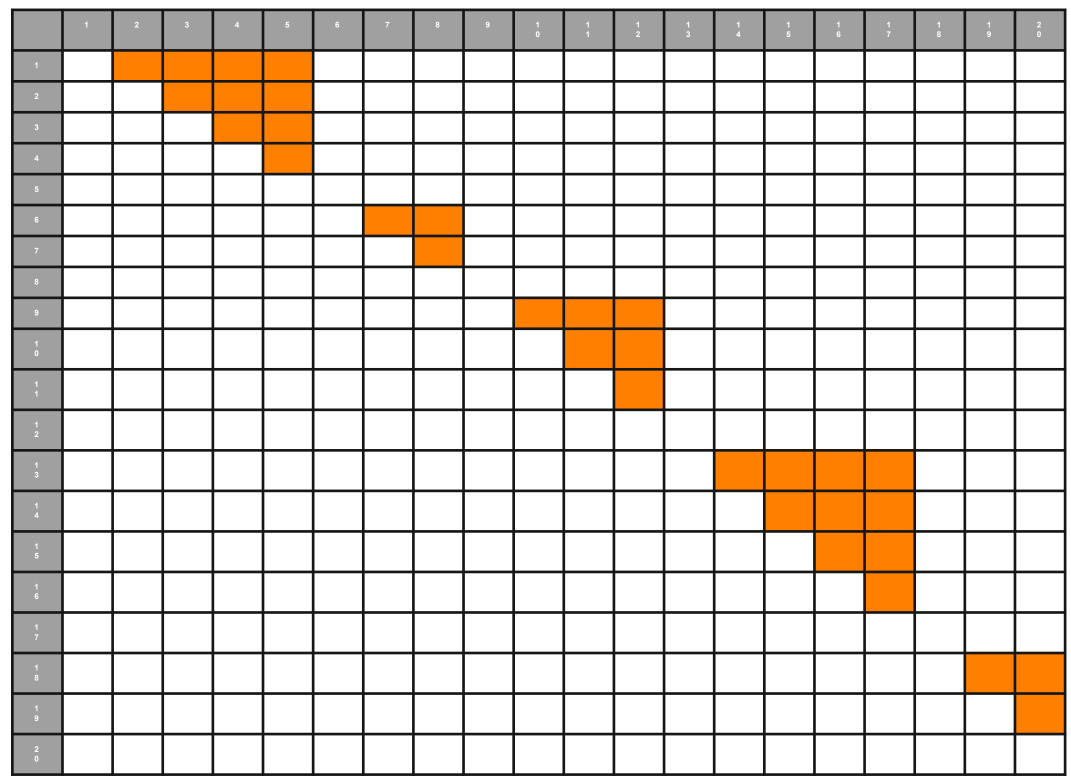
We will discuss them later …



The Two Central Challenges of Identity Resolution

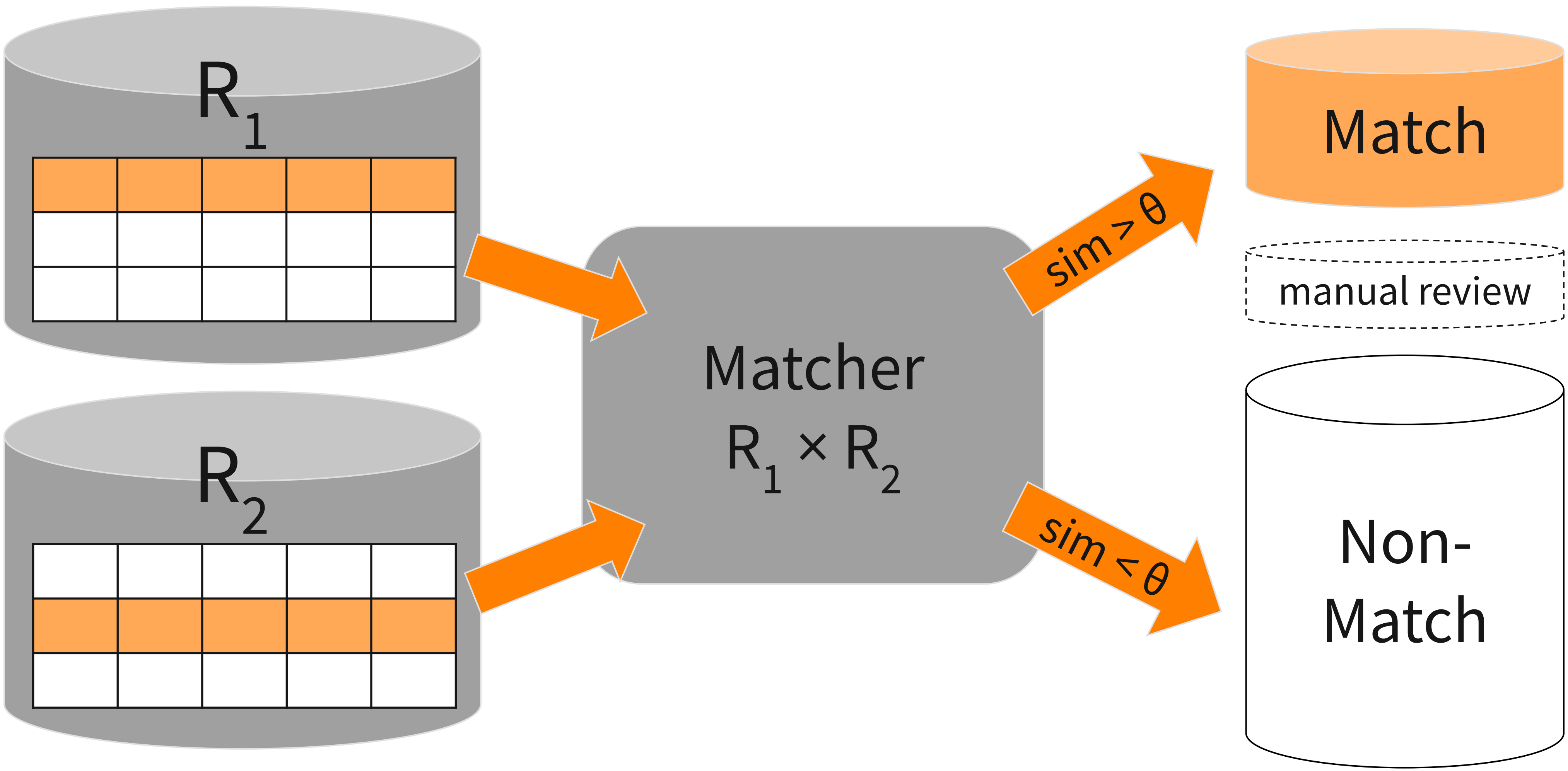
- **Challenge 2: Data sets are large**
  - quadratic runtime complexity: Comparing every pair of records is too expensive
- Solution: Blocking methods
  - avoid unnecessary comparisons





2. Entity Matching

Challenge 1: Representations of the same real-world entity are not identical



2.1 Linearly Weighted Matching Rules

- Compute the similarity score between records x and y as a linearly weighted combination of individual attribute similarity scores
  - $sim(x,y) = \sum_{i=1}^{n} \alpha_i * sim_i(x,y)$
  - n is number of attributes in each table
  - $sim_i(x,y)$ is similarity score between the i-th attributes of x and y
  - $\alpha_i \in [0,1]$ is a pre-specified weight that indicates the importance of the i-th attribute for the matching decision

Example Matching Rule

Table X

	Name	Phone	City	State
x_1	Dave Smith	(608) 395 9462	Madison	WI
x_2	Joe Wilson	(408) 123 4265	San Jose	CA
x_3	Dan Smith	(608) 256 1212	Middleton	WI

(a)

Table Y

	Name	Phone	City	State
y_1	David D. Smith	395 9426	Madison	WI
y_2	Daniel W. Smith	256 1212	Madison	WI

(b)

Matches

(x_1, y_1)
(x_3, y_2)

(c)

$sim(x,y) = 0.3s_{name}(x,y) + 0.3s_{phone}(x,y) + 0.1s_{city}(x,y) + 0.3s_{state}(x,y)$

$s_{name}(x,y)$: using the Jaro-Winkler similarity measure

$s_{phone}(x,y)$: based on edit distance between x's phone (after removing area code) and y's phone

$s_{city}(x,y)$: based on edit distance

$s_{state}(x,y)$: based on exact match; yes 1, no 0

2.2 Non-Linear Matching Rules

- Often better than linear rules, but require specific domain knowledge.
- Example 1: Two persons match if names match approximately and either phones match exactly or addresses match exactly
  1. if $sim_{name}(x,y) < 0.8$ then return “not matched”
  2. otherwise if $equal_{phone}(x,y)$ = true then return “matched”
  3. otherwise if $equal_{city}(x,y)$ = true and $equal_{state}(x,y)$ = true then return “matched”
  4. otherwise return “not matched”
- Example 2: Two genes match if their names match approximately and any of the different, alternative gene identifiers match exactly (deals with missing values)
  - if $\max(equal_{genID}(x,y), equal_{componentID}(x,y), equal_{structureID}(x,y)) = 1$
  - and $sim_{name}(x,y) > 0.7$
  - then return “matched”

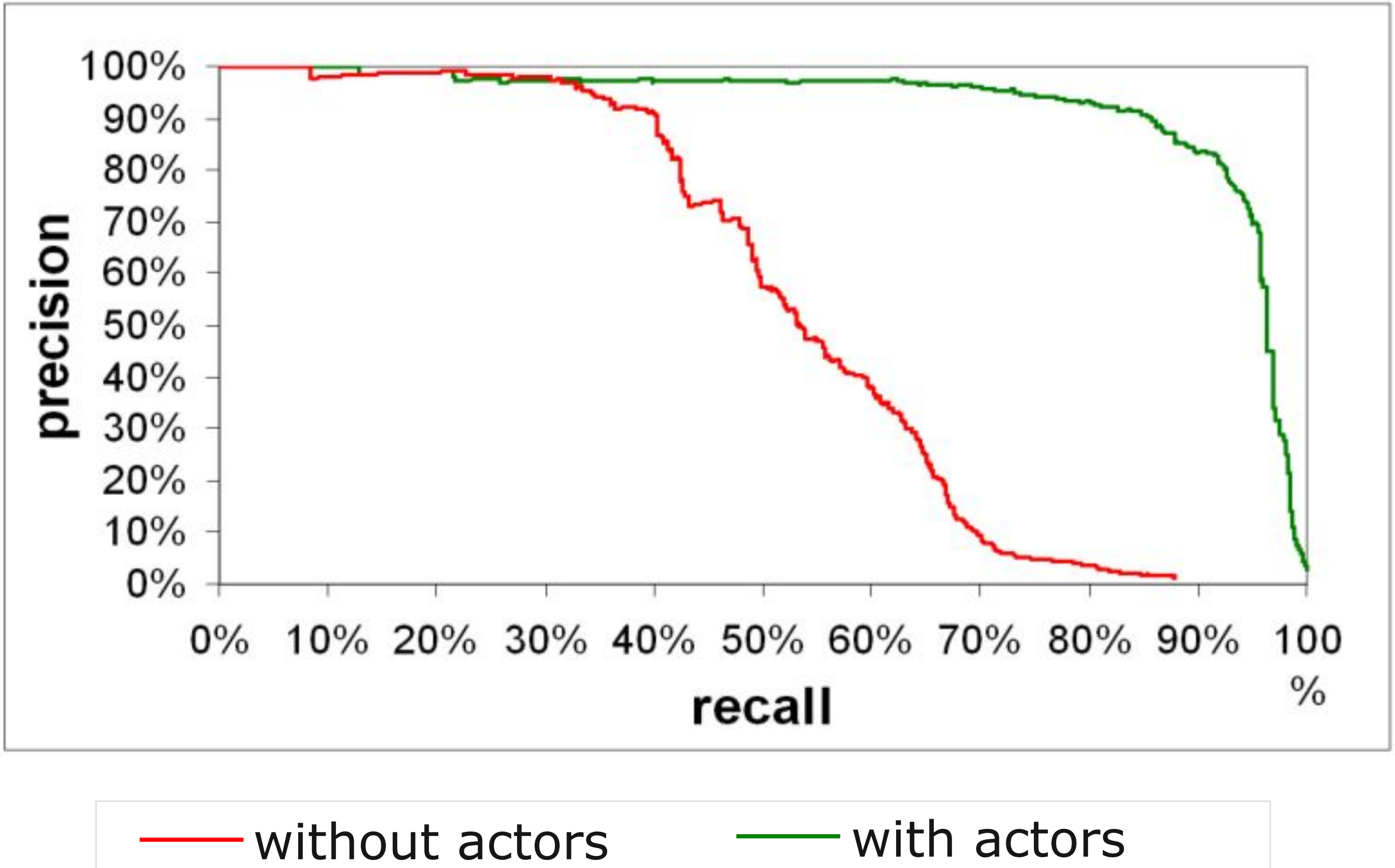
2.3 Data Gathering for Matching

- Not only values of the records to be compared, but also values of related records are relevant for the similarity computation
  - Movies: Actors
  - CDs: Songs
  - Persons: Spouse, children, employer, publications
- Example: The movie names look quite similar to the edit distance measure

ID	Actor
1	Ewan McGregor
2	Natalie Portman
3	Mark Hamill
4	Harrison Ford
5	Ewan McGregor
6	Natalie Portman

ID	Film
1	Star Wars 1
2	Star Wars 4
3	Star War 1

Example: Matching Films



2.4 Data Preprocessing for Matching

In order to enable similarity measures to compute reliable scores, the data needs to be normalized.

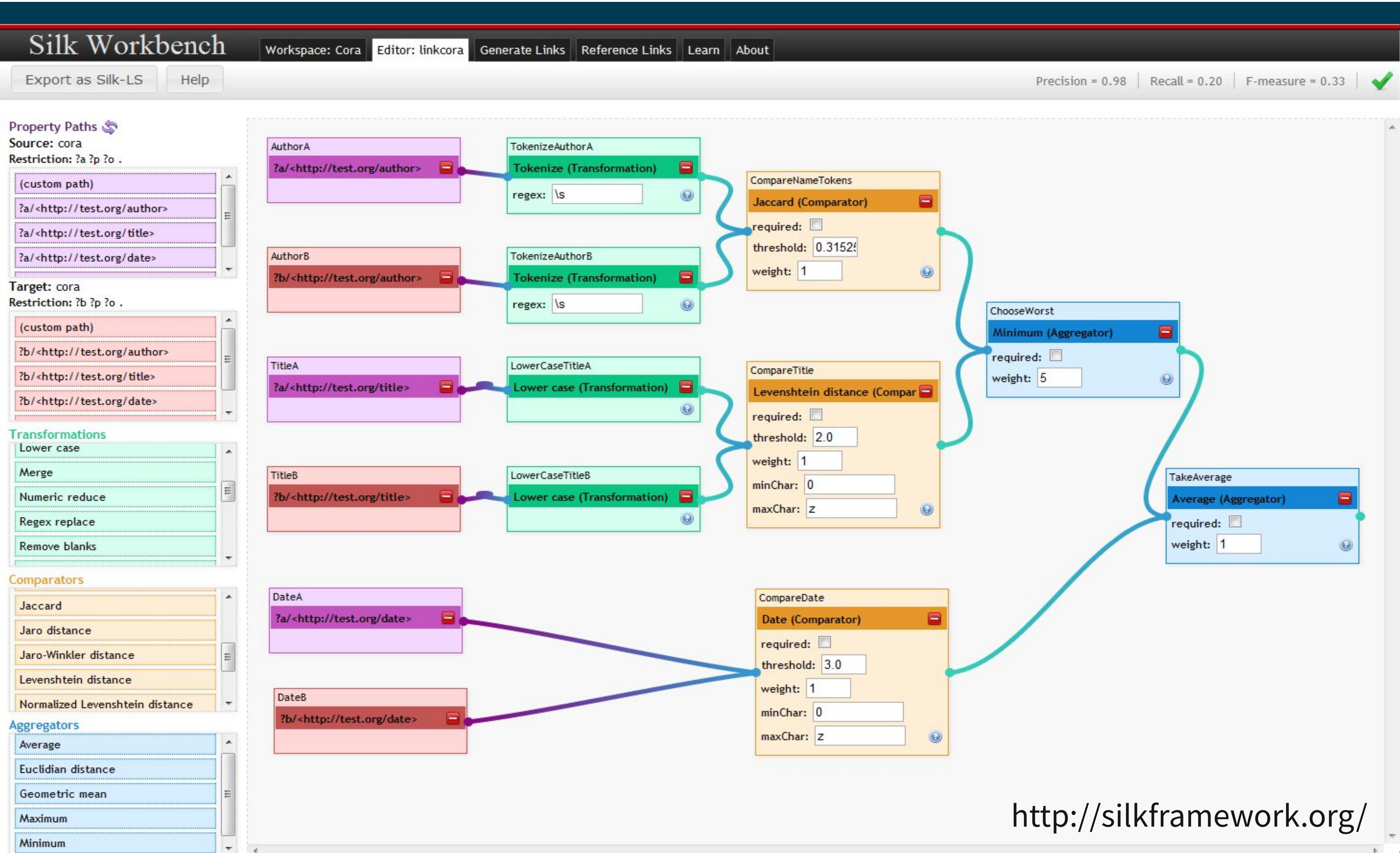
- Normalize spelling
  - lower case everything: Müller and mueller □ mueller
  - remove punctuation: U.S.A □ usa
- Remove stopwords
  - The Netherlands □ netherlands
- Normalize value formats and units of measurement
  - +49 621 181 2677 and (621) 181 2677 □ 496211812677
  - 1000 MB and 1 GB □ 1000 MB
- Normalize abbreviations and synonyms/surface forms
  - Inc. □ Incorporated, Mr.□ Mister, USA □ United State of America
  - using domain-specific lists of abbreviations and synonyms/surface forms

Parsing and Translation

- Parsing
  - Extract attribute/value pairs from compound descriptions or titles
    - using regular expressions or attribute specific extractors (e.g list of all brands)
  - Often required for e-commerce data or postal addresses:
    - Apple MacBook Air MC968/A 11.6-Inch Laptop
    - Apple MacBook Air 11-in, Intel Core i5 1.60GHz, 64 GB, Lion 10.7
- Translation using external services
  - Geocoding
    - translate addresses into geo-coordinates and compare coordinates afterwards
    - e.g. using Google Geocoding API
  - Translate into target language
    - מנהיים Mannheim
    - e.g. using Google Translate API or other translation software

Petrovski, Bryl, Bizer: Integrating Product Data from Websites offering Microdata Markup. DEOS, 2014.
Kannan, et al: Matching unstructured Product Offers to structured Product Specifications. KDD, 2011.

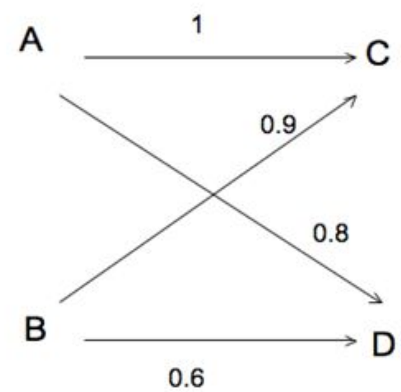
Example: Complex Matching Rule including Preprocessing



http://silkframework.org/

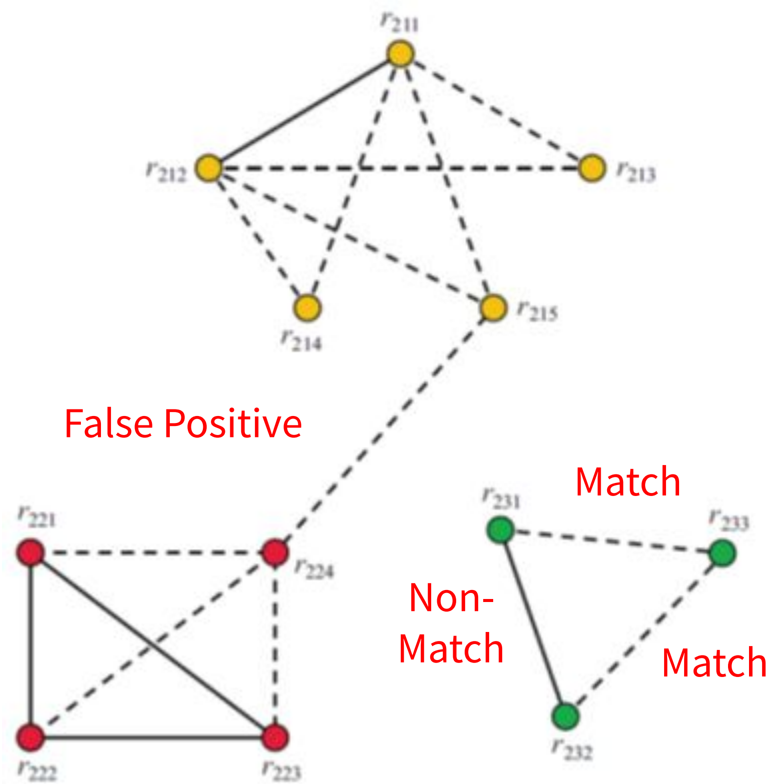
2.5 Local versus Global Matching

- Input: A matrix containing record similarities
- Output: A set of correspondences connecting pairs of matching records
- Local Matching
  - consider all pairs above threshold as matches
  - implies that one record can be matched with several other records
  - makes sense for duplicate detection within single data source
- Global Matching
  - enforce constraint that one record in data set A should only be matched to one record in data set B
  - makes sense for data sources that do not contain duplicates
  - Approaches:
    1. Bipartite pairs with the maximal sum of similarity values
    2. Stable marriage (see Chapter Schema Mapping)



2.6 Cluster Records using Pairwise Correspondences

- Goal: Create groups of records describing the same real-world entity from pairwise correspondences
  - relevant for matching multiple data sources and for the deduplication of a single source
- Simple Approach: Connected Components
  - transitive closure of pairwise correspondences
  - problem: correspondences might be inconsistent as they result from separate local decisions
- Smarter Approach: Correlation Clustering
  - cuts graph into coherent groups by minimizing disagreement with pairwise correspondences
    - Cohesion penalty: Non-matching records in cluster
    - Correlation penalty: Removing correspondences



Saeedi, et al.: Comparative Evaluation of Distributed Clustering Schemes for Multi-source Entity Resolution. ADBIS 2017.
Hassanzadeh, et al.: Framework for Evaluating Clustering Algorithms in Duplicate Detection. *VLDB Endowment*, 2009.

Summary: The Entity Matching Process

Gather Data for Matching

↓

Normalize Attribute Values

↓

Apply Attribute-specific Similarity Measures

↓

Combine Similarity Scores

↓

Decide Match/Non-Match

↓

Cluster Records based on Correspondences

3. Blocking

- Real world data sets are often large
- Problem: Quadratic complexity of matching process
  - comparing every pair of records is too expensive:
    - 100 customers □ 10,000 comparisons
    - 10,000 customers □ 100 million comparisons
    - 1,000,000 customers □ 1 trillion comparisons
  - Each comparison itself is also expensive as it involves calculating various similarity scores
    - calculation of a string similarity score often has quadratic complexity itself
- Solution: Reduce number of pairs of records that are compared by

1. avoiding unnecessary comparisons (next 3 slides)
   - no negative effect, but faster ☺
2. applying blocking methods that further reduce the number of comparisons
   - negative effect: True matches might be missed ☹

Number of comparisons: All pairs

Complexity: n²

	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20
1																				
2																				
3																				
4																				
5																				
6																				
7																				
8																				
9																				
10																				
11																				
12																				
13																				
14																				
15																				
16																				
17																				
18																				
19																				
20																				

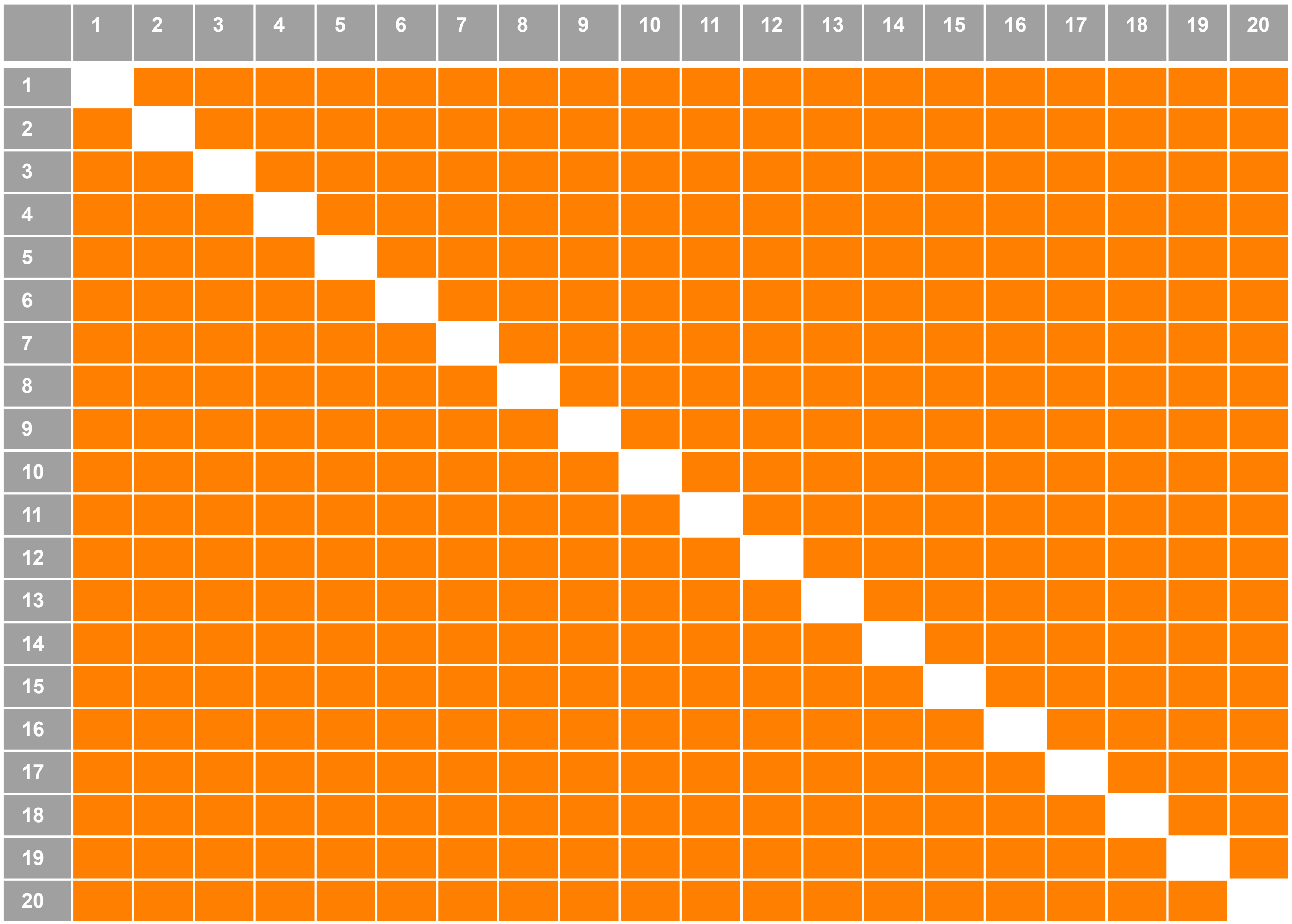
20 records

□□

400 comparisons

Reflexivity of Similarity

Complexity: n^2-n



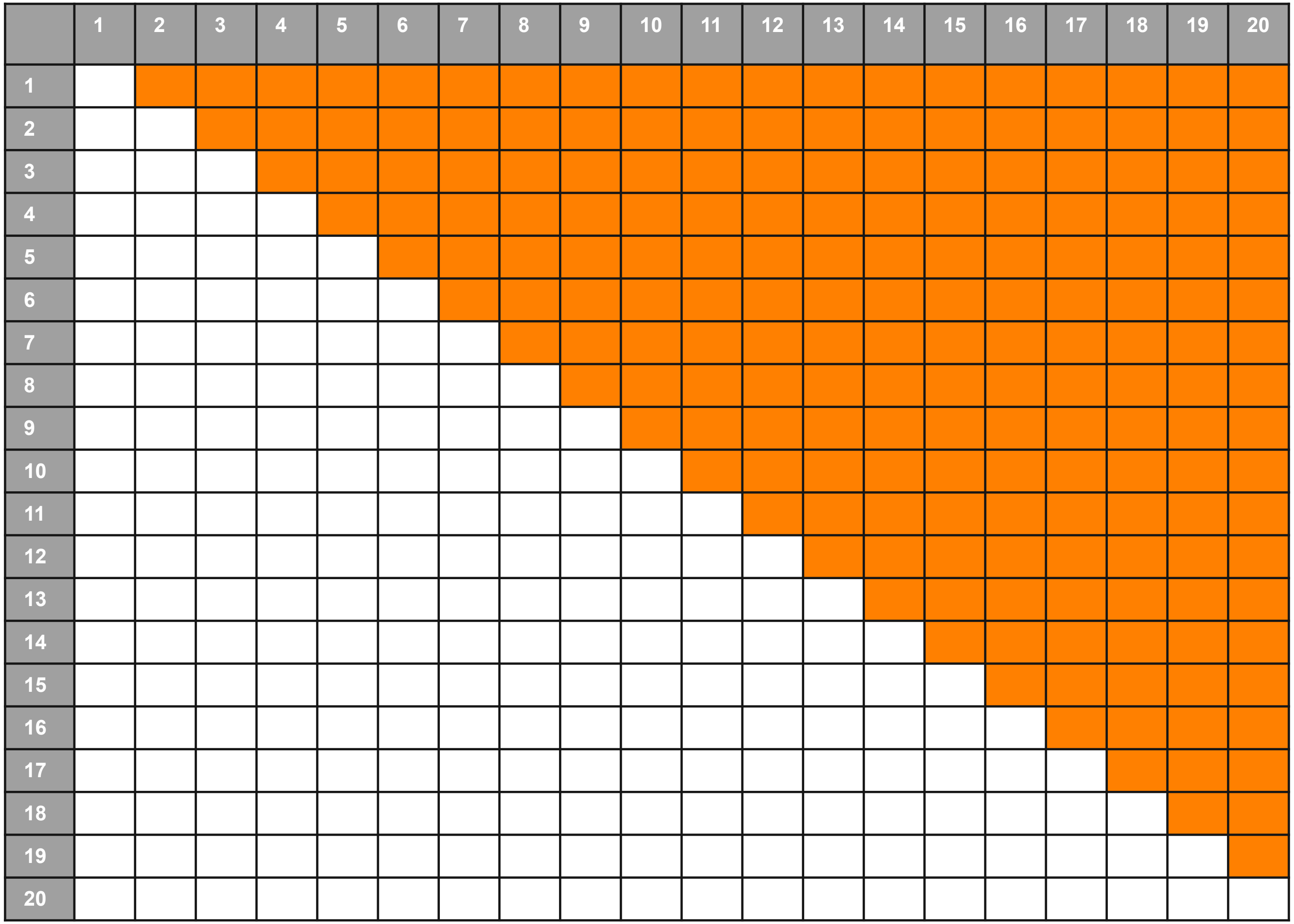
Similarity is reflexive:
sim(x, x) = 1

380
comparisons

- Applies to duplicate detection use case
- but not to two data sources use case

Symmetry of Similarity

Complexity: $(n^2-n) / 2$



Similarity is **symmetric**: sim(x,y) = sim(y,x)

190 comparisons

Still quadratic ☹

3.1 Standard Blocking

Idea: Reduce number of comparisons by partitioning the records into buckets and compare only records within each bucket.

− Examples:
  - partition customers by first two digits of their zip code
    - results in about 100 partitions for Germany
    - given about 100 customers per partition

     495,000 comparisons instead of 49,995,000

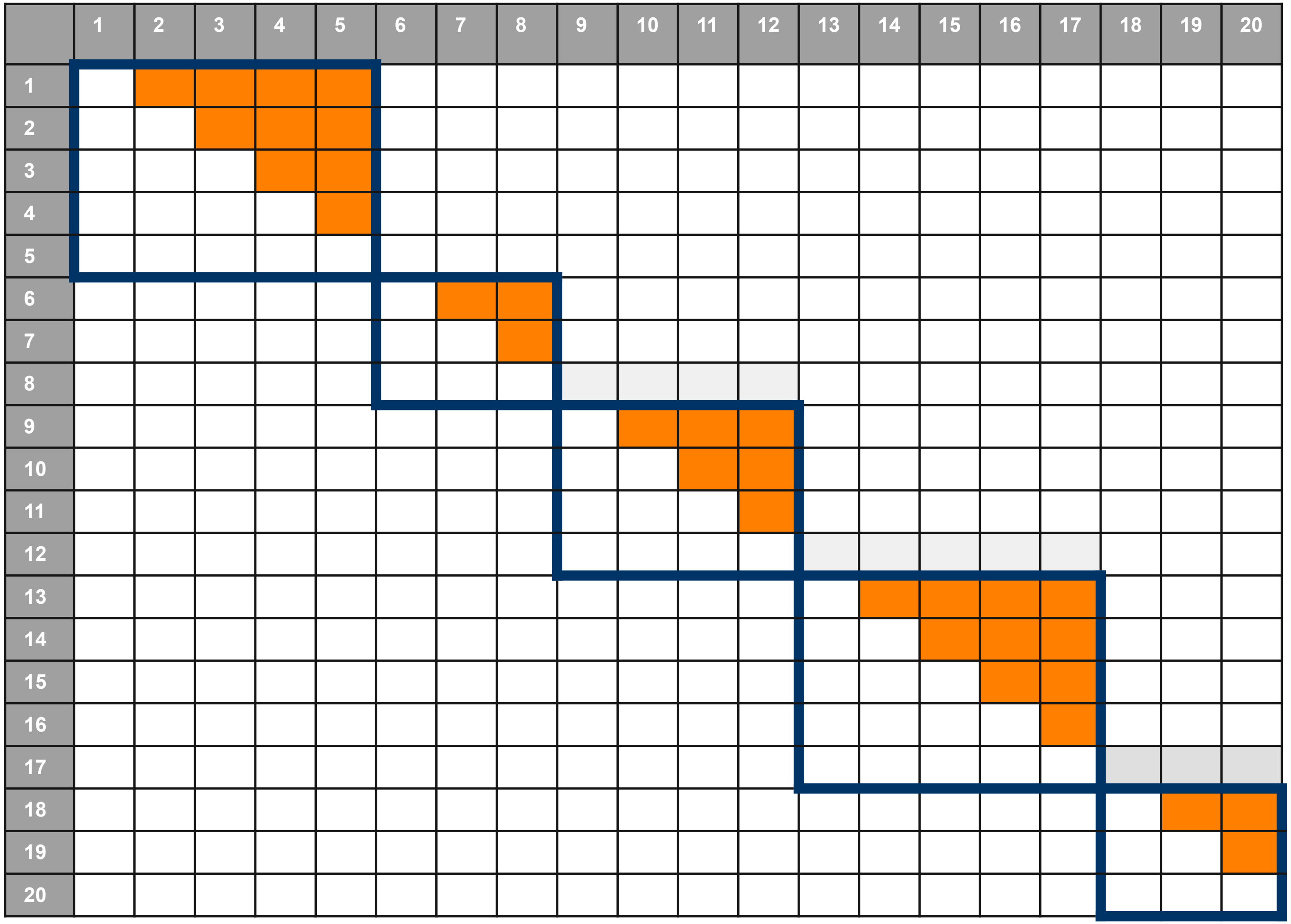
    + algorithm ~100 times faster

    - matches with wrong zip code might be missed
  - partition books by publisher
  - partition people by first two characters of surname

− Blocking is also called hashing or partitioning

Source: wikipedia.de

Standard Blocking



32 comparisons

+ much faster than 190 comparisons

- might miss Matches ☹

Choosing a Good Blocking Key

- Reduction ratio depends on effectiveness of blocking key
  - high: if records are equally distributed over buckets
  - low: if majority of the records end up in one bucket
    - example: 90% of all customers are from Mannheim
  - possible workaround: build sub-buckets using a second blocking attribute
    - block houses by zip first. Afterward, block within each bucket by street name
- Recall depends on actually matching pairs being kept (compared)
  - pairs might not compared as their blocking key values differ
    - typo in zip code, customer has moved
  - possible workaround: use only first letters as they often contain less typos
- Example combining both workarounds

FirstName	Name	Adresse	ID
Sal	Stolpho	123 First St.	456780
Mauricio	Hernandez	321 Second Ave	123456

Blocking Key
STOSAL
HERMAU

3.2 The Sorted Neighborhood Method (SNM)

Idea: Sort records so that similar records are close to each other. Only compare records within a small neighborhood window.

1. Generate key
   - e.g. first 3 letters of social security number + first 3 letters of surname
2. Sort by key
   - so that similar records end up close to each other
3. Slide window over sorted records
   - match each record with only the next w-1 records, where w is a pre-specified window size

3. Slide window w=2

FirstName	Surname	Address	SSN
Mauricio	Hernandez	321 Second Ave	123456
All	Stolpho	123 First St.	456780
Sal	Stolpho	123 First St.	456780
Sal	Stelfo	123 First Street	456789

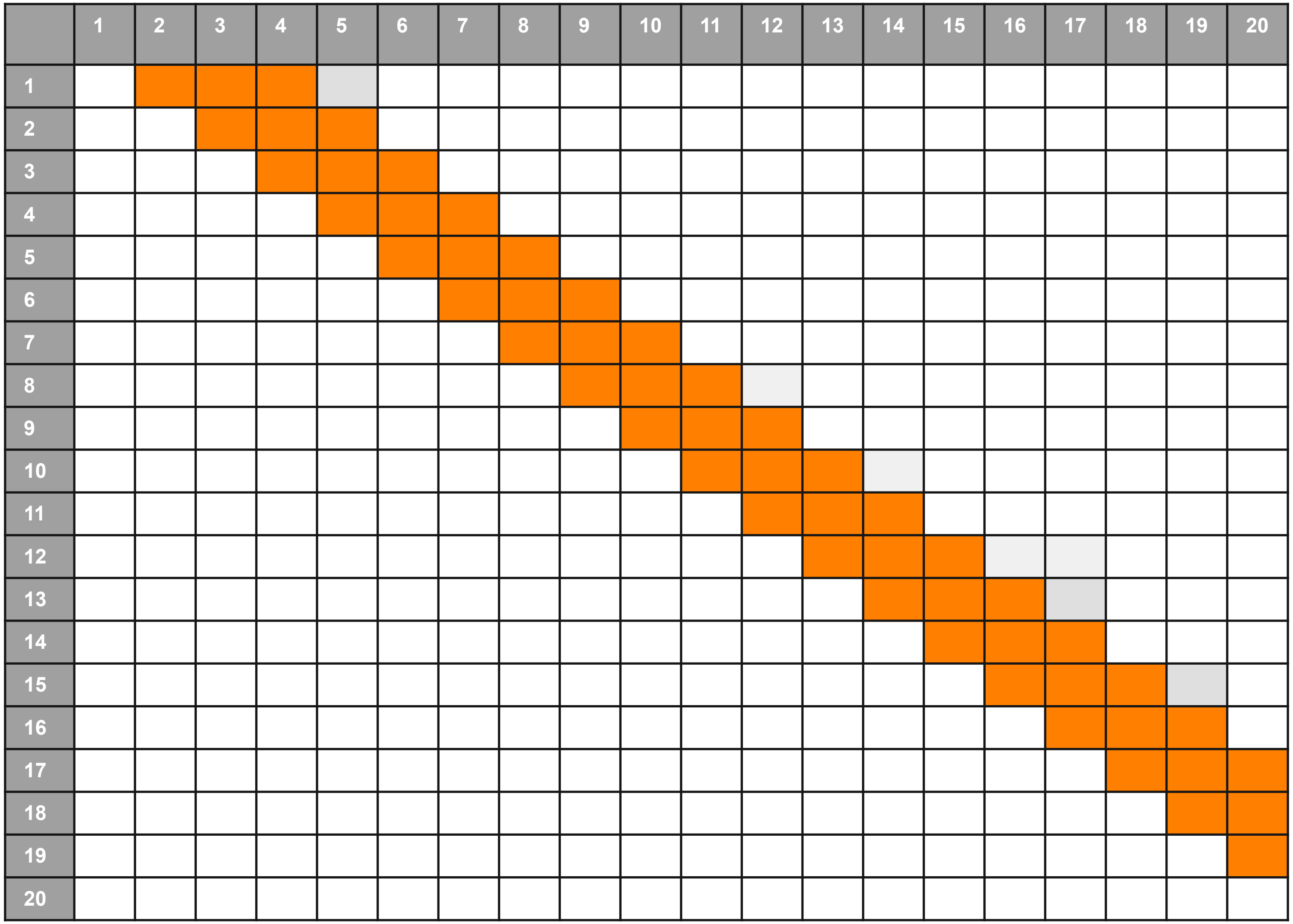
1. Generate key

Key
123HER
456STO
456STO
456STE

2. Sort

The Sorted Neighborhood Method (SNM)

Window size = 4



54 comparisons

+ no problem with different bucket sizes

Complexity:

1. Key generation: O(n)
2. Sorting: O(n*log(n))
3. Comparisons: O(n*w)

Challenges when Applying the SNM

- Choice of Blocking Key
  - SNM assumes that records that are likely to match fall within the window
  - Thus, key should be **strongly “discriminative”** and bring together records that are likely to match, and pushes apart records that are not
    - example keys: social sec, student ID, two characters of first + surname
- Choice of Window Size
  - Depends on the types and frequency of the errors/typos in the data
  - Practical experience: w = 20 often a good compromise
- Workaround: Use Multi-Pass Approach
  1. Run SNM several times with different blocking keys
     - use simple keys and a small w, e.g. 1. social sec, 2. two characters first + surname
  2. Merge sets of matches found in each run
  - Less efficient, but much more effective than single-pass

3.3 Token Blocking for Textual Attributes

- Identifying attributes are often rather textual, e.g.
  - Product names: Samsung Galaxy S10 SM-G975, 128GB, 8GB RAM
  - Names of local business: Wong Restaurant, Hoy Wong Greenwich
- Token Blocking
  - builds an inverted index that associates every token with all entities containing it in their attribute values
  - using only the identifying attribute or a concatenation of multiple attributes
  - afterwards, all pairs that sharing at least one (or more) tokens are compared
- N-Gram Blocking
  - variation of token blocking that uses character n-grams in order to deal with typos
  - n=3 : men, end, edo, ...

Set X

1: {lake, mendota}
2: {lake, monona, area}
3: {lake, mendota, monona, dane}

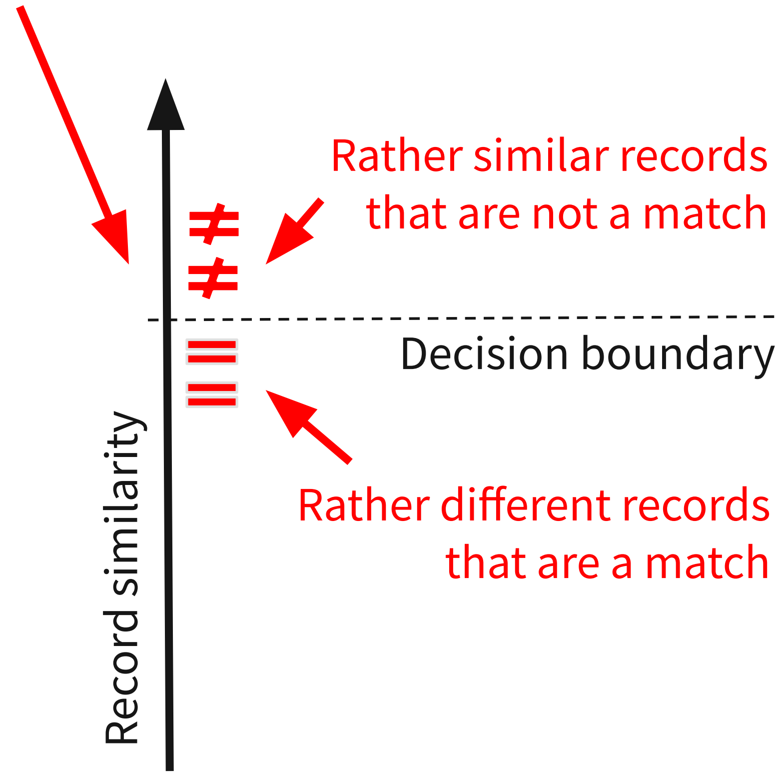
Set Y

4: {lake, monona, university}
5: {monona, research, area}
6: {lake, mendota, monona, area}

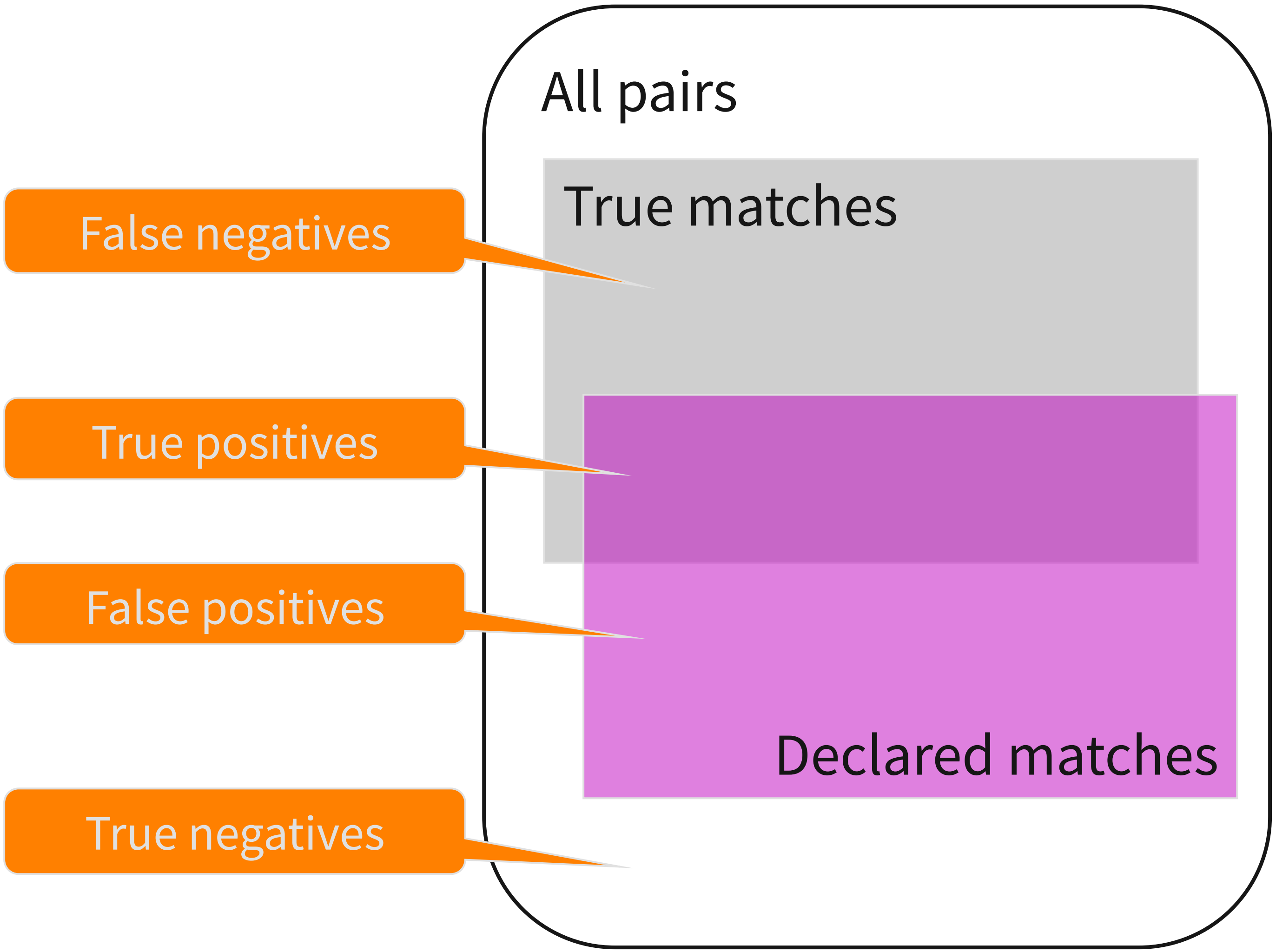
Terms in Y	ID Lists
area	5
lake	4, 6
mendota	6
monona	4, 5, 6
research	5
university	4

4. Evaluation

- You need ground truth (gold standard) for the evaluation
- To create a gold standard, manually label a set of record pairs as matches or non-matches including corner cases
- Rule of thumb for creating a *suitable* gold standard with *acceptable* manual effort:
  1. match records using several simple matching techniques (similar to multi-pass blocking) and sort record pairs according to their similarity
  2. use existing information about matches (e.g. ISBN or GTIN numbers that exist in multiple sources)
  3. manually verify a fair amount of the resulting pairs (e.g. 500 pairs) including
     1. matching record pairs (randomly chosen, 20% of GS)
     2. corner cases (30% of GS)
     3. non-matching record pairs (randomly chosen, 50% of GS)



Evaluation Metrics: Precision, Recall & F1



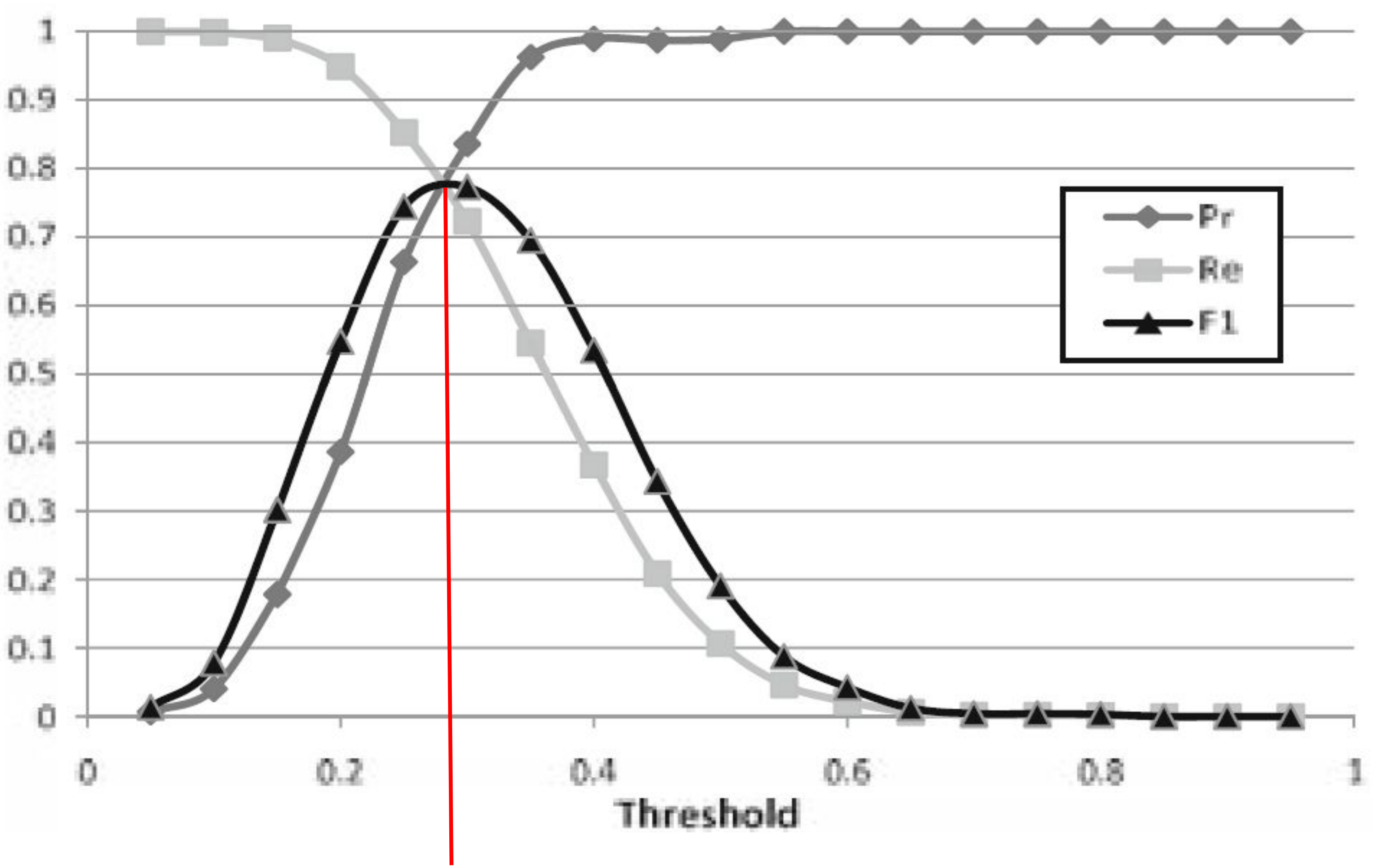
$$\text{Precision} = \frac{\text{True positives}}{\text{Declared matches}}$$

$$\text{Recall} = \frac{\text{True positives}}{\text{True matches}}$$

$$\text{F1-Measure} = \frac{2 \cdot \text{Precision} \cdot \text{Recall}}{\text{Precision} + \text{Recall}}$$

Accuracy is not a good metric as true negatives usually dominate overall result.

F1-Measure Graph

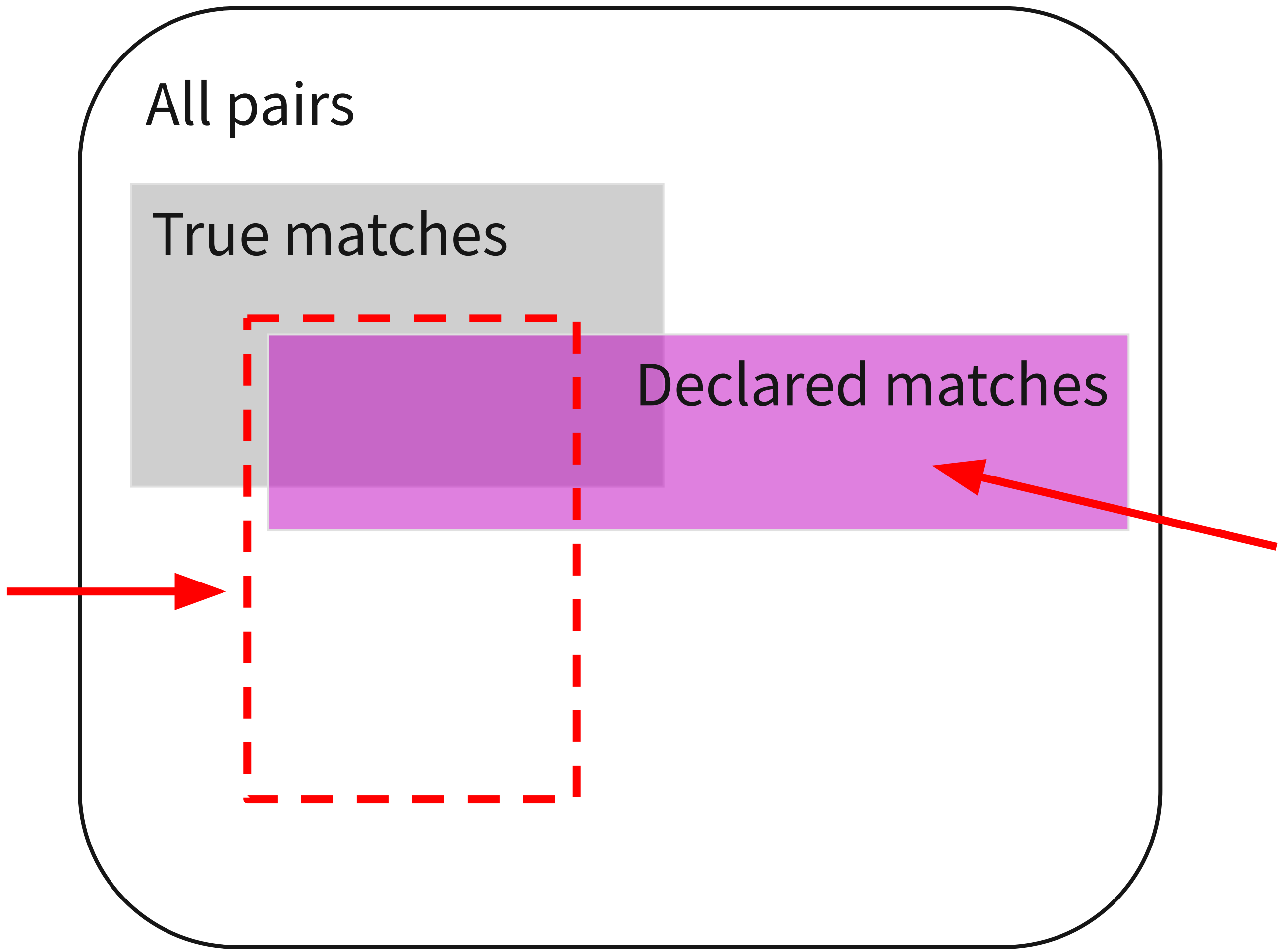


Optimal threshold of linearly weighted matching rules

Gold Standard Pairs versus All Pairs

Be aware that the selection bias of the record pairs in gold standard influences the evaluation result (and the data fusion quality).

Gold standard pairs

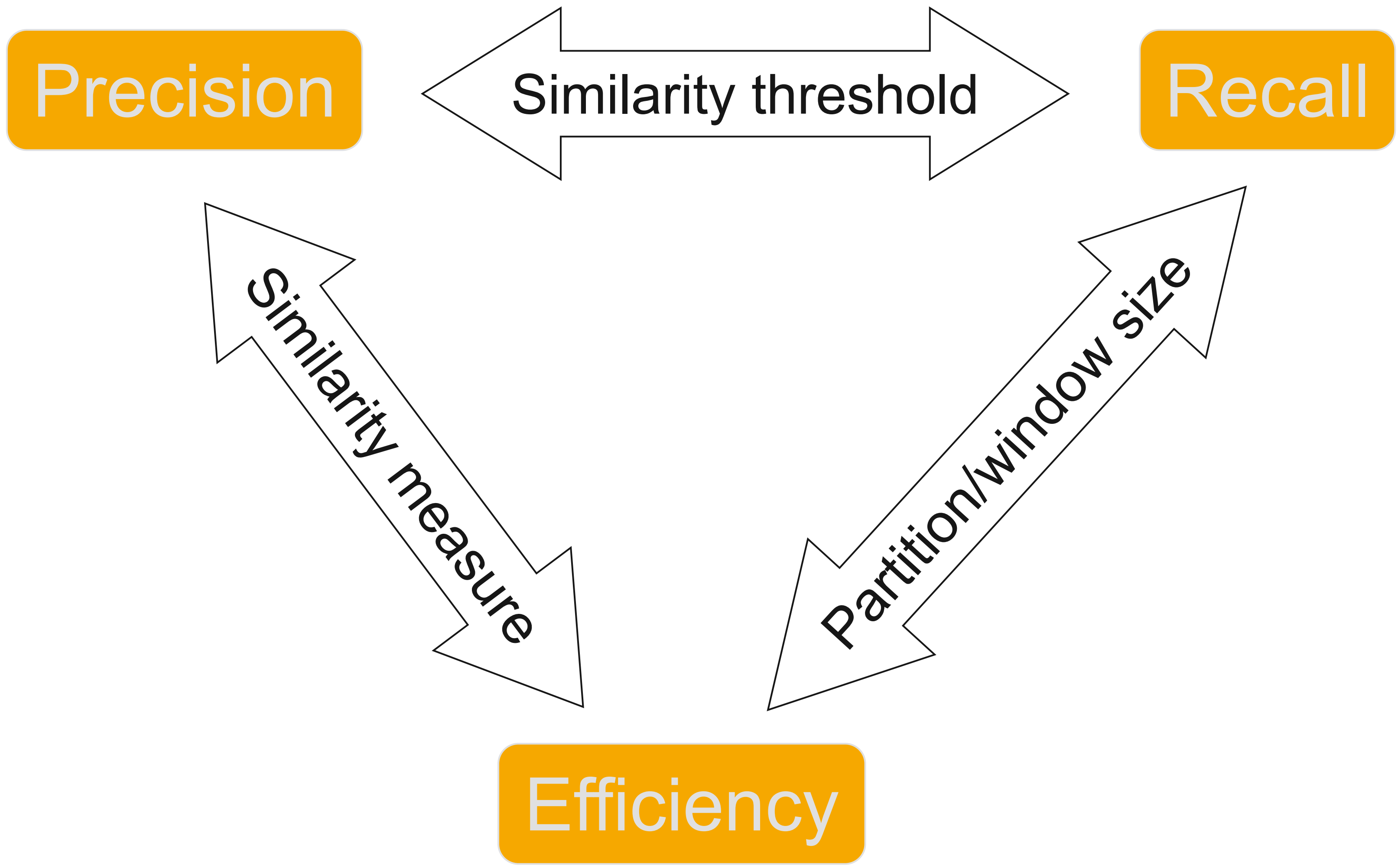


Additional wrong correspondences lead to unnaturally large clusters which might confuse the fusion method later

Efficiency Measures

- Besides of the quality of the matching rule, the quality of the blocking method is also important
- Option 1: Runtime measurements
  - but: different hardware, replicability problematic
- Option 2: Measure how well/poor the blocking method filters the candidate pairs
  - by which ratio does the blocking method reduce the number of comparisons?
  - how many true positives are missed?
- Reduction Ratio $= 1 - \frac{pairs_{afterBlocking}}{pairs_{beforeBlocking}}$
- Pairs Completeness = $matches_{afterBlocking}$ / $matches_{beforeBlocking}$
- Pairs Quality = $matches_{afterBlocking}$ / all $pairs_{selectedByBlocking}$

Evaluating Identity Resolution



Evaluation Datasets

Matching methods should be evaluated using the same datasets in order to make the results comparable.

1. DBLP-ACM-Scholar, Amazon-Google Products Datasets

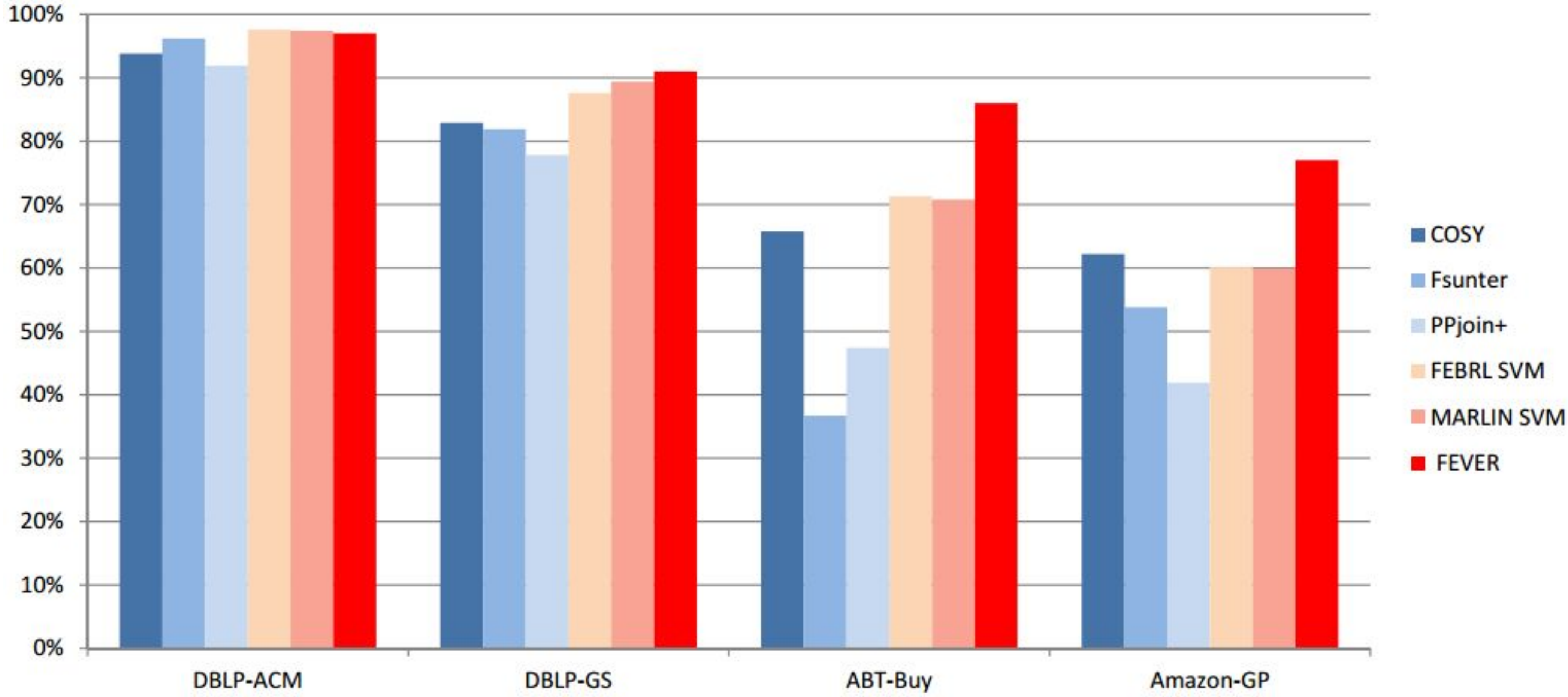
Match task		Source size (#entities)		Mapping size (#correspondences)		
Domain	Sources	Source 1	Source 2	Full input mapping (cross product)	Reduced input mapping (blocking)	perfect match result
Bibliographic	DBLP-ACM	2,616	2,294	6 million	494,000	2224
	DBLP-Scholar	2,616	64,263	168.1 million	607,000	5343
E-commerce	Amazon-GoogleProducts	1,363	3,226	4.4 million	342,761	1300
	Abt-Buy	1,081	1,092	1.2 million	164,072	1097

   - Köpcke, Thor, Rahm: Evaluation of entity resolution approaches. VLDB 2010.

2. Magellan/Deepmatcher Evaluation Datasets
   - https://github.com/anhaidgroup/deepmatcher/blob/master/Datasets.md

3. WDC Training Dataset and Gold Standard for Large-Scale Product Matching
   - http://webdatacommons.org/largescaleproductcorpus/

F-Measure for Bibliographic and E-Commerce Data

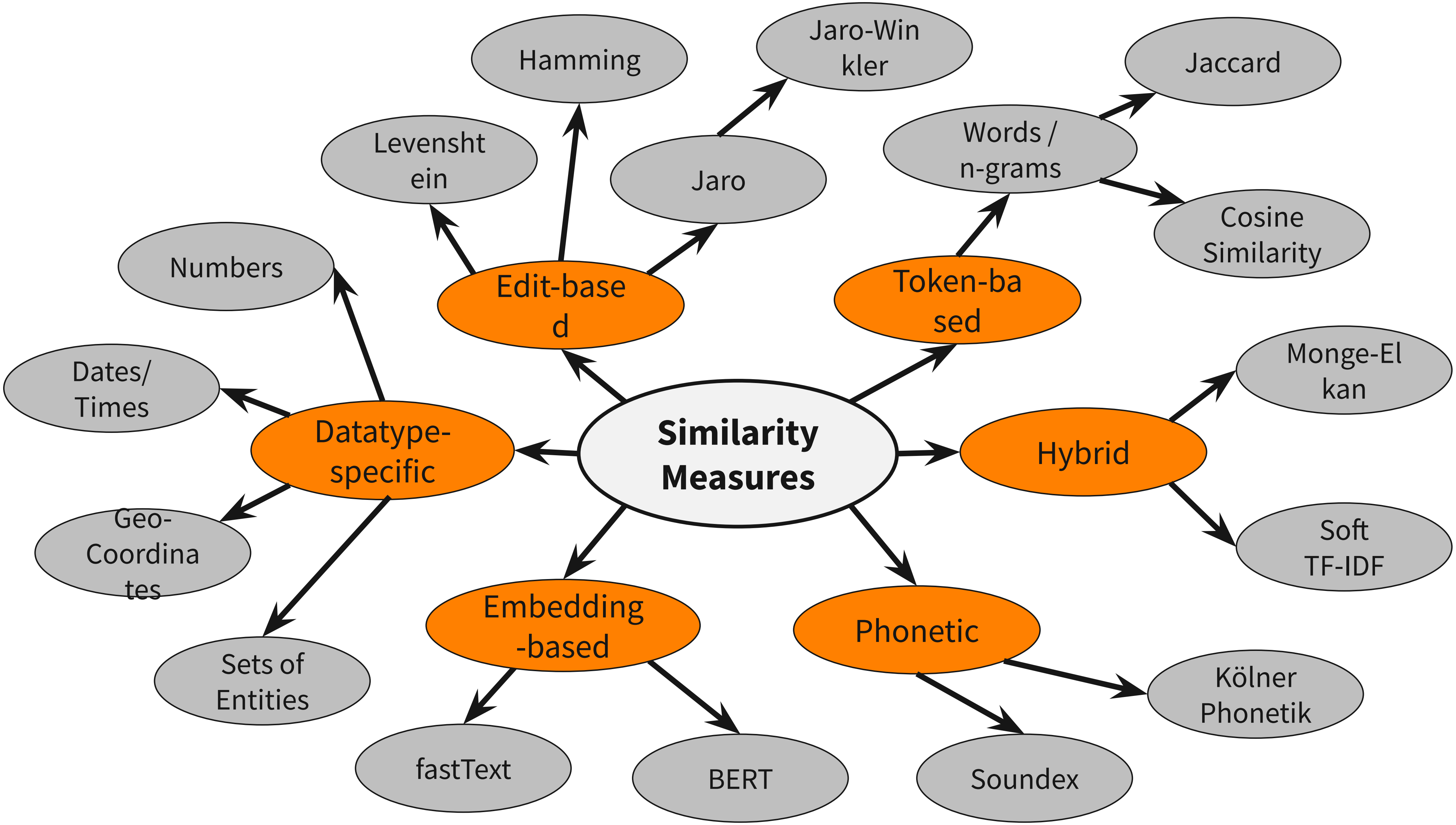


- Köpcke, Thor, Rahm: Evaluation of entity resolution approaches on real-world match problems. VLDB 2010.
- Anna Primpeli, Christian Bizer: Profiling entity matching benchmark tasks. CIKM 2020.

Outline – Identity Resolution Part 2

1. Introduction
2. Entity Matching
3. Blocking
4. Evaluation
5. Similarity Measures – In Detail
6. Learning Matching Rules
7. Combining Schema and Entity Matching

5. Similarity Measures – In Detail



Similarity Measures within the Entity Matching Process

Gather Data for Matching

Normalize Attribute Values

Apply Attribute-specific Similarity Measures

We are here

Combine Similarity Scores

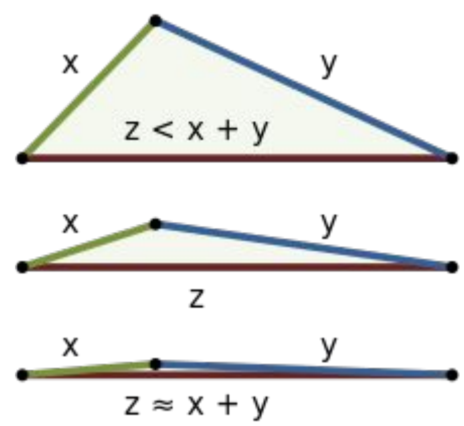
But do not forget the importance of the first two steps!

Decide Match/Non-Match

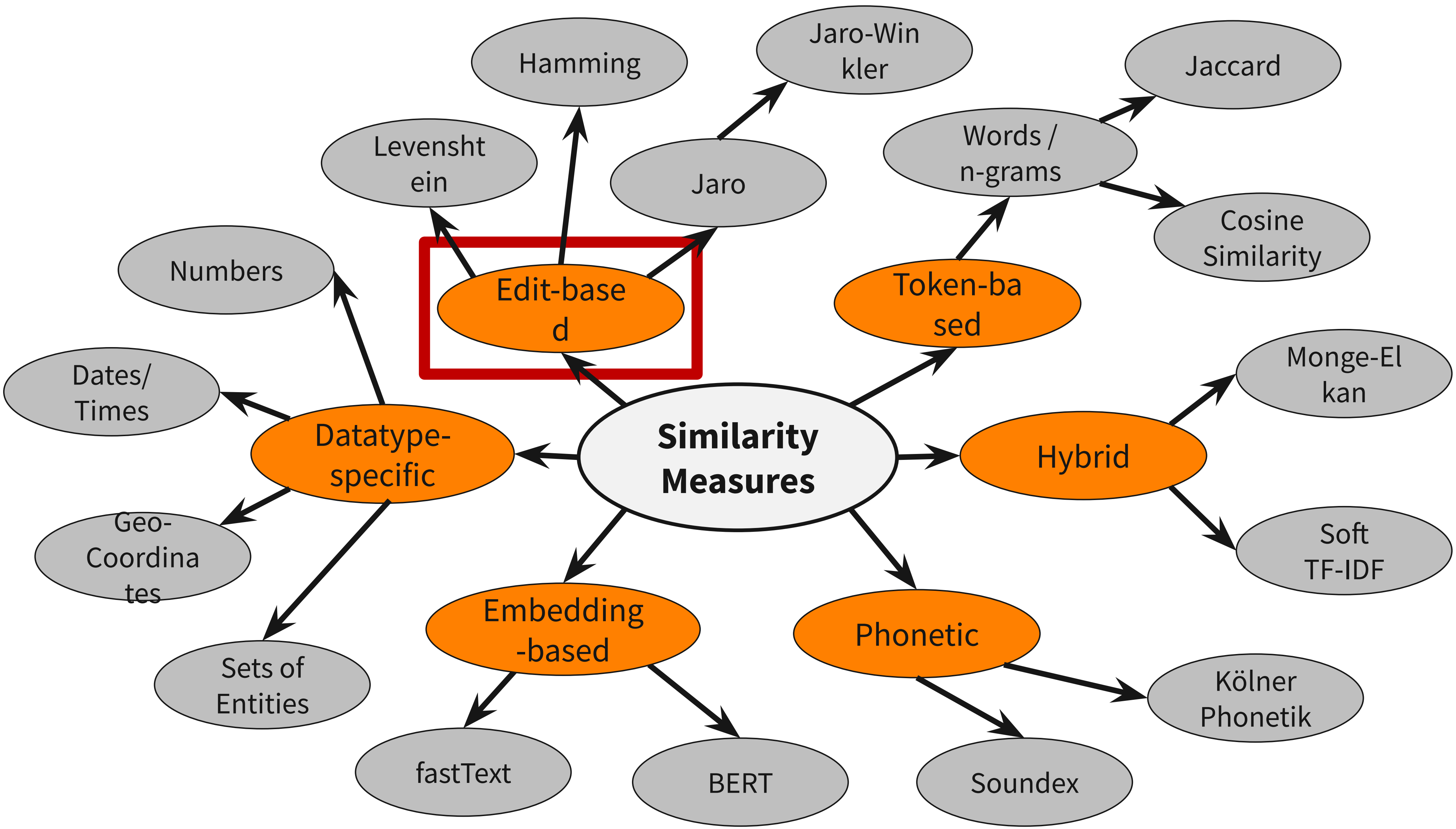
Cluster Records based on Correspondences

Similarity and Distance Measures

- Similarity is a rather universal but vague concept: sim(x,y)
  - x and y can be strings, numbers, geo coordinates, images, songs, persons, …
- Normalized: sim(x,y) ∈ [0,1]
  - sim(x,y) = 1 for exact match
  - sim(x,y) = 0 for „completely different“ x and y
- Distance measures
  - Positive: dist(x,y) ≥ 0
  - Reflexive: dist(x,x) = 0
  - Symmetric: dist(x,y) = dist(y,x)
  - Triangular inequation: dist(x,z) ≤ dist(x,y) + dist(y,z)
- Converting distances to similarities
  - sim(x,y) = 1/(dist(x,y)+1) if dist(x,y) ∈ [0,∞]



5.1 Edit-based String Similarity Measures



Levenshtein Distance (aka Edit Distance)

- Measures the dissimilarity of two strings
- Measures the minimum number of edits needed to transform one string into the other
- Allowed edit operations:
  1. insert a character into the string
  2. delete a character from the string
  3. replace one character with a different character
- Examples:
  - levensthein('table', 'cable') = 1 (1 substitution)
  - levensthein('Chris Bizer', 'Bizer, Chris') = 11 (10 substitution, 1 deletion)
- Levenshtein distance is often called „*edit distance*"
  - as it is the most widely used edit-based measure

Levenshtein Similarity

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

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

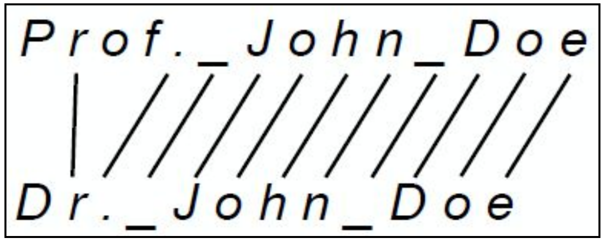
Levenshtein Discussion

- Good general purpose string similarity measure
  - can deal with typos
  - does not work if parts of string (words) have different order
    - ‘Firstname Surname’ vs. ‘Surname, Firstname’
  - other similarity measures are optimized for specific strings like names
- Has quadratic runtime complexity ☹
  - Levenshtein distance is calculated using dynamic programming
  - runtime complexity $O(|x| \cdot |y|)$

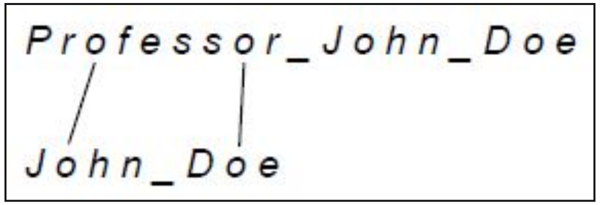
Jaro Similarity

- Specifically designed for matching names at US Census Bureau
- Applies heuristics that empirically proofed to work for names
  - first names, surnames, street names, city names

1. Search for matching characters within a specific distance
   - *m :* number of matching characters
   - search range for matching characters: $\frac{\max(|x|,|y|)}{2} - 1$
   - division by 2 as names often have two parts



2. Look for swapped adjacent characters
   - *transposition: 'pe' vs. 'ep'*
   - *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**

$m = 4 \qquad t = 1$

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

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

$m = 4 \qquad t = 0$

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

Winkler Similarity

- Intuition: Similarity of first few letters is more important
  - less typos in first letters
  - dealing with abbreviations
    - ‘Apple Corp.’ vs. ‘Apple Cooperation’
    - ‘Bizer, Christian’ vs. ‘Bizer, Chris’
- 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 $\geq$10

Jaro-Winkler Similarity

- Extension of Jaro similarity considering a common prefix

$$\text{if } sim_{jaro} \le 0.7 : sim_{jarowinkler} = sim_{jaro}$$

$$\text{otherwise}: \qquad sim_{jarowinkler} = sim_{jaro} + l \cdot p \cdot (1 - sim_{jaro})$$

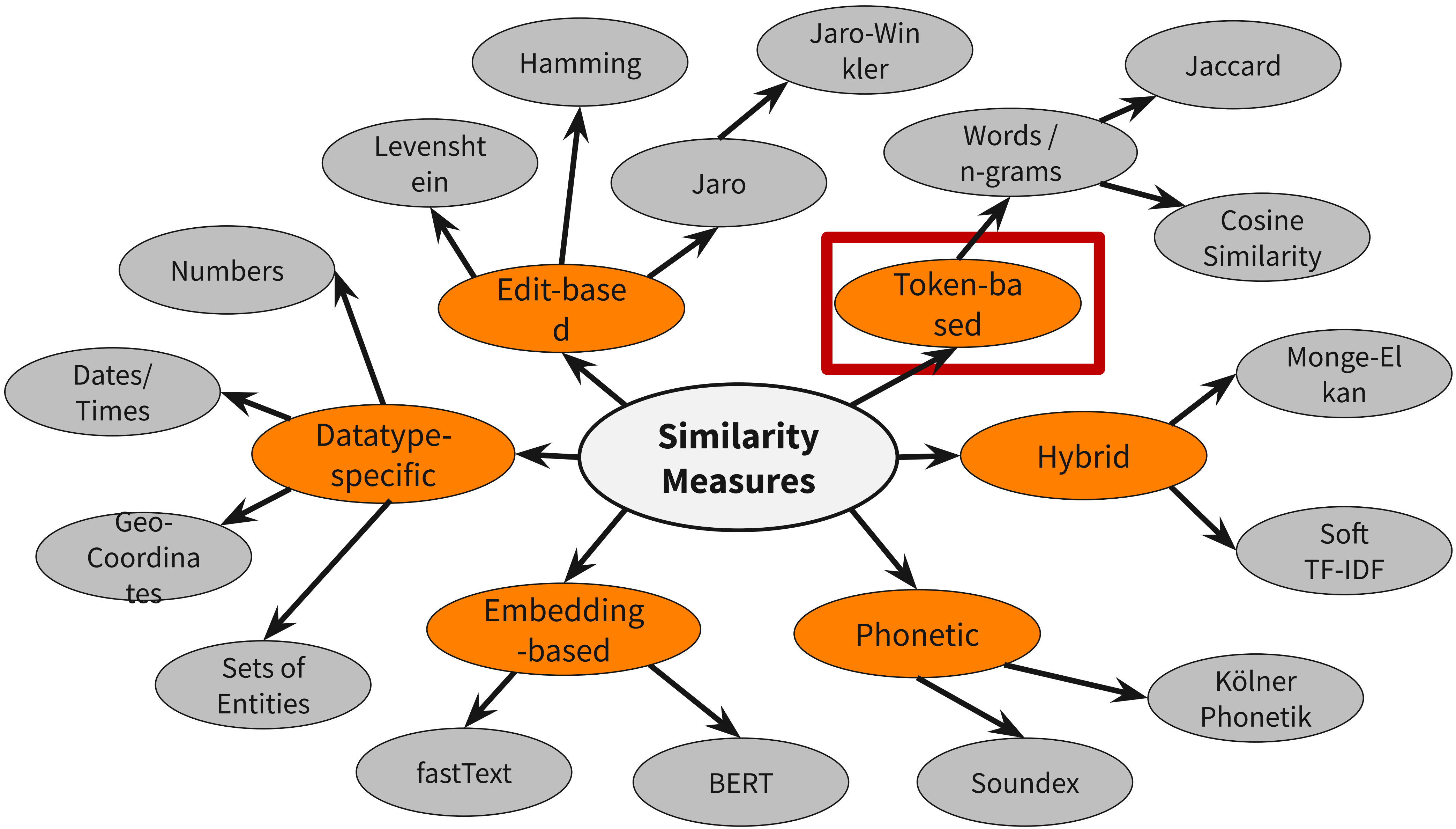
- *l:* Length of common prefix up to a maximum of 4 characters
- *p* : Constant scaling factor for how much the score is adjusted upwards for having common prefixes (typically *p*=0.1)

- Examples:

$$s_1 = PAUL \qquad s_2 = PUAL$$
$$sim_{jaro} = 0.92$$
$$l = 1$$
$$p = 0.1$$
$$sim_{jarowinkler} = 0.92 + 1 \cdot 0.1 \cdot (1 - 0.92) = 0.928$$

$$s_1 = JONES \qquad s_2 = JOHNSON$$
$$sim_{jaro} = 0.79$$
$$l = 2$$
$$p = 0.1$$
$$sim_{jarowinkler} = 0.79 + 2 \cdot 0.1 \cdot (1 - 0.79) = 0.832$$

5.2 Token-based String Similarity Measures



Token-based Similarity

Token-based measures ignore the order of words which is often desirable for comparing multi-word strings.

- ‘Chris Bizer’ and ‘Bizer, Chris’ do not look similar to edit-based measures
- ‘Processor: Intel Xeon E5620’ vs. ‘Intel Xeon E5620 processor’ vs. ‘Intel Xeon E5620’ consist of similar tokens
- Tokenization
  - forming words from sequence of characters
- General idea: Separate string into tokens using some separator
  - possible separators: space, hyphen, punctuation, special characters
- Alternative: Split string into short substrings
  - n-grams: See next slide

n-grams (aka q-grams)

- Split string into short substrings of length *n*
  - by sliding a length n window over the string
  - *n*=2: Bigrams
  - *n*=3: Trigrams
  - Variation: pad with *n* – 1 special characters
    - Emphasizes beginning and end of string
  - Variation: include positional information in order to weight similarities later
- Goals:
  1. Deal with typos and different order of words
  2. Reduce the time complexity compared to Levenshtein

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

- Can be applied to words or n-grams
- Overlap Coefficient: $sim_{overlap}(x,y) = \frac{|tok(x) \cap tok(y)|}{\min(|tok(x)|,|tok(y)|)}$
  - example: useful for attribute label matching if one label might contain additional information, such as units of measurements or years
- Jaccard Coefficient:

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

  - focuses of both strings as all unique tokens are considered
  - widely used general purpose similarity measure for tokens
- Speeding up the calculation using an inverted index, see
  - Doan, Halevy: Principles of Data Integration, Chapter 4.3

Cosine Similarity and TF-IDF

- Rare tokens are often more distinguishing and thus more relevant for determining the similarity of two strings
- TF/IDF weighting gives less weight to common tokens (domain-specific stopwords)

	Samsung	Galaxy	S9	S4	32GB	64GB
p1	0	0	0.04	0	0	0.12
p2	0	0	0.04	0	0.04	0
p3	0	0	0	0.12	0.04	0

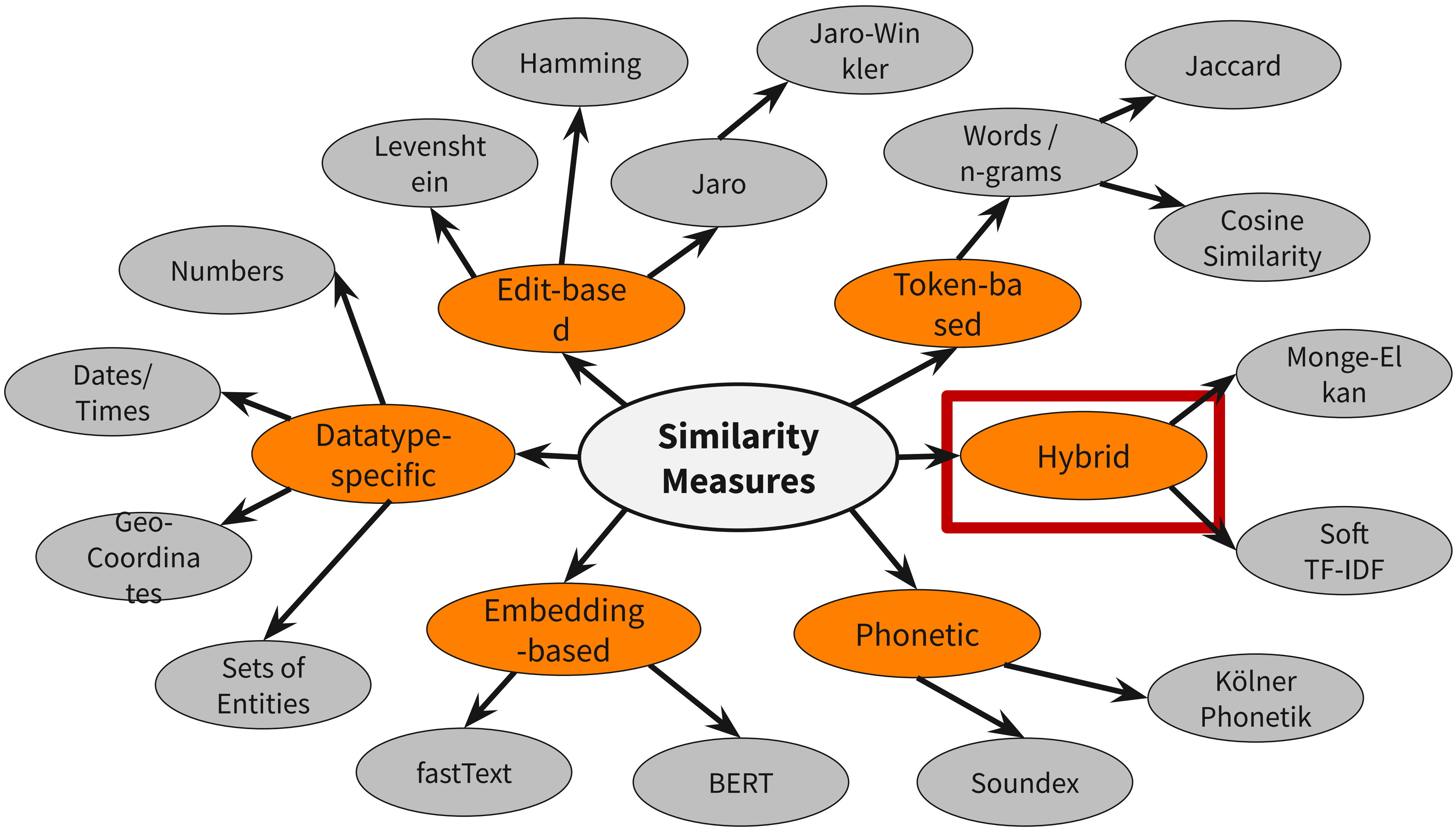
$$w_{ij} = tf_{ij} \times idf_i$$

$$idf_i = \log \frac{N}{df_i}$$

- Cosine similarity
  - popular similarity measure for comparing weighted term vectors

$$\cos(d_1, d_2) = \frac{d_1 \bullet d_2}{\| d_1 \| \| d_2 \|}$$

5.3 Hybrid String Similarity Measures



Monge-Elkan Similarity

– hybrid similarity measures split strings into tokens and apply internal similarity function to compare tokens

– can deal with typos and different order of words

– Monge-Elkan similarity searches for the best match for each token of the first sting x in the second string y

– $$sim_{MongeElkan}(x,y) = \frac{1}{|x|}\sum_{i=1}^{|x|} \max_{j=1,|y|} sim'(x[i],y[j])$$

  - $|x|$ is number of tokens in *x*
  - *sim'* is internal similarity function (e.g. Levenshtein or Jaro depending on the specific requirements of the application)

– focuses on first string x, as length of y does not matter

– runtime 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

- $sim_{MongeElkan}(peter\ christen, christian\ pedro) = \frac{1}{2}(0.7333 + 0.8843) = 0.8088$

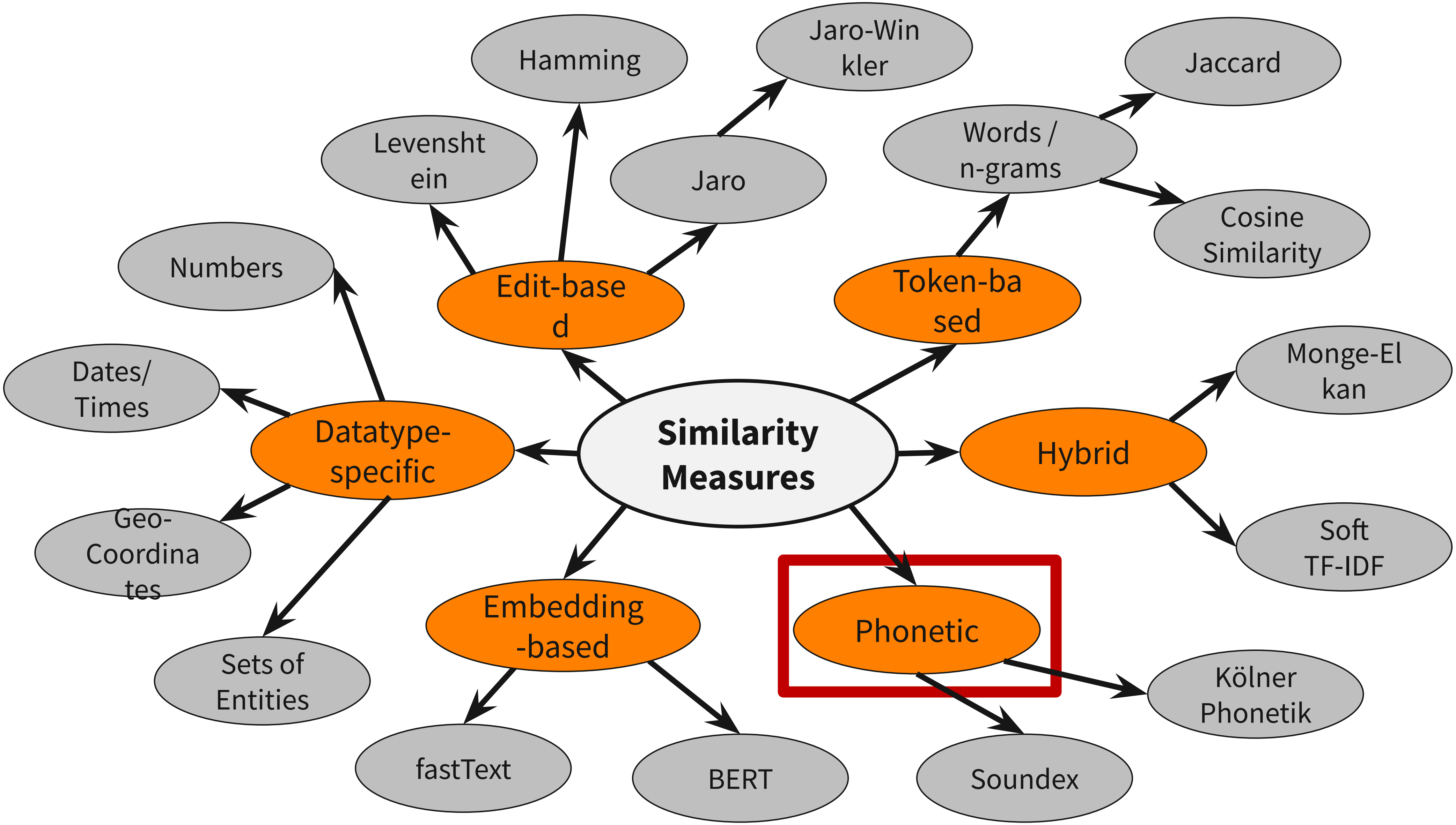
Extended Jaccard Similarity

- uses an internal similarity function (e.g. Levenshtein or Jaro) to calculate similarity between all pairs of tokens
- considers tokens as shared if similarity is above threshold
  - 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: $U_{tok(x)} = \{x_i | x_i \in tok(x) \wedge y_j \in tok(y) \wedge (x_i, y_j) \notin S\}$
- calculates overall similarity as

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

- focuses of both strings as all unique tokens are considered
  - as opposed to Monge-Elkan which focuses on tokens of first string

5.4 Phonetic String Similarity Measures



Soundex

- Soundex codes a last name based on the way a name sounds
- Algorithm:
  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
- Rules have been generated empirically

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

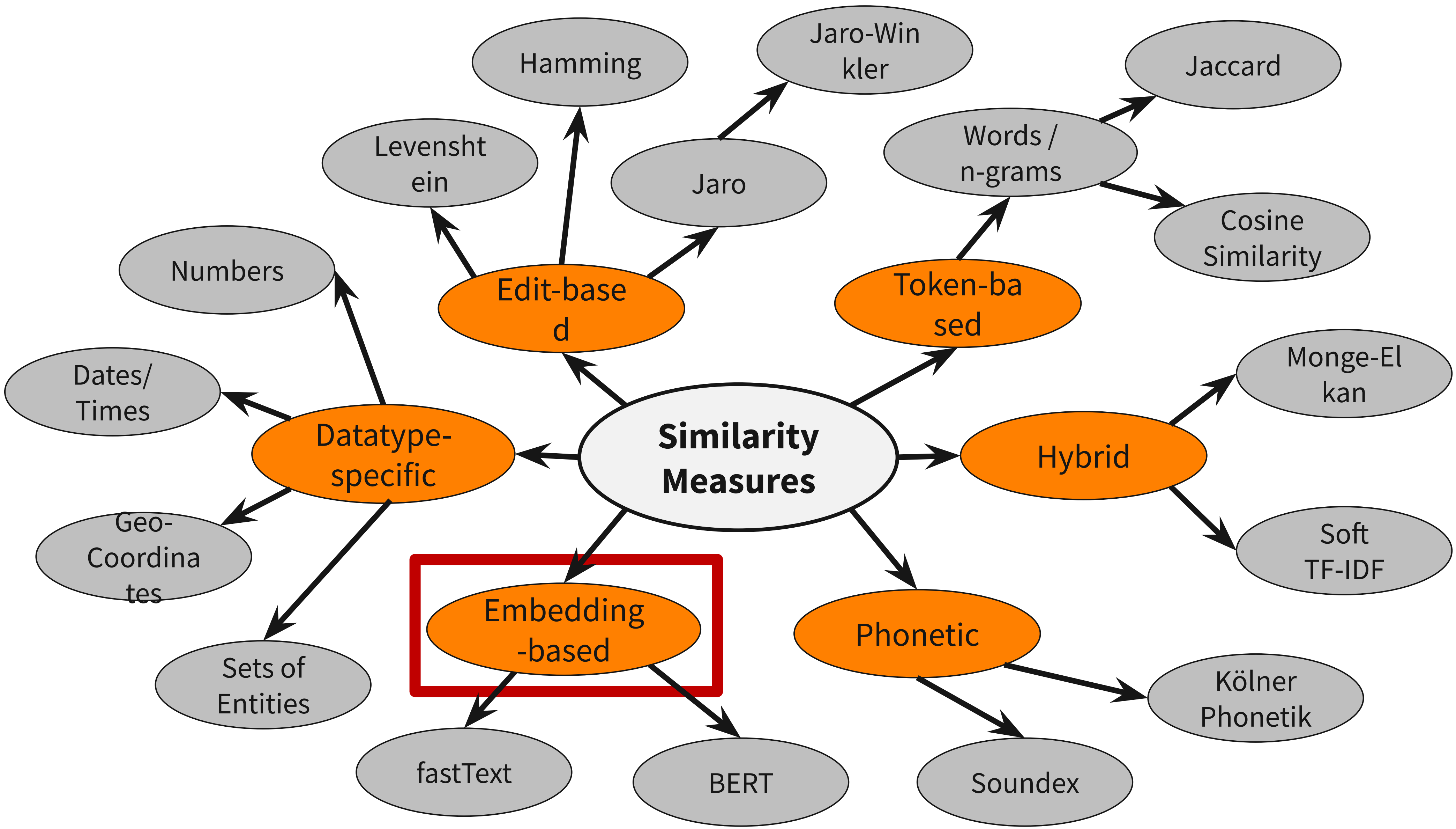
J525 also: Jenkins, Jansen, Jameson

Kölner Phonetik

- Like Soundex, but aimed at 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
- Examples:
  - PAUL: 15
  - PUAL: 15
  - JONES: 68
  - JOHNSON: 686

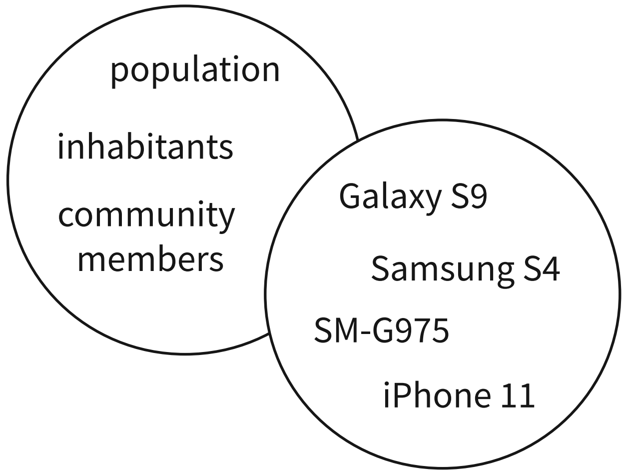
Letter	Context	Code
A, E, I, J, O, U, Y		-
H		-
B		1
P	not before H	
D, T	not before C, S, Z	2
F, V, W		3
P	before H	
G, K, Q		4
C	in the initial sound before A, H, K, L, O, Q, R, U, X	
	before A, H, K, O, Q, U, X but not after S, Z	
X	not after C, K, Q	48
L		5
M, N		6
R		7
S, Z		8
C	after S, Z	
	in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

5.5 Embedding-based String Similarity Measures

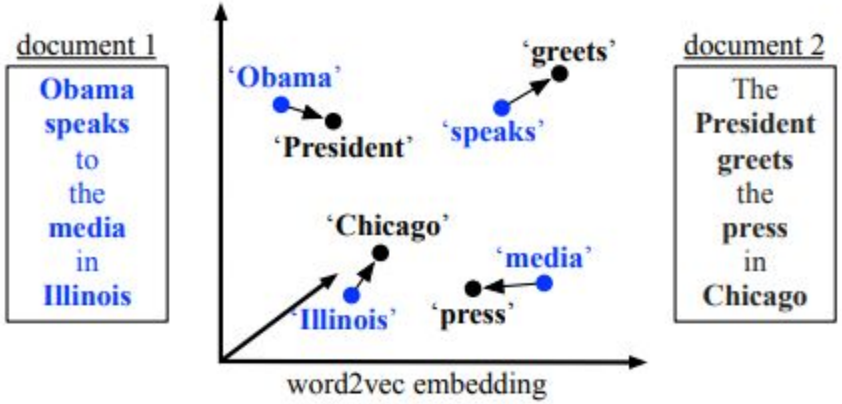


Embedding-based String Similarity

- Embeddings represent words as points in a multidimensional vector space
  - the calculation of embeddings exploits that semantically related words appear in similar contexts in large text corpora (distributional similarity)

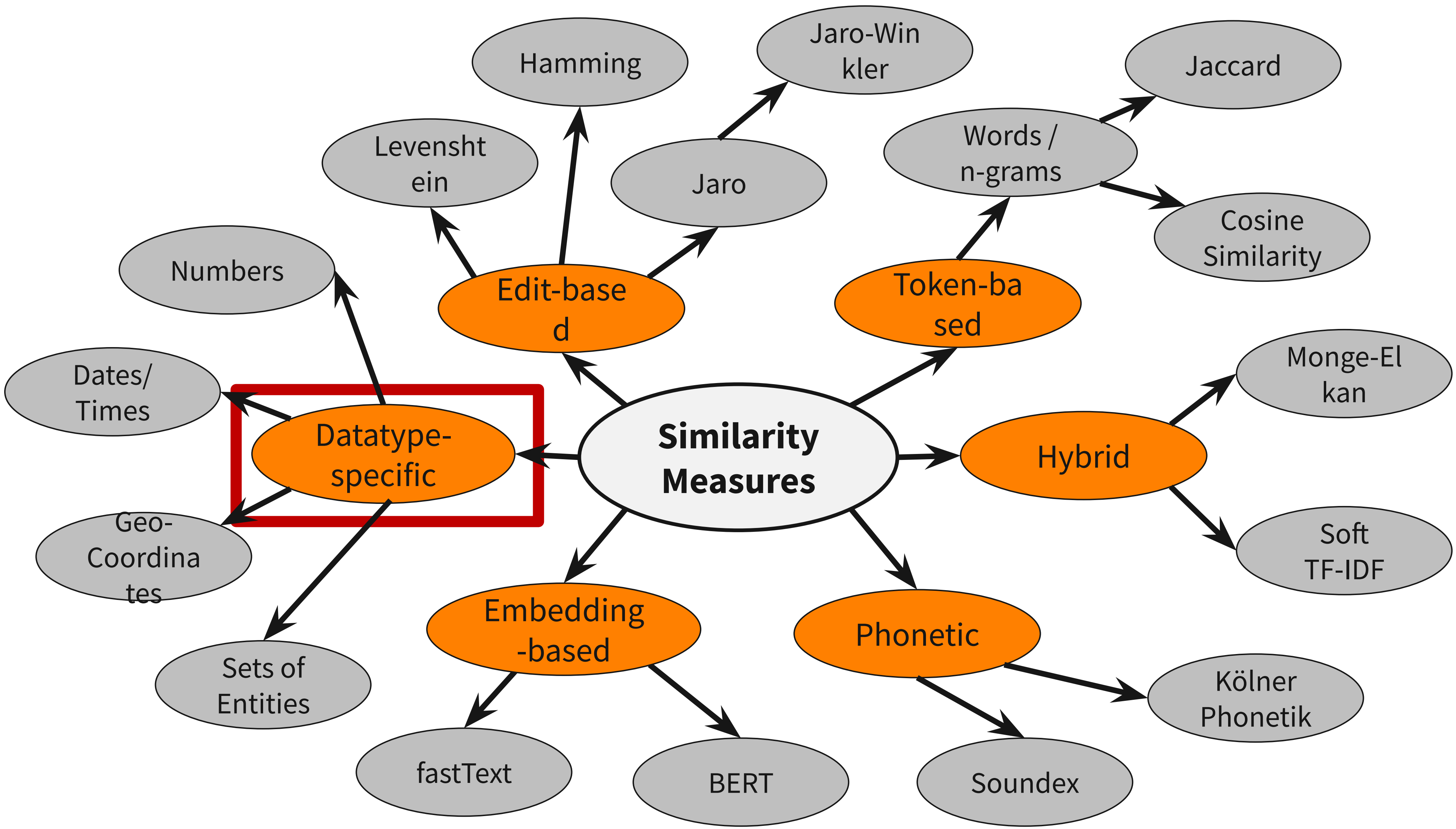


- Similarity of two embeddings
  - cosine similarity, inner product
- Similarity of two sequences of embeddings
  - word movers distance
  - neural networks (LTSMs, BERT, SBERT)
- Embeddings are successfully used for
  - schema matching
  - blocking before entity matching
  - as foundation for supervised entity matching methods



Mudgal, Sidharth: Deep Learning for Entity Matching: A Design Space Exploration. SIGMOD, 2018.

5.6 Data Type Specific Similarity Measures



Numerical Comparison

Approach 1: Tolerate absolute difference between values, independently of absolute values

- $$sim_{num\_abs}(n,m) = \begin{cases} 1 - \left(\frac{|n-m|}{d_{max}}\right) & if\ |n-m| < d_{max} \\ 0 & else \end{cases}$$
  - Linear extrapolation between 0 and d_{max}
  - d_{max} = maximal numeric distance in which numbers should be considered similar
- Example:
  - d_{max} = $1,000
  - $sim_{num\_abs}(2{,}000,\ 2{,}500) = 1 - \frac{500}{1{,}000} = 0.5$
  - $sim_{num\_abs}(200{,}000,\ 200{,}500) = 1 - \frac{500}{1{,}000} = 0.5$

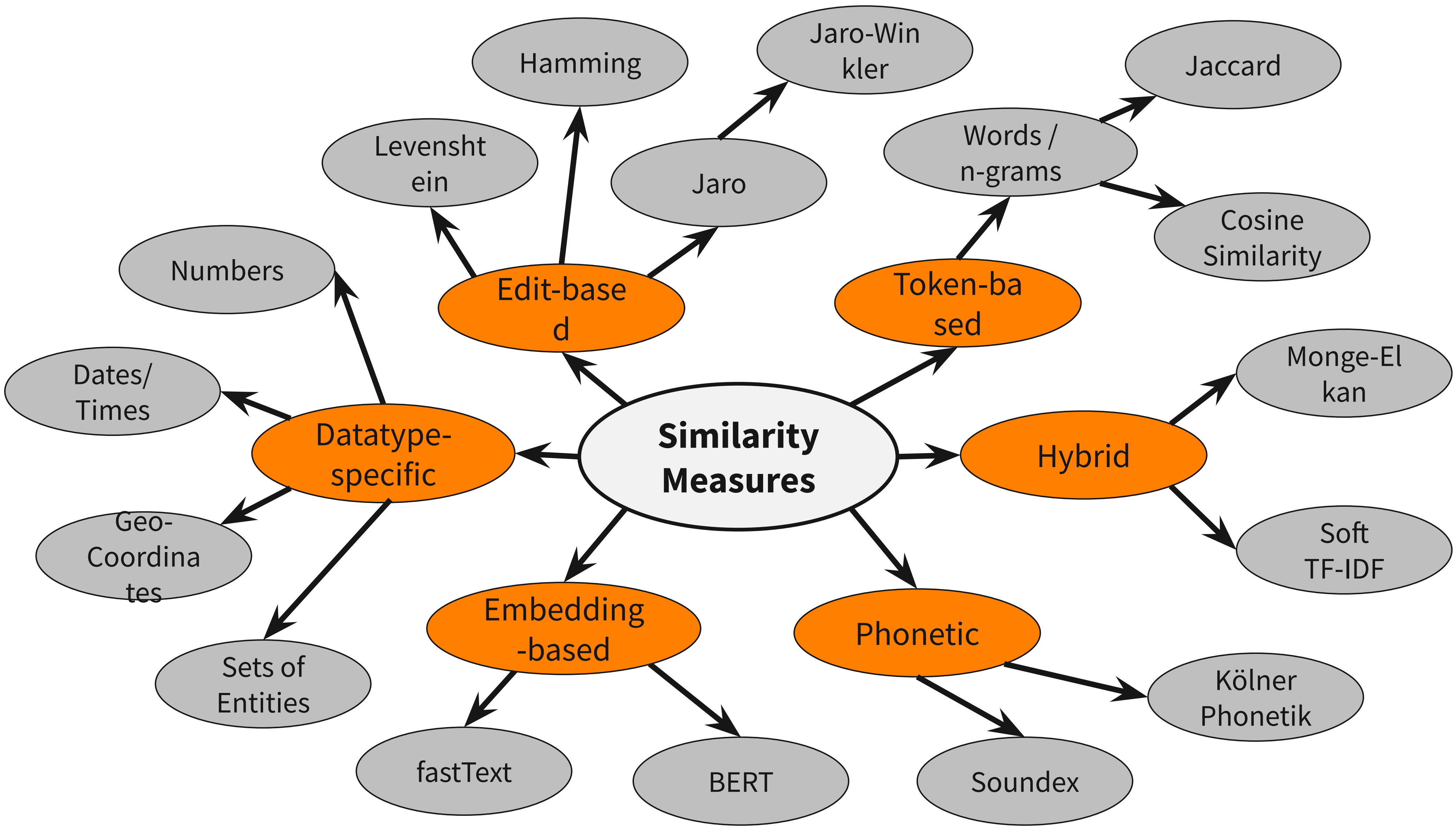
Approach 2: Tolerate difference up to a certain percentage of the absolute values

- $$sim_{num\_perc}(n,m) = \begin{cases} 1 - \left(\frac{pc}{pc_{max}}\right) & if\ pc < pc_{max} \\ 0 & else \end{cases}$$
  - $pc = \frac{|n-m|}{\max(|n|,|m|)} \cdot 100$ is percentage difference
  - pc_{max} = 33% is the maximal percentage that should be tolerated
  - $sim_{num\_perc}(2{,}000,\ 2{,}500) = 1 - \frac{20}{33} = 0.394$ because $pc = \frac{|2{,}000 - 2{,}500|}{2{,}500} \cdot 100 = 20\%$
  - $sim_{num\_perc}(200{,}000,\ 200{,}500) = 1 - \frac{0{,}25}{33} = 0.992$ because $pc = \frac{500}{200{,}500} \cdot 100 = 0.25\%$

Time and Space Comparisons

- Dates
  - convert dates into days after year 0 □ integer
  - afterwards use sim_{num_abs}
- Geographic Coordinates
  - distance is measured along the surface of the Earth in kilometers or miles
  - compute distance based on geographic projection of coordinates
  - Java package for calculating geographic distances: Geographiclib
  - http://geographiclib.sourceforge.net
- More Similarity Measures for other Data Types
  - Tan, Steinbach, Kumar: Introduction to Data Mining. Chapter 4
  - e.g. shopping baskets □ vector of asymmetric binary variables □ Jaccard

Summary



Similarity Measures
Edit-based
Levenshtein
Hamming
Jaro
Jaro-Winkler
Token-based
Words / n-grams
Jaccard
Cosine Similarity
Hybrid
Monge-Elkan
Soft TF-IDF
Phonetic
Kölner Phonetik
Soundex
Embedding-based
fastText
BERT
Datatype-specific
Numbers
Dates/ Times
Geo-Coordinates
Sets of Entities

6. Learning Matching Rules

- Problem
  It is hard for humans to write good matching rules, as this requires a lot of knowledge about the data set and matching techniques
  - What kind of typos and other errors are contained in the data?
  - Which string similarity measure fits which attribute?
  - How to set similarity thresholds?
  - How to weight different attributes?
- Possible solution
  1. Manually label a certain amount of pairs as matches and non-matches
  2. Use machine learning to generate matching rule from this training data
- Advantage
  - The human does what she is good at: Understand the data
  - The computer does what it is good at: Learn detailed rules from examples

Training Data and Feature Generation

- **Training data**: $T = \{(x_1,y_1,l_1), \ldots (x_n,y_n,l_n)\}$, where
  - each (x_i,y_i) is a record pair and
  - l_i is a label: "yes" if x_i matches y_i and "no" otherwise
- **Feature Generation**
  - define a set of features $f_1, \ldots, f_m$, each quantifying one aspect of the domain judged possibly relevant to matching the records
  - feature = similarity measure applied to attribute pair
    - after normalizing both values
  - if you want the learning algorithm to decides which similarity measure fits best for a specific attribute pair, you generate multiple features for the pair
    - Levenshtein(x.name, y.name)
    - Jaro(x.name, y.name)
    - Jaccard(tokens(x.name, y.name))
  - Feature engineering requires domain-knowledge, e.g. for value normalization

Example: Feature Generation

<a_1 = (Mike Williams, (425) 247 4893, Seattle, WA), b_1 = (M. Williams, 247 4893, Redmond, WA), yes>
<a_2 = (Richard Pike, (414) 256 1257, Milwaukee, WI), b_2 = (R. Pike, 256 1237, Milwaukee, WI), yes>
<a_3 = (Jane McCain, (206) 111 4215, Renton, WA), b_3 = (J. M. McCain, 112 5200, Renton, WA), no>

match names | match phones | match cities | match states | check area code against city

$v_1 = \langle [s_1(a_1,b_1), s_2(a_1,b_1), s_3(a_1,b_1), s_4(a_1,b_1), s_5(a_1,b_1), s_6(a_1,b_1)], 1 \rangle$
$v_2 = \langle [s_1(a_2,b_2), s_2(a_2,b_2), s_3(a_2,b_2), s_4(a_2,b_2), s_5(a_2,b_2), s_6(a_2,b_2)], 1 \rangle$
$v_3 = \langle [s_1(a_3,b_3), s_2(a_3,b_3), s_3(a_3,b_3), s_4(a_3,b_3), s_5(a_3,b_3), s_6(a_3,b_3)], 0 \rangle$

Label:
Match = 1
Non-Match = 0

- s_1 and s_2 use Jaro-Winkler <u>and</u> edit distance
- s_3 uses edit distance (ignoring area code of a)
- s_4 and s_5 return 1 if exact match, 0 otherwise
- s_6 encodes a heuristic constraint (using a lookup table)

Learn Matching Model M

1. Convert each training example (x_i, y_i, l_i) in T to a pair (v_i, l_i)
   - v_i = $f_1(x_i,y_i)$, …, $f_m(x_i,y_i)$ is a feature vector that encodes (x_i,y_i) in terms of the features (list of similarity values)
   - thus T is transformed into T' = {(v_1,l_1), …, (v_n,l_n)}

2. Apply a learning algorithm to T' to learn a matching model M
   - linear models: logistic regression, linear regression, SVMs
   - non-linear models: decision tree, random forest, XGBoost, neural net

3. Optimize the hyperparameters of the learning algorithm
   - using training, validation (!), and test set

Example: Learning a Linearly Weighted Matching Rule

- Goal: Learn rule $sim(a,b) = \sum_{i=1}^{6} \alpha_i * s_i(a,b)$
- Perform a least-squares linear regression on training data

  v_1 = <[$s_1(a_1,b_1)$, $s_2(a_1,b_1)$, $s_3(a_1,b_1)$, $s_4(a_1,b_1)$, $s_5(a_1,b_1)$, $s_6(a_1,b_1)$], 1>

  v_2 = <[$s_1(a_2,b_2)$, $s_2(a_2,b_2)$, $s_3(a_2,b_2)$, $s_4(a_2,b_2)$, $s_5(a_2,b_2)$, $s_6(a_2,b_2)$], 1>

  v_3 = <[$s_1(a_3,b_3)$, $s_2(a_3,b_3)$, $s_3(a_3,b_3)$, $s_4(a_3,b_3)$, $s_5(a_3,b_3)$, $s_6(a_3,b_3)$], 0>

  to find weights α_i that minimize the squared error

$$\sum_{i=1}^{3} \left(c_i - \sum_{j=1}^{6} \alpha_j * s_j(v_i)\ \right)^2$$

Example: Learning a Decision Tree

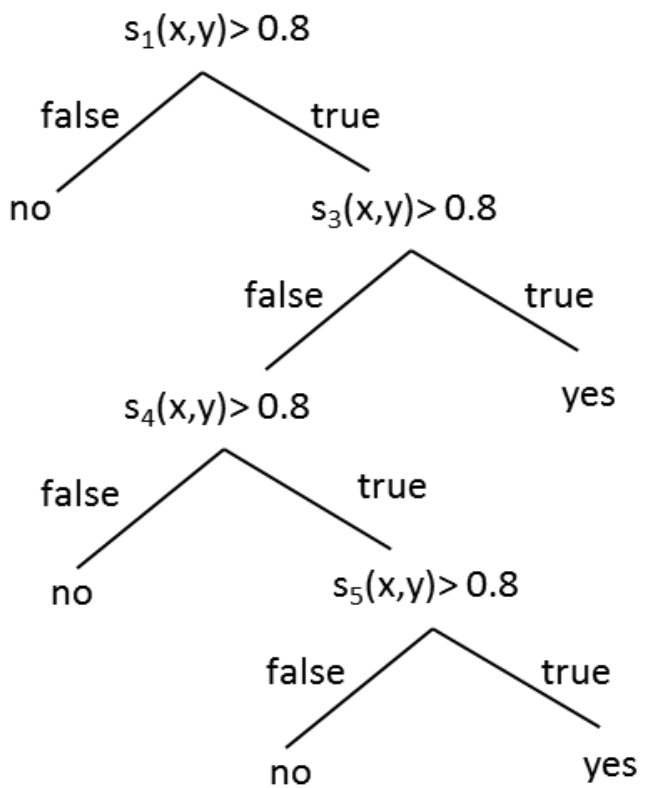
match names   match phones   match cities   match states   check area code against city

v_1 = <[$s_1(a_1,b_1)$, $s_2(a_1,b_1)$, $s_3(a_1,b_1)$, $s_4(a_1,b_1)$, $s_5(a_1,b_1)$, $s_6(a_1,b_1)$], yes>

v_2 = <[$s_1(a_2,b_2)$, $s_2(a_2,b_2)$, $s_3(a_2,b_2)$, $s_4(a_2,b_2)$, $s_5(a_2,b_2)$, $s_6(a_2,b_2)$], yes>

v_3 = <[$s_1(a_3,b_3)$, $s_2(a_3,b_3)$, $s_3(a_3,b_3)$, $s_4(a_3,b_3)$, $s_5(a_3,b_3)$, $s_6(a_3,b_3)$], no>

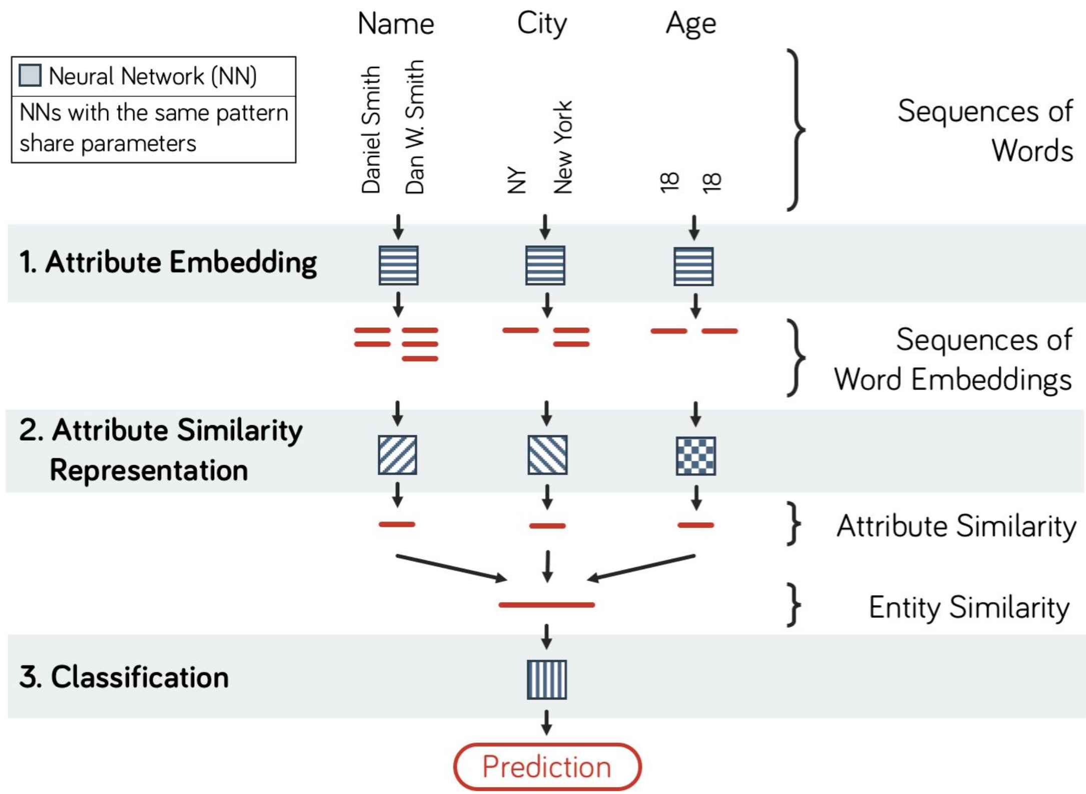
Label:
Match = yes
Non-Match = no



- Tree-based models often perform better than linear models
- The tree learning algorithm automatically selects the most discriminative features
- Always also test random forests and XGBoost

Example: Deep Learning of Matching Models

- Deep learning-based matching models often combine
  1. embeddings for attribute value representation
  2. neural nets for similarity computation, e.g. Siamese networks, and LSTMs, BERT
  3. neural nets for the final matching decision, e.g. fully connected layers on top of concatenated attribute similarity representations

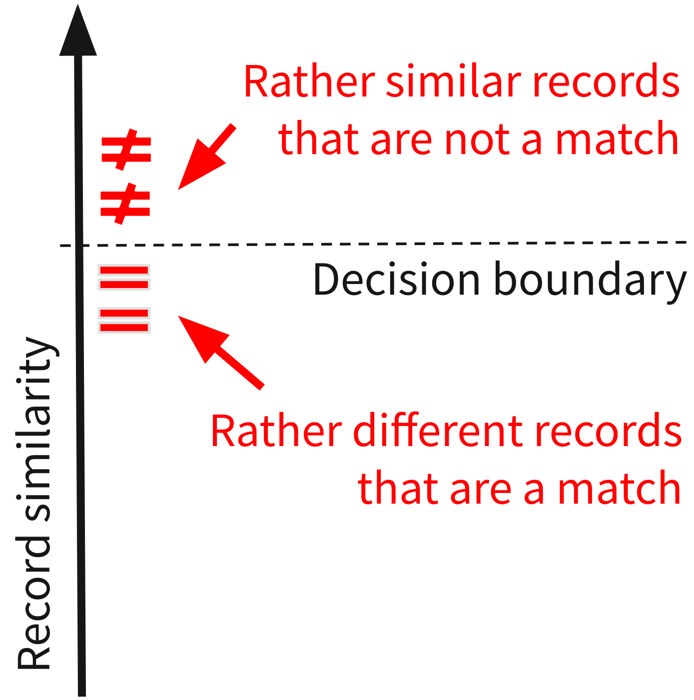


- Often outperform linear and tree-based matching models for less structured textual data given enough training pairs
  - e.g. product titles and descriptions, not numeric sensor data

- Mudgal, Sidharth, et al.: Deep Learning for Entity Matching: A Design Space Exploration. SIGMOD, 2018.
- Yuliang, et al: Deep entity matching with pre-trained language models. VLDB2021.

How to Assemble Good Training Data?

- Training data must
  1. be balanced as random pairs would be highly skewed towards non-matches
  2. contain corner cases as they are most informative
     - especially “near-miss” negative examples are more informative for training than randomly selected pairs which tend to be “easy” non-matches.
     - Star Wars 1 vs. Star Wars 2, Mannheim vs. Ludwigshafen
     - rule of thumb: 50% corner cases



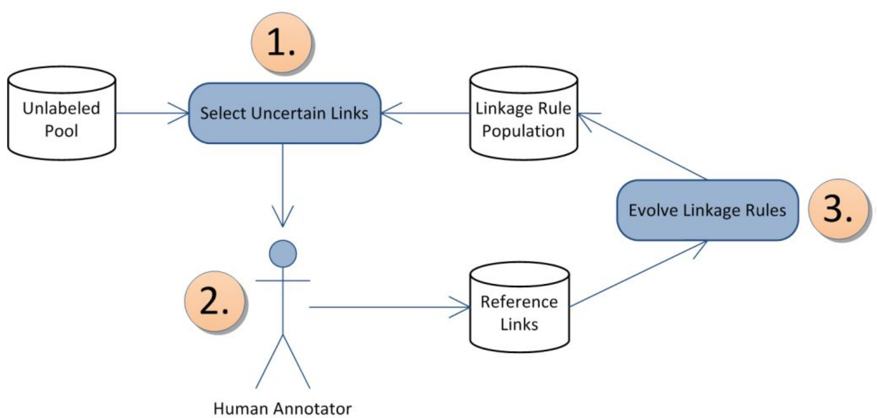
- The more training data the better!
  - remember the learning curve
- Try to reduce labeling effort by
  - reusing existing information about matches e.g. ISBN or GTIN numbers, owl:sameAs

    = weak supervision as quality is often questionable

Hanna Köpcke, Erhard Rahm: Training selection for tuning entity matching. *QDB/MUD*, 2008.
Ratner, et al.: Snorkel: Rapid Training Data Creation with Weak Supervision. VLDB Journal, 2019

Discussion Learning-based Approaches

- Pros compared to writing matching rules by hand
  - when writing rules by hand, you must manually decide if a particular feature is useful □ labor intensive and limits the number of features you can consider
  - learning-based approaches can automatically examine a large number of features
- Cons
  - you need to label training examples
  - you don’t know which examples matter to the algorithm and thus might label an unnecessary large amount of examples in order to cover the relevant corner-cases
- Alternative
  - use Active Learning in order to let the algorithm decide which examples matter
  - practical experience: Often $F_1 > 0.9$ after labeling less than 300 pairs



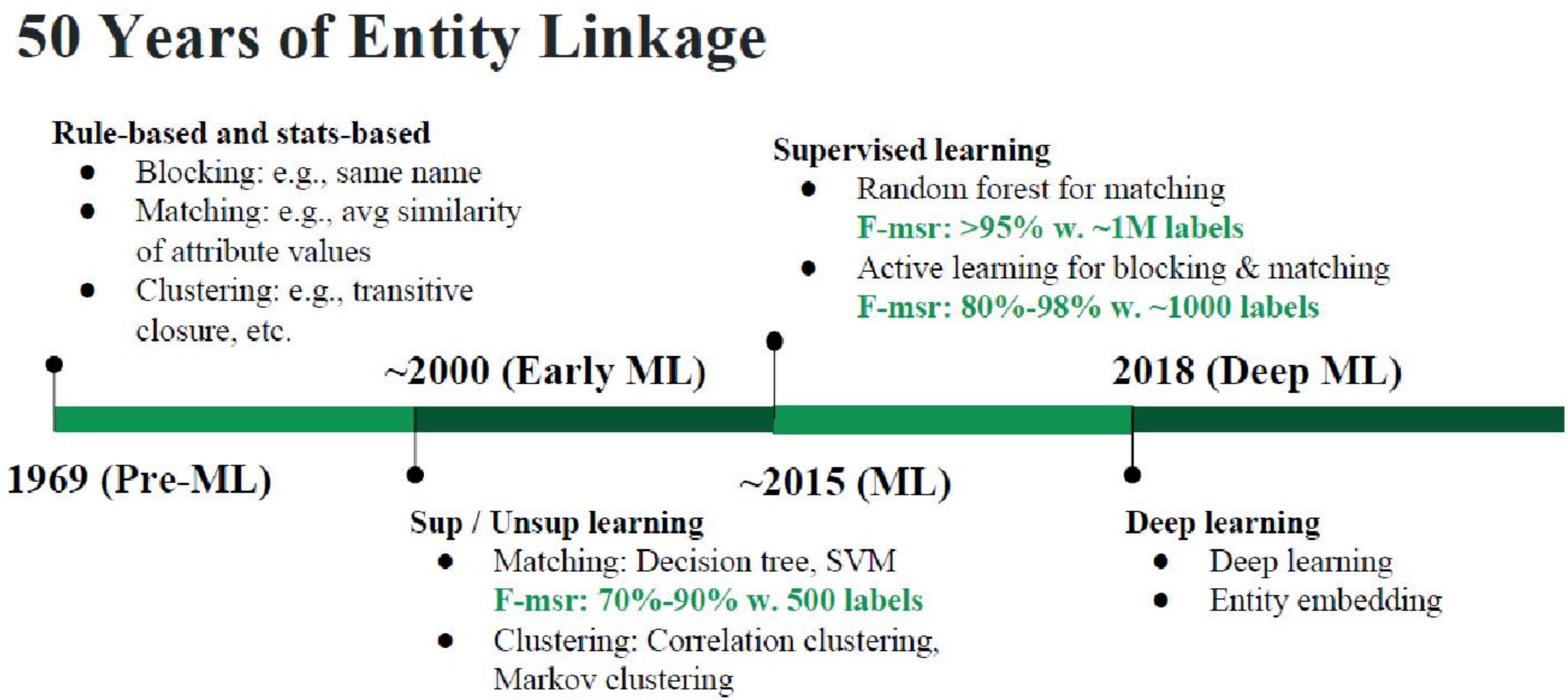
Isele, Bizer: Active Learning of Expressive Linkage Rules using Genetic Programming. Journal of Web Semantics, 2013.

7. Combining Entity and Schema Matching

- Often both entity and schema correspondences are unknown:
  - Matching offers by e-shops to a central product catalog
    - Which product category? Which product? Which product feature?
  - Matching Web tables to a central knowledge base
    - Which ontology class? Which instance? Which property?
- Approach: Combine entity and schema matching in an iterative fashion
  1. Compare entity names to generate candidate entity matches (Star Wars 1-6)
  2. Determine class per table using voting (Class: Movie)
  3. Employ duplicate-based schema matching to align attributes (attributes: name, year, director, producer)
  4. Re-rank entity candidates based on attribute value similarity (matching rule: Similar name and similar year and similar director)
  5. Go back to step 3 until correspondences stabilize

Ritze, Lehmberg, Bizer: Matching HTML Tables to DBpedia. WIMS 2015.
Suchanek, Abiteboul: PARIS - Probabilistic Alignment of Relations, Instances, and Relations. VLDB 2012.

Summary: The Historic Perspective



Dong: ML for Entity Linkage. DI&ML tutorial at SIGMOD 2018.
https://thodrek.github.io/di-ml/sigmod2018/sigmod2018.html

References

- Peter Christen: Data Matching. Springer 2012.
- Christophides, et al: Entity Resolution in the Web of Data. *Morgan & Claypool Publishers*, 2015.
- Naumann, Herschel: An Introduction to Duplicate Detection. *Morgan & Claypool Publishers*, 2010.
- Papadakis, Palpanas: Web-scale, Schema-Agnostic, End-to-End Entity Resolution. Tutorial at WWW2018.
- Barlaug, Gulla: Neural Networks for Entity Matching: A Survey. TKDD, 2021.
- Doan, Halevy, Ives: Principles of Data Integration. Chapter 4 & 7, *Morgan Kaufmann*, 2012.