Approximate Duplicate Detection

Helena Galhardas
IST, University of Lisbon and INESC-ID

Who am I?

- http://web.ist.utl.pt/helena.galhardas/
- Teaching at IST, University of Lisbon
- Researching at INESC-ID

- Areas of Research:
  - Databases
  - Data Cleaning
  - Information Extraction
  - ETL (Extraction, Transformation, Loading)

Context

Data journalism/Fact-checking

Context

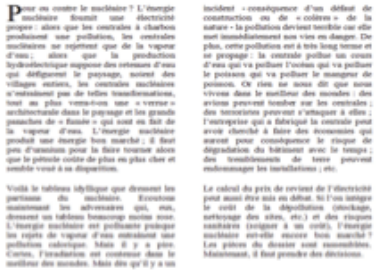
Data journalism/Fact-checking

Numéro	Nom	Prénom	Email
1	Jean	Dupont	jdupont@email.com
2	Marie	Malherbe	mama@email.com
3	Nicolas	Jacques	Jacques.nicolas@email.com
4	Hadrien	Piroux	happi@email.com

Numéro	Client	Produit	Quantité
1	3	Tube de colle	3
2	2	Rame de papier A4	6
3	2	Ciseaux	2

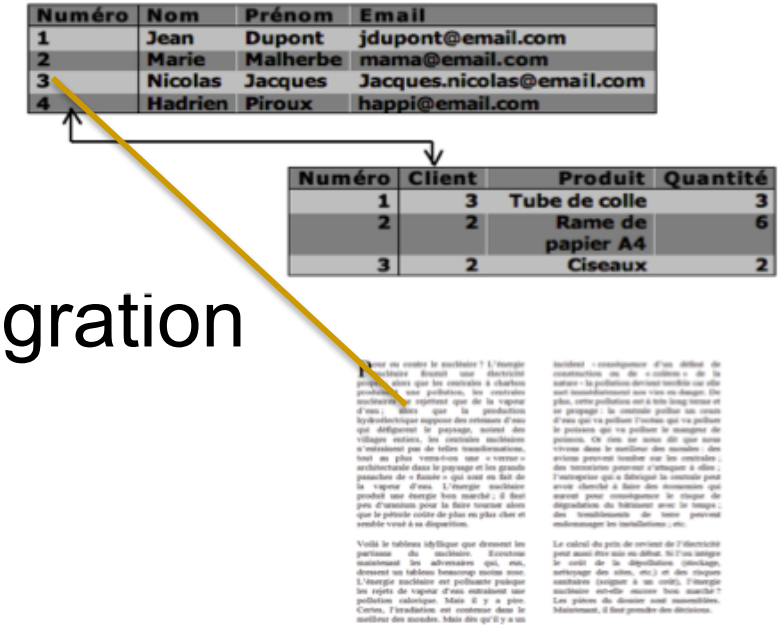
needs

Heterogeneous Data Integration



Context

Data journalism/Fact-checking



Numéro	Nom	Prénom	Email
1	Jean	Dupont	jdupont@email.com
2	Marie	Malherbe	mama@email.com
3	Nicolas	Jacques	Jacques.nicolas@email.com
4	Hadrien	Piroux	happi@email.com

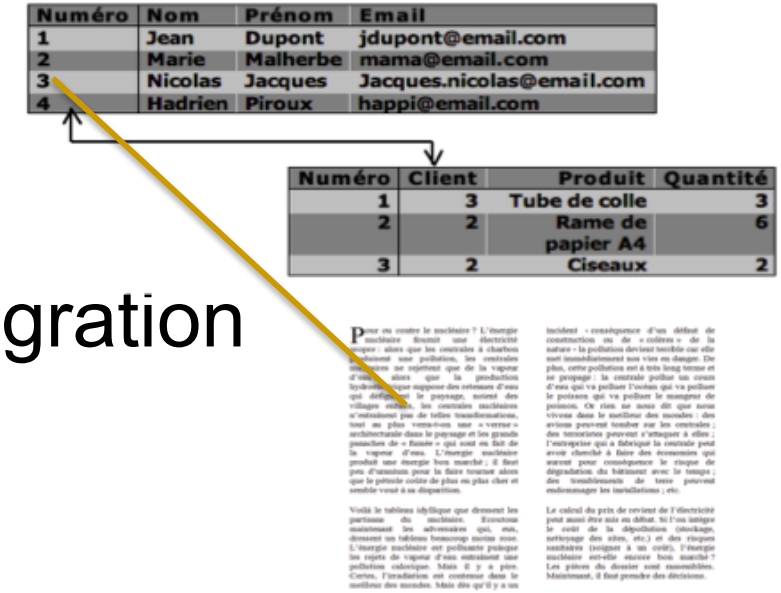
Numéro	Client	Produit	Quantité
1	3	Tube de colle	3
2	2	Rame de papier A4	6
3	2	Ciseaux	2

needs

Heterogeneous Data Integration

Context

Data journalism/Fact-checking



Numéro	Nom	Prénom	Email
1	Jean	Dupont	jdupont@email.com
2	Marie	Malherbe	mama@email.com
3	Nicolas	Jacques	Jacques.nicolas@email.com
4	Hadrien	Piroux	happi@email.com

Numéro	Client	Produit	Quantité
1	3	Tube de colle	3
2	2	Rame de papier A4	6
3	2	Ciseaux	2

needs

Heterogeneous Data Integration

needs

Approximate Duplicate *Detection*

(e.g., PERSON: Anne Martin and person: A. Martin)

Example (relational)

Table R

Name	SSN	Addr
Jack Lemmon	430-871-8294	Maple St
Harrison Ford	292-918-2913	Culver Blvd
Tom Hanks	234-762-1234	Main St
…	…	…

Table S

Name	SSN	Addr
Ton Hanks	234-162-1234	Main Street
Kevin Spacey	-	Frost Blvd
Jack Lemon	430-817-8294	Maple Street
…	…	…

- Find records from different datasets that could be the same entity

The example refers to a data quality problem that is known under different names:

- approximate duplicate detection
- record linkage
- entity resolution
- merge-purge
- data matching …

It is one of the data quality problems addressed by data cleaning

Other Data Quality Problems

incomplete: lacking attribute values, lacking certain attributes of interest, or containing only aggregate data

- e.g., occupation=“”

noisy: containing errors (spelling, phonetic and typing errors, word transpositions, multiple values in a single free-form field) or outliers

- e.g., Salary=“-10”

inconsistent: containing discrepancies in codes or names (synonyms and nicknames, prefix and suffix variations, abbreviations, truncation and initials)

- e.g., Age=“42” Birthday=“03/07/1997”
- e.g., was rating “1,2,3”, now rating “A, B, C”
- e.g., discrepancy between approximate duplicate records

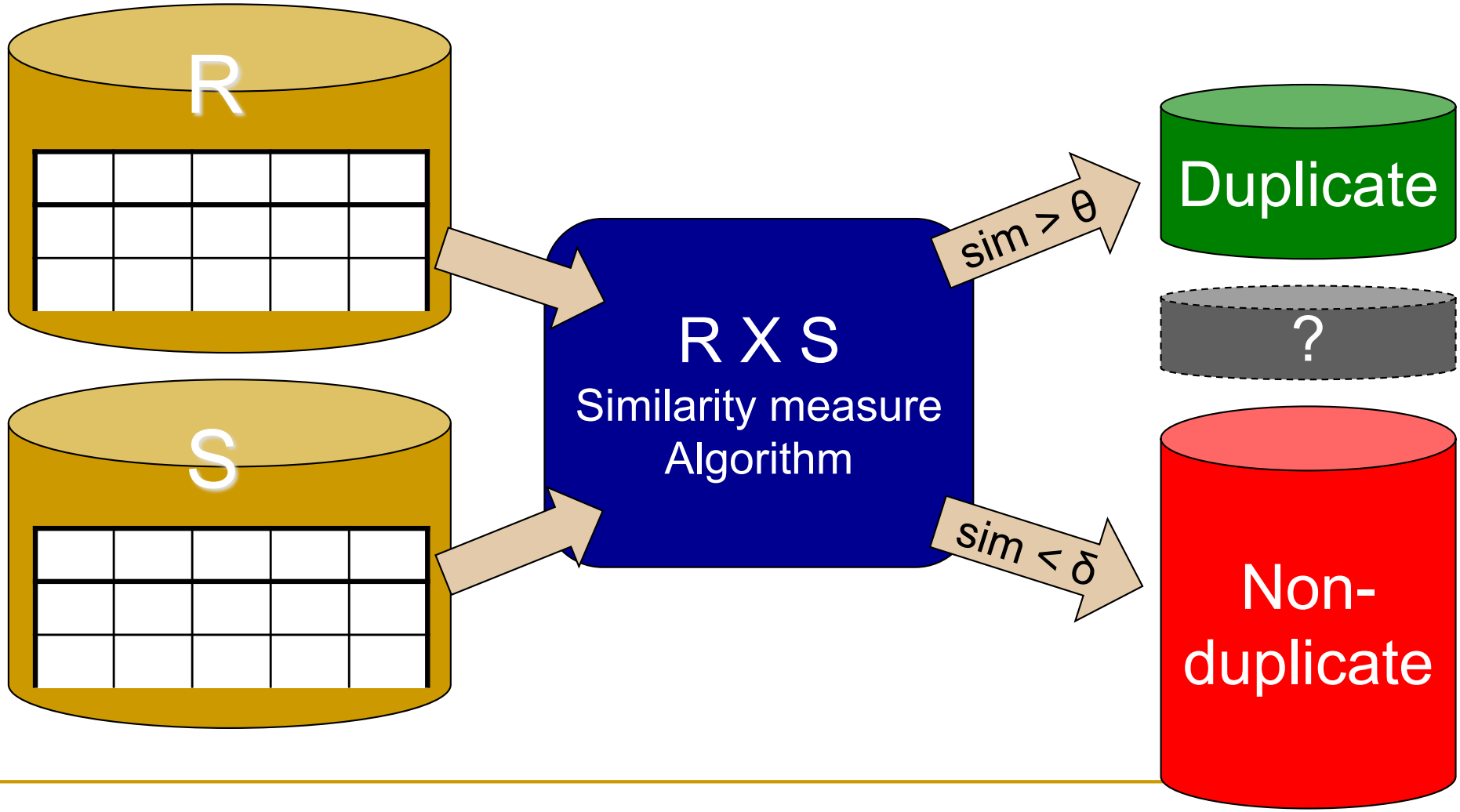
Outline

- **Approximate Duplicate Detection**

App. Duplicate Detection:Problem definition (in relational model)

- Given two relational tables `R` and `S` with identical schema, we say tuple `r` in `R` matches a tuple `s` in `S` if they refer to the same real-world entity
  - Those kind of pairs are called matches
    - We want to find all such matches

App. Duplicate Detection



1st Challenge App. Duplicate Detection

- Match tuples accurately
  - Record-oriented matching: A pair of records with different fields is considered
  - Difficult because matches often appear quite differently, due to typing errors, different formatting conventions, abbreviations, etc
  - Use **string matching algorithms**

2nd Challenge App. Duplicate Detection

- Efficiently match a very large amount (tens of millions) of tuples
  - Record-set oriented matching: A potentially large set (or two sets) of records needs to be compared
  - Aims at minimizing the number of tuple pairs to be compared and perform each of the comparisons efficiently

String matching – what is it?

- Problem of finding strings that refer to the same real-world entity

  Exs:
  - "David R. Smith" and "David Smith"
  - "1210 W. Dayton St, Madison WI" and "1210 West Dayton, Madison WI 53706"
- Formally:
  - Given two sets of strings **X** and **Y**, we want to find all pairs of strings **(x,y)**, where **x** ∈ **X**, **y** ∈ **Y** and such that **x** and **y** refer to the same real-world entity
  - These pairs are denoted matches

1st Challenge String Matching

- Accuracy
  - Strings referring the same entity are often very different (due to typing/OCR errors, different formatting conventions, abbreviations, nicknames, etc)
  - **Solution**: define a similarity measure `s` that takes two strings `x` and `y` and returns a score in [0,1]; `x` and `y` match if `s(x,y) >= t`, being `t` a pre-specified threshold.

2nd Challenge String Matching

- Scalability
  - To apply the similarity metric to a large number of strings
  - Cartesian product of sets X and Y is quadratic in the size of data – impractical!
  - **Solution**: to apply the similarity test only to the most promising pairs

Outline String Matching

- String similarity measures
  - Sequence-based
  - Set-based
  - Hybrid
  - Phonetic
- Scaling up string matching

Sequence-based similarity measures

- View the strings as sequences of characters, and compute the cost of transforming one string into the other
  - **Edit distance**
  - Needleman-Wunch measure
  - Affine Gap measure
  - Smith-Waterman measure
  - **Jaro measure**
  - **Jaro-Winkler measure**

Edit distance

- Levenshtein distance:
  - Minimum number of operations (insertions, deletions or replacements of characters) needed to transform one string into another

Ex: The cost of transforming string “David Smiths” into the string “Davidd Simth” is 4 where the required operations are:

- Inserting character d (after David)
- Substituting m by i
- Substituting i by m
- Deleting the last character of x, which is s

- Given two strings s1 and s2 and their edit distance, denoted by `d(s1,s2)`, the similarity function can be given by:

```
s(s1, s2) = 1 - d(s1, s2)/max(length(s1), length(s2))
```

Ex: The similarity between “David Smiths” and “David Smiths” is 0.67

Computing the edit distance (recurrence equation)

$$d(i,j) = \min \begin{cases} d(i-1,j-1) + c(x_i,y_j) & \text{// copy or substitute} \\ d(i-1,j) + 1 & \text{// delete } x_i \\ d(i,j-1) + 1 & \text{// insert } y_j \end{cases}$$

$c(xi,yj) = 0$ if $x_i = y_j$, 1 otherwise

$d(0,0) = 0$; $d(i,0) = i$; $d(0,j) = j$

Dynamic programming matrix

Computing the distance between “dva” and “dave”

		y0	y1	y2	y3	y4
			d	a	v	e
x0		0	1	2	3	4
x1	d	1	0	1	2	3
x2	v	2	1	1	1	2
x3	a	3	2	1	2	**2**

x=d – v a

y=d a v e

d(x,y) = **2**

Cost of computing `d(x,y)` is `length(x) * length(y)`

Example

- Surtday
- Saturday

		S	A	T	U	R	D	A	Y
	0	1	2	3	4	5	6	7	8
S	1	**0**	**1**	**2**	3	4	5	6	7
U	2	1	1	2	**2**	3	4	5	6
R	3	2	2	2	3	**2**	3	4	5
T	4	3	3	2	3	**3**	3	4	5
D	5	4	4	3	3	4	**3**	4	5
A	6	5	4	4	4	4	4	**3**	4
Y	7	6	5	5	5	5	5	4	3

- Edit distance = 3
- Optimal alignment:

```
S A T U R - D A Y
| | | | | | | | |
S - - U R T D A Y
```

Jaro measure

- Developed mainly to compare short strings, such as first and last names
- Given two strings `x` and `y`,
  - Find the common chars x_i and y_i such that

    $x_i = y_j$ and $|i - j| <= \min\{|x|,|y|\}/2$
    - Common chars: Those that are identical and are positionally "close to one another"
    - The number of common chars x_i in x and y_i in y is the same – it is called **c**
  - Compare the i-th comon character of x with the i-th common character of y. If they don't match, then there is a transposition. Number of transpositions is **t**
  - Compute the Jaro score as:

    $$jaro(x,y) = 1/3 \; [\; c/|x| + c/|y| + (c - t/2)/c \;]$$
- Cost of computation: O(|x||y|)

Example

- x = jon; y = john
  - c = 3
  - Common character sequence in x and y is: jon
  - Nb. transpositions, t = 0
  - Jaro(x,y) = 1/3 (3/3 + ¾ + 3/3) = 0.917
  - Similarity according to edit distance (x,y) = 0.75
- x = jon; y = ojhn
  - C = 3
  - Common character sequence in x: jon
  - Common character sequence in y: ojn
  - t = 2
  - Jaro(x,y) = 1/3 (3/3 + ¾ + (3-2/2)/3) = 0.81

Jaro-Winkler measure

- Modifies the Jaro measure by adding more weight to a common prefix
- Introduces two parameters:
  - PL: length of the longest common prefix between the two strings
  - PW: weight to give the prefix

Jaro-Winkler(x,y) = (1 – PL*PW) * jaro(x,y) + PL*PW

Set-based similarity measures

- View the strings as sets or multi-sets of tokens, and use set-related properties to compute similarity scores
  - Overlap measure
  - Jaccard measure
  - TF/IDF measure
- Several ways of generating tokens from strings
  - Words in the string (delimited by space char)
    - Tokens of “david smith”: {david, smith}
  - Q-grams: substrings of length q that are present in the string
    - 3-grams of “david”: {#da, dav, avi, vid, id#}

Overlap measure

- Let B_x and B_y be the sets of tokens generated for strings x and y
  - Overlap measure: returns the number of common tokens

$$O(x,y) = |Bx \cap By|$$

Ex: x = dave; y = dav

- Set of all 2-grams of x: Bx = {#d, da, av, ve, e#}
- Set of all 2-grams of y: By = {#d, da, av, v#}
- O(x,y) = 3

Jaccard measure

- Jaccard similarity score between two strings x and y is:

  $J(x,y) = |Bx \cap By| / |Bx \cup By|$

Ex: x = “dave” with B_x = {#d, da, av, ve, e#}

y = “dav” with B_y = {#d, da, av, v#}

$J(x,y) = 3/6$

TF/IDF measure

Intuition: two strings are similar if they contain common distinguishing terms

Ex: x = “Apple Corporation, CA”

y = “IBM Corporation, CA”

z = “Apple Corp.”

- Edit distance and Jaccard measure would match x and y
- TF/IDF is able to recognize that “Apple” is a distinguishing term, whereas “Corporation” and “CA” are not

Definitions

- Each string is converted into a bag of terms (a document in IR terminology)

  Ex: x = aab; y=ac; z=a

  string x is converted into document Bx={a, a, b}

- For every term t and document d, compute:
  - Term frequency, **`tf(t, d)`**: number of times t occurs in d
    - `tf(a, x) = 2`
  - Inverse document frequency, **`idf(t)`**: total number of documents in the collection divided by the number of documents that contain t
    - `idf(a) = 3/3`

More definitions

- Each document d is represented into a feature vector v_d
  - Vector v_d has a feature $v_d(t)$ for each term t, and the value of $v_d(t)$ is a function of the TF and IDF scores
  - Vector v_d has as many features as the number of terms in the collection
- Two documents are similar if their corresponding vectors are close to each other

Example

x=aab; y=ac; z=a

B_x={a, a, b}; B_y={a, c}; B_z={a}

tf(a, x) = 2; tf(b, x) = 1; … tf(c, z) = 0

idf(a) = 3/3 = 1; idf(b) = 3/1 = 3; idf(c)=3/1

	a	b	c
vx	2	3	0
vy	3	0	3
vz	3	0	0

with $v_d(t)$ = tf(t,d).idf(t)

Computing the TF/IDF similarity score

- Given two strings p and q
- Let T be the set of all terms in the collection
- Vectors v_p and v_q can be viewed as vectors in the |T|-dimensional space, where each dimension corresponds to a term

TF/IDF score between p and q is the cosine of the angle between these two vectors:

$$S(p,q) = \Sigma_{t \in T} v_p(t) . v_q(t) / \sqrt{(\Sigma_{t \in T} v_p(t)^2)} . \sqrt{(\Sigma_{t \in T} v_q(t)^2)}$$

Ex: $s(x,y) = 2*3/(\sqrt{(2^2+3^2)}*\sqrt{(3^2+3^2)})$

Observations

- TF/IDF similarity score between two strings is high if they share many frequent terms (terms with high TF scores), unless these terms also commonly appear in other strings in the collection (terms with low IDF scores)
- An alternative score computation for dampening the TF and IDF components by a log factor is:

  $$v_d(t) = \log(\, tf(t,d) + 1).\log(idf(t))$$

  - With $v_d(t)$ normalized to length 1:
    - $V_d(t) = v_d(t)/\sqrt{(\Sigma_{t \in T} v_d(t)^2)}$

Hybrid similarity measures

- Combine the benefits of sequence-based and set-based methods
  - Generalized Jaccard measure
    - Enables approximate matching between tokens
  - Soft TF/IDF similarity measure
    - Similar to Generalized Jaccard measure, but using TF/IDF
  - Monge-Elkan similarity measure
    - Breaks both strings into sub-strings and then applies a similarity function to each pair of sub-strings

Phonetic similarity measures

- Match strings based on their sound
  - Specially effective in matching names (e.g., “Meyer” and “Meier”), which are often spelled in different ways but sound the same
- Most commonly used similarity measure: soundex
  - Maps a surname x into a four-character code that captures the sound of the name
  - Two surnames are considered similar if they **share** the same code

Mapping a surname into a code (1)

Ex: `x = Ashcraft`

1. Keep the first letter of `x` as the first letter of the code

   Ex: First letter of `x` is A

2. Remove all occurrences of W and H. Go over the remaining letters and replace them with digits as follows:
   - Replace B, F, P, V with 1
   - Replace C, G, J, K, Q, S, X, Z with 2
   - Replace D, T with 3
   - Replace L with 4
   - Replace M, N with 5
   - Replace R with 6

Mapping a surname into a code (2)

- Do not replace the vowels A, E, I, O, U, and Y
- Ex: Ashcraft -> A226a13

3. Replace each sequence of identical digits by the digit itself
   - A226a13 -> A26a13
4. Drop all the non-digit letters, except the first one. Return the first four letters as the soundex code
   - A26a13 -> A261

Observations about the soundex code

- Is always a letter followed by three digits, padded by 0 if there are not enough digits

  Ex: soundex of Sue is S000
- Hashes similar sounding consonants (such as B, F, P, and V) into the same digit, which means it maps similar sounding names into the same soundex code

  Ex: Both Robert and Rupert map into R163
- Is not perfect

  Ex: fails to map Gough and Goff into the same code
- Widely used to match names in census records, vital records, geneology databases
  - Works well for names from different origins
  - Doesn't work well for Asian names, because the discriminating power of these names is based on vowels that are ignored by the code

A better phonetic similarity measure

- Metaphone
  - A string is converted into a code with variable size
  - It takes into account English pronouncing rules
- Improved versions of the algorithm:
  - Double metaphone
  - Metaphone 3

Outline String Matching

- ✓ Similarity measures
  - ✓ Sequence-based
  - ✓ Set-based
  - ✓ Hybrid
  - ✓ Phonetic
- ➢ **Scaling up string matching**
  - ➢ **Inverted index over strings**
  - ➢ **Size filtering**
  - ➢ Prefix filtering

Recap. Challenge (2)

- Scalability
  - To apply the similarity metric to a large number of strings
  - Cartesian product of sets X and Y is quadratic in the size of data – impractical!
  - Solution: to apply the similarity test only to the most promising pairs.

Naïve matching solution

```
for each string x ∈ X
  for each string y ∈ Y
     if s(x,y) >= t, return (x,y) as a
          matched pair
```

Computational cost: O(|X||Y|) is impractical for large data sets

Solution: Blocking

- To develop a method `FindCands` to quickly find the string that may match a given string x
- Then, use the following algorithm:

```
for each string x ∈ X
    use a method FindCands to find a
    candidate set Z ⊇ Y
    for each string y ∈ Z
        if s(x,y)>= t, return(x,y)
    as a matched pair
```

- Takes O(|X||Z|) time, much faster than O(|X||Y|), because |Z| is much smaller than |Y| and finding |Z| is inexpensive
- Set Z should contain all true positives and as few negative positives as possible

Techniques used in `FindCands`

- **Inverted indexes over strings**
- **Size filtering**
- Prefix filtering
- Position filtering

  - Explained using the Jaccard and Overlap measures

Inverted index over strings

1. Converts each string $y \in Y$ into a document $D(y)$, then builds an inverted index I_y over these documents
2. Given a term t, use I_y to quickly find the list of documents created from Y that contain t, which gives the strings $y \in Y$ that contain t

Example

Two sets of strings X and Y to be matched:

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}

Inverted index I_y

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

Given a string `x = {lake, mendota}` `FindCands` uses I_y to find and merge the ID lists for `lake` and `mendota` and obtain `Z = {4, 6}`

Limitations

- Inverted list of some terms (e.g., stopwords) can be very long
  - Building and manipulating such lists is quite costly
- Requires enumerating all pairs of strings that share at least one term
  - The set of such pairs can still be very large

Size filtering

- Retrieves only the strings in Y whose size makes them match candidates
  - Given a string $x \in X$, infer a constraint on the size of strings in Y that can possibly match x
  - The filter uses a B-tree index to retrieve only the strings that fit the size constraints

Derivation of constraints on the size of strings

$J(x,y) = |x \cap y|/|x \cup y|$

$1/J(x,y) >= |y|/|x| >= J(x,y)$ (can be proved)

If x and y match, then

$J(x,y) >= t$

So:

$1/t >= |y|/|x| >= t \Leftrightarrow$

$|x|/t >= |y| >= |x|.t$

Given a string $x \in X$, only the strings that satisfy this Equation can possibly match x

Example

`x = {lake, mendota}`

`t = 0.8`

Using the equation: **`|x|/t >= |y| >= |x|.t`**

If `y ∈ Y` matches `x`, we must have:

`2/0.8 = 2.5 >= |y| >= 2*0.8 = 1.6`

So, none of the strings in the set Y satisfies this constraint!

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

B-tree index

- Procedure FindCands builds a B-tree over the sizes of strings in `Y`
- Given a string `x` ∈ `X`, it uses the index to find strings in `Y` that satisfy equation:

  `|x|/t >= |y| >= |x|.t`
- Returns that set of strings that fit the size constraint
  - Effective when there is significant variability in the number of tokens in the strings `X` and `Y`

Outline

- **Approximate Duplicate Detection**
  - String Matching (accuracy and efficiency)
  - **Record-oriented matching**
- Approximate Duplicate Elimination (Data Fusion)

Example (recap)

Table R

Name	SSN	Addr
Jack Lemmon	430-871-8294	Maple St
Harrison Ford	292-918-2913	Culver Blvd
Tom Hanks	234-762-1234	Main St
…	…	…

Table S

Name	SSN	Addr
Ton Hanks	234-162-1234	Main Street
Kevin Spacey	-	Frost Blvd
Jack Lemon	430-817-8294	Maple Street
…	…	…

- Find records from different datasets that could be the same entity

1st Challenge

- Match tuples accurately
  - Record-oriented matching: A pair of records with different fields is considered

Record-oriented matching techniques

- Treat each tuple as a string and apply string matching algorithms
- Exploit the structured nature of data – hand-crafted matching rules
- Automatically discover matching rules from training data – supervised learning
- Iteratively assign tuples to clusters, no need of training data – clustering
- Model the matching domain with a probability distribution and reason with the distribution to take matching decisions – probabilistic approaches
- Exploit correlations among tuple pairs to match them all at once – collective matching

Record-oriented matching techniques

- Treat each tuple as a string and apply string matching algorithms
- **Exploit the structured nature of data – hand-crafted matching rules**
- **Automatically discover matching rules from training data – supervised learning**
- Iteratively assign tuples to clusters, no need of training data – clustering
- Model the matching domain with a probability distribution and reason with the distribution to take matching decisions – probabilistic approaches
- Exploit correlations among tuple pairs to match them all at once – collective matching

2nd Challenge

- Efficiently match a very large amount (tens of millions) of tuples
  - Record-set oriented matching: A potentially large set (or two sets) of records needs to be compared
  - Aims at minimizing the number of tuple pairs to be compared and perform each of the comparisons efficiently

Record-set oriented matching techniques

- For minimizing the number of tuple pairs to be compared
  - Hashing the tuples into buckets and only match those within a bucket
  - Sorting the tuples using a key and then compare each tuple with only the previous (w-1) tuples, for a pre-defined window size w
  - Index tuples using an inverted index on one attribute, for instance
  - Use a cheap similarity measure to quickly group tuples into overlapping clusters called canopies
  - Use representatives: tuples that represent a cluster of matching tuples against which new tuples are matched
  - Combine the techniques: because using a single heuristic runs the risk of missing tuple pairs that should be matched but are not

- And for minimizing the time taken to match each pair
  - Short-circuiting the matching process – exit immediately if one pair of attributes doesn't match

Record-set oriented matching techniques

- For minimizing the number of tuple pairs to be compared
  - Hashing the tuples into buckets and only match those within a bucket
  - **Sorting the tuples using a key and then compare each tuple with only the previous (w-1) tuples, for a pre-defined window size w**
  - Index tuples using an inverted index on one attribute, for instance
  - Use a cheap similarity measure to quickly group tuples into overlapping clusters called canopies
  - Use representatives: tuples that represent a cluster of matching tuples against which new tuples are matched
  - Combine the techniques: because using a single heuristic runs the risk of missing tuple pairs that should be matched but are not

- And for minimizing the time taken to match each pair
  - Short-circuiting the matching process – exit immediately if one pair of attributes doesn't match

Outline Record-oriented Matching

- **Record-oriented matching approaches**
  - Rule-based matching
  - Learning-based matching

- **Scaling up record-oriented matching**
  - Sorting: Sorted Neighborhood Method (SNM)

Rule-based matching

- Hand-crafted matching rules that can be (linearly weighted) combined through:

  $\mathbf{sim(x,y) = \sum_{i=1}^{n} \alpha_i . sim_i(x,y)}$

  that returns the similarity score between two tuples x and y, where:
  - **n** is the nb attributes in each table X and Y
  - $\mathbf{sim_i(x,y)}$ is the similarity score between the i-th attributes of x and y
  - $\boldsymbol{\alpha_i}$ is a pre-specified weight indicating the importance of the i-th attribute to the total similarity score
    - $\boldsymbol{\alpha_i}$ **in [0,1];** $\mathbf{\sum_{i=1}^{n} \alpha_i = 1}$

- If $\mathbf{sim(x,y) >= \beta}$ we say tuples x and y **match**

Example

Table X

Name	SSN	Addr
Jack Lemmon	430-871-8294	Maple St
Harrison Ford	292-918-2913	Culver Blvd
Tom Hanks	234-762-1234	Main St
…	…	…

Table Y

Name	SSN	Addr
Ton Hanks	234-162-1234	Main Street
Kevin Spacey	928-184-2813	Frost Blvd
Jack Lemon	430-817-8294	Maple Street
…	…	…

- To match names, define a similarity function $\mathbf{sim_{Name}(x,y)}$ based on the Jaro-Winkler distance
- To match SSNs, define a function $\mathbf{sim_{SSN}(x,y)}$ based on edit distance, etc
- $\mathbf{sim(x,y) = 0.3*sim_{Name}(x,y) + 0.3*sim_{SSN}(x,y) + 0.2*sim_{Addr}(x,y)}$

Complex matching rules (1)

- Linearly weighted matching rules do not work well when encoding more complex matching knowledge
  - Ex: two persons match if their names match approximately and either the SSN matches exactly or otherwise the addresses must match exactly
- Modify the similarity functions
  - Ex: $sim'_{SSN}(x,y)$ returns true only if the SSN match exactly; analogous with $sim'_{Address}(x,y)$
- And then the matching rule would be:

```
If sim_name(x,y) < 0.8 then return "no match"
Else if sim'_SSN(x,y) = true then return "match"
Else if sim'_SSN(x,y) >= 0.9 and sim'_Address(x,y) =
  true then return "match"
Else return "no match"
```

Complex matching rules (2)

- This kind of rules are often written in a high-level declarative language
  - Easier to understand, debug, modify and maintain

- Still, it is labor intensive to write good matching rules
- Or not clear at all how to write them
- Or difficult to set the parameters α, β

Learning-based matching

- Supervised learning
  - can also be unsupervised (clustering)
- Idea: learn a matching model M from the training data, then apply M to match new tuple pairs.
- Training data has the form:

  $T = \{(x_1, y_1, l_1), (x_2, y_2, l_2), \ldots, (x_n, y_n, l_n)\}$

  where each triple (x_i, y_i, l_i) consists of a tuple pair (x_i, y_i) and a label l_i with value "yes" if x_i matches y_i and "no" otherwise.

Training (1)

- Define a set of features $f_1, f_2, \ldots, f_m$ thought to be potentially relevant to matching
  - each f_i quantifies one aspect of the domain judged possibly relevant to matching the tuples
  - Each feature f_i is a function that takes a tuple pair `(x,y)` and produces a numerical, categorical, or binary value.
- The learning algorithm will use the training data to decide which features are in fact relevant

Training (2)

- Convert each training example (x_i, y_i, l_i) in the set T into a pair:

  $(<f_1(x_i, y_i), f_2(x_i, y_i), \ldots f_m(x_i, y_i)>, c_i)$

  where $v_i = <f_1(x_i, y_i), f_2(x_i, y_i), \ldots f_m(x_i, y_i)>$ is a feature vector that encodes the tuple pair (x_i, y_i) in terms of the features and c_i is an appropriately transformed version of label l_i
- Training set T is converted into a new training set T':

  $\{(v_1, c_1), (v_2, c_2), \ldots, (v_n, c_n)\}$

  and then we apply a learning algorithm such as SVM or Decision Trees to T' to learn a matching model M

Matching

- Given a new pair **(x,y)**, transform it into a feature vector

  $\mathbf{v} = \langle f_1(x, y), f_2(x, y), \ldots f_m(x, y) \rangle$

- And then apply model M to predict whether **x** matches **y**

Example

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

Matches:
(x_1, y_1)
(x_3, y_2)

Table Y

	Name	Phone	City	State
y_1	David D. Smith	395 9462	Madison	WI
y_2	Daniel W. Smith	256 1212	Madison	WI

Goal: learn a linearly weighted rule to match x and y

$$\mathbf{sim(x,y) = \sum_{i=1}^{n} \alpha_i . sim_i(x,y)}$$

Training data

```
<x1 = (Mike Williams, (425) 247 4893, Seattle, WA),
  y1 = (M. Williams, 247 4893, Redmond, WA), yes>
<x2 = (Richard Pike, (414) 256 1257, Milwaukee, WI),
   y2 = (R. Pike, 256 1237, Milwaukee, WI), yes>
<x3 = (Jane McCain, (206) 111 4215, Renton, WA),
   y3 = (J.M. McCain, 112 5200, Renton, WA), no>
```

- Consider 6 possibly relevant features:
  - $f_1(x,y)$ and $f_2(x,y)$: Jaro-Winkler and edit distance between person names of tuples x and y
  - $f_3(x,y)$: edit distance between phone numbers, ignoring the area code
  - $f_4(x,y)$ and $f_5(x,y)$: returns 1 if the city names and the state names match exactly
  - $f_6(x,y)$ returns 1 if the area code of x is an area code of the city of y

Transforming training data and learn

$$<v_1, c_1> = <[f_1(x_1, y_1), f_2(x_1, y_1), f_3(x_1, y_1), f_4(x_1, y_1), f_5(x_1, y_1), f_6(x_1, y_1)], 1>$$

$$<v_2, c_2> = <[f_1(x_2, y_2), f_2(x_2, y_2), f_3(x_2, y_2), f_4(x_2, y_2), f_5(x_2, y_2), f_6(x_2, y_2)], 1>$$

$$<v_3, c_3> = <[f_1(x_3, y_3), f_2(x_3, y_3), f_3(x_3, y_3), f_4(x_3, y_3), f_5(x_3, y_3), f_6(x_3, y_3)], 0>$$

- Goal: learn the weight α_i,with i in [1, 6] that gives a linearly weighted matching rule of the form: $sim(x,y) = \sum_{i=1}^{6} \alpha_i . f_i(x,y)$
- Perform a **least-squares linear regression** on the transformed data set for finding the weights α_i that minimize the squared error:

  $$\sum_{i=1}^{3} (c_i - \sum_{j=1}^{6} \alpha_j . f_j(v_i))^2$$

  where c_i is the label associated with feature vector v_i and $f_j(v_i)$ is the j-th element of feature vector v_i
- Learn β from the training set by setting it to the value that lets us minimize the number of incorrect matching predictions.

Advantages/inconvenients supervised learning

- Advantages:
  - Can automatically examine a large set of features to select the most useful ones
  - Can construct very complex rules, very difficult to construct in rule-based learning
- Inconvenients:
  - Requires a large number of training examples which can be labor intensive to obtain

Outline Record-oriented matching

- Record-oriented matching approaches
  - Rule-based matching
  - Learning-based matching

- **Scaling up record-oriented matching**
  - Sorting: Sorted Neighborhood Method (SNM)

Record Pairs as Matrix

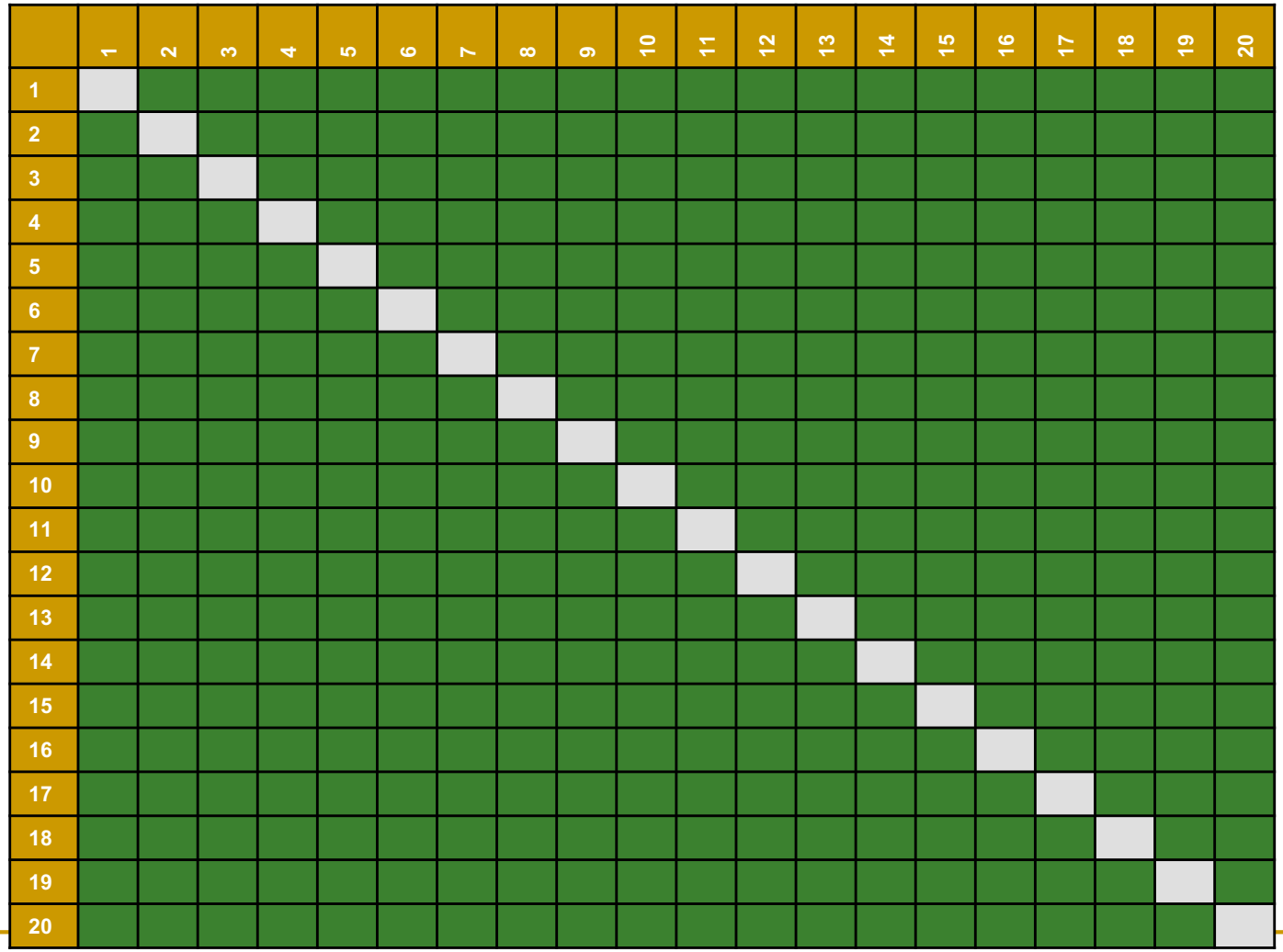
	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20
1																				
2																				
3																				
4																				
5																				
6																				
7																				
8																				
9																				
10																				
11																				
12			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

Number of comparisons: All pairs

	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																				

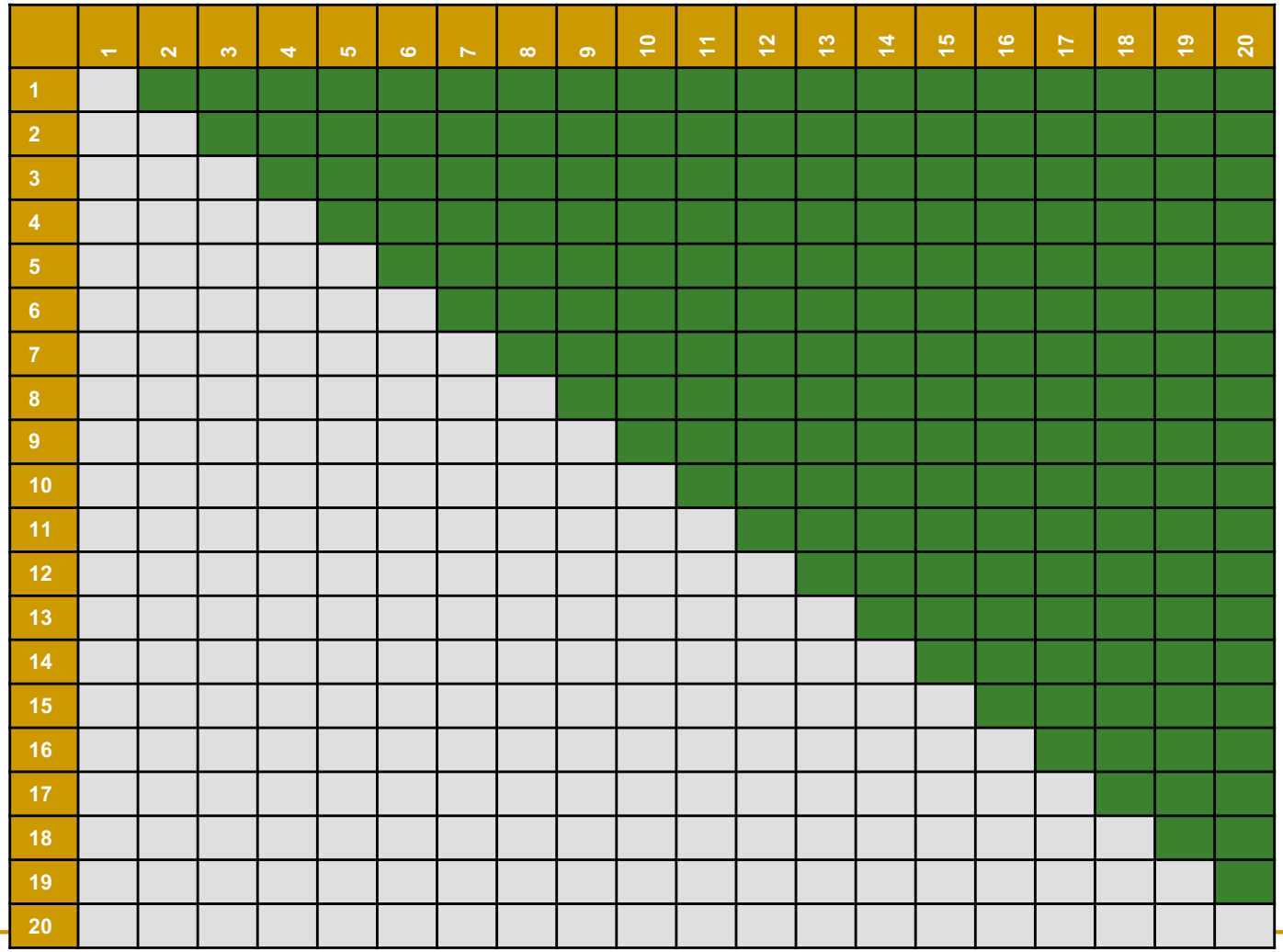
400 comparisons

Reflexivity of Similarity



380 comparisons

Symmetry of Similarity



190 comparisons

Complexity

- Problem: Too many comparisons!
  - 10.000 customers => 49.995.000 comparisons
    - $(n^2 - n) / 2$
    - Each comparison is already expensive.

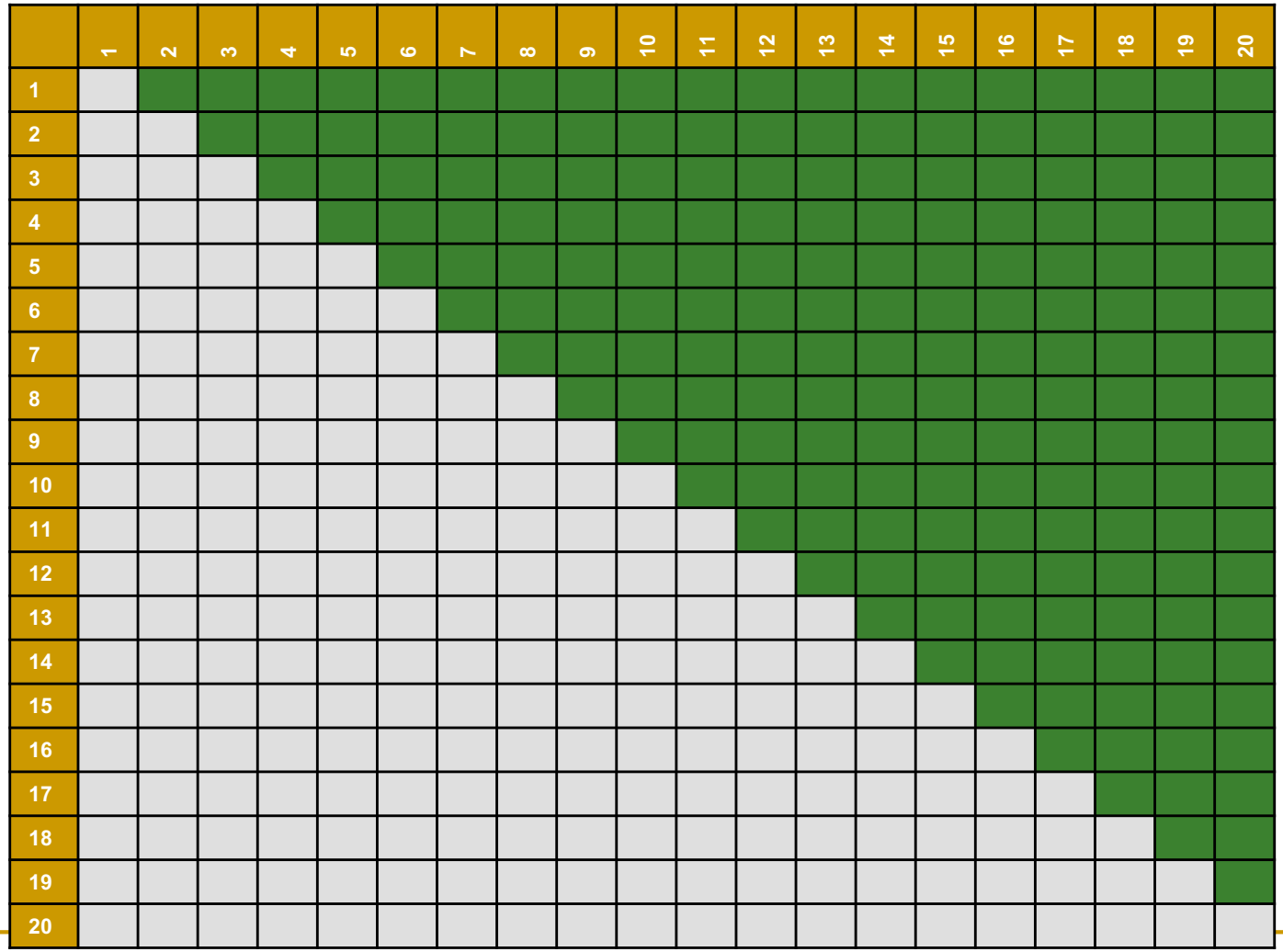
- Idea: Avoid comparisons…
  - … by filtering out individual records.
  - … by partitioning the records and comparing only within a partition.

Partitioning / Blocking

- Partition the records (horizontally) and compare pairs of records only within a partition
  - Ex1: Partitioning by first two zip-digits
    - Ca. 100 partitions in Germany
    - Ca. 100 customers per partition
    - => 495.000 comparisons
  - Ex2: Partition by first letter of surname
  - …

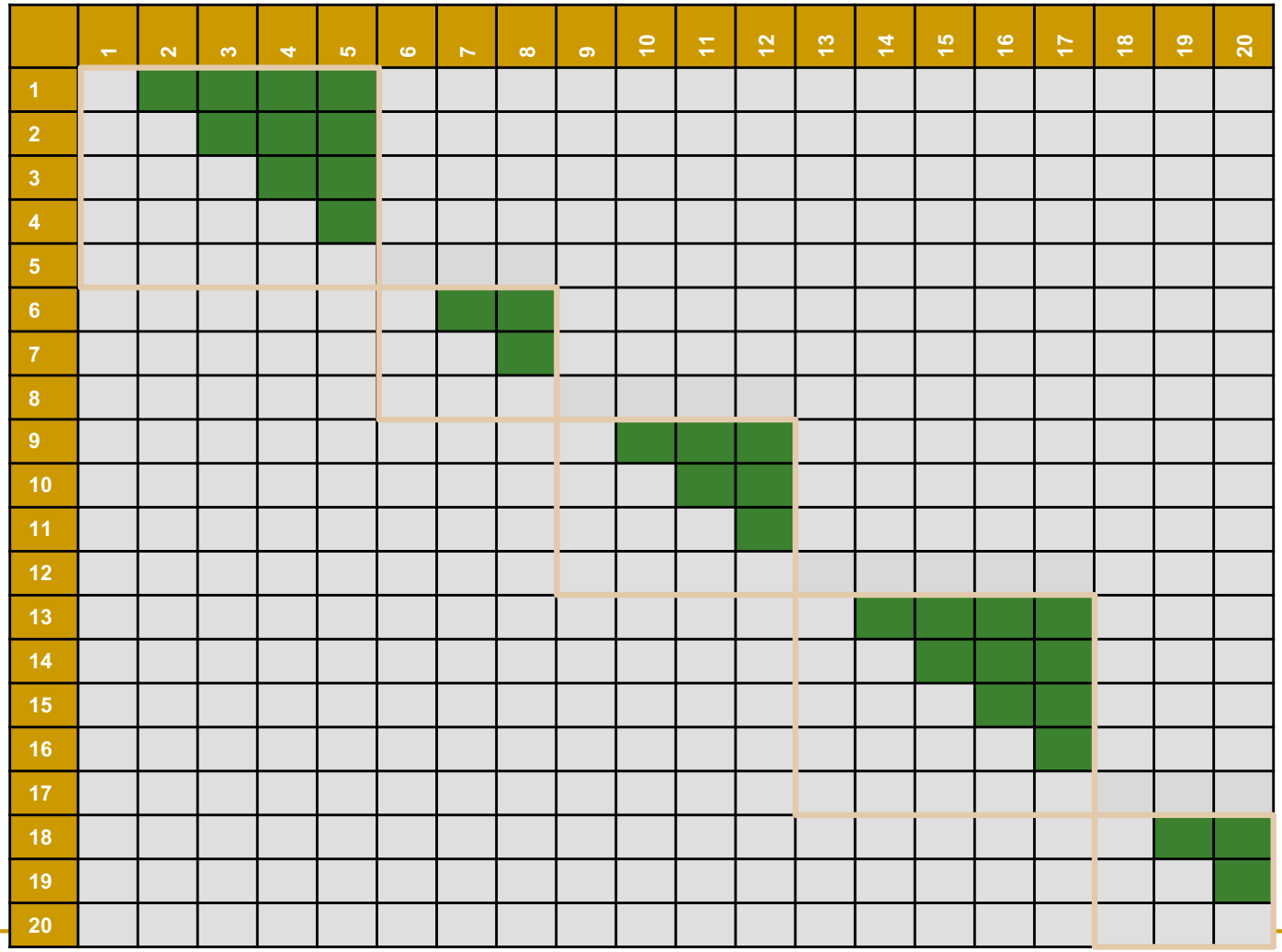
- Idea: Partition multiple times by different criteria
  - Then apply **transitive closure** on discovered duplicates.

Records sorted by ZIP



190 comparisons

Blocking by ZIP



32 comparisons

Sorted Neighbourhood Method - SNM (or Windowing)

- **Concatenate** all records to be matched in a single file (or table)
- **Sort** the records using a pre-defined key based on the values of the attributes for each record
- **Move** a window of a specific size `w` over the file, comparing only the records that belong to this window

1. Create key

- Compute a key for each record by extracting relevant fields or portions of fields

Example:

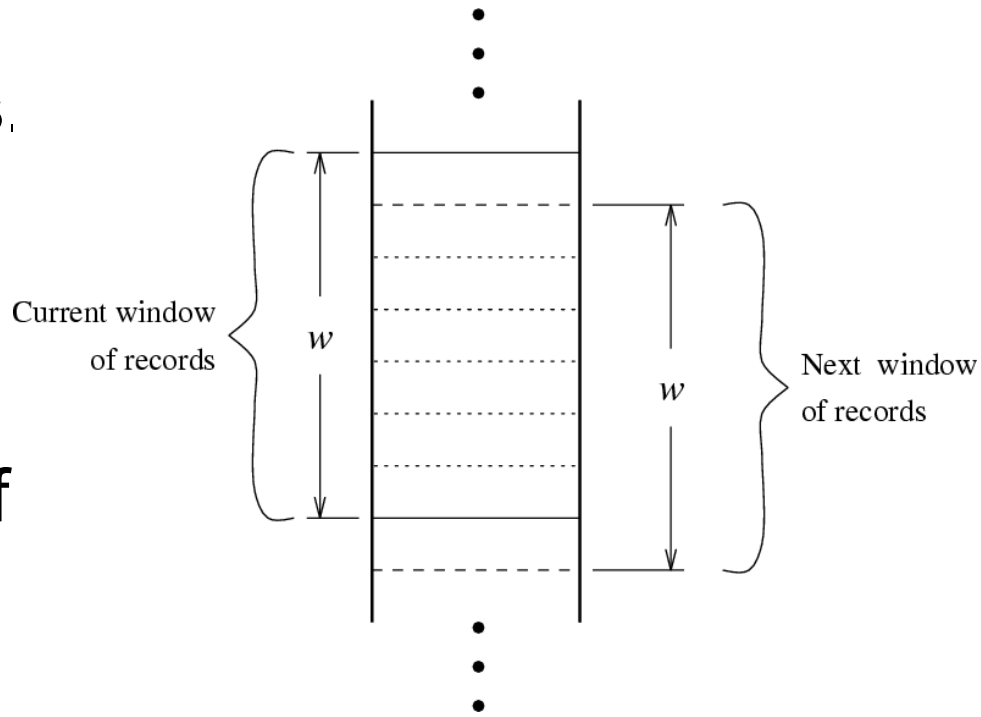
First	Last	Address	ID	Key
Sal	Stolfo	123 First Street	45678987	STLSAL123FRST456

2. Sort Data

- Sort the records in the data list using the key in step 1
- This can be very time consuming
  - O(NlogN) for a good algorithm,
  - $O(N^2)$ for a bad algorithm

3. Merge records

- Move a fixed size window through the sequential list of records.
- This limits the comparisons to the records in the window
- To compare each pair of records, a set of complex rules (called equational theory) is applied



Considerations

- What is the optimal window size while
  - Maximizing accuracy
  - Minimizing computational cost
- The effectiveness of the SNM highly depends on the key selected to sort the records
  - A key is defined to be a sequence of a subset of attributes
  - Keys must provide sufficient discriminating power

Example of Records and Keys

First	Last	Address	ID	Key
Sal	Stolfo	123 First Street	45678987	STLSAL123FRST456
Sal	Stolfo	123 First Street	45678987	STLSAL123FRST456
Sal	Stolpho	123 First Street	45678987	STLSAL123FRST456
Sal	Stiles	123 Forest Street	45654321	STLSAL123FRST456

Equational Theory - Example

- Two names are spelled nearly identically and have the same address
  - It may be inferred that they are the same person
- Two social security numbers are the same but the names and addresses are totally different
  - Could be the same person who moved
  - Could be two different people and there is an error in the social security number

A simplified rule in English

```
Given two records, r1 and r2
IF the last name of r1 equals the last
  name of r2,
AND the first names differ slightly,
AND the address of r1 equals the address
  of r2
THEN
  r1 is equivalent to r2
```

Building an equational theory

- The process of creating a good equational theory is similar to the process of creating a good knowledge-base for an expert system

- In complex problems, an expert's assistance is needed to write the equational theory

Looses some matching pairs

- In general, no single pass (i.e. no single key) will be sufficient to catch all matching records
- An attribute that appears first in the key has higher discriminating power than those appearing after them
  - If an employee has two records in a DB with SSN 193456782 and 913456782, it's unlikely they will fall under the same window

Possible solutions

- Goal: To increase the number of similar records being matched

- Widen the scanning window size, w
- Execute several independent runs of the SNM
  - Use a different key each time
  - Use a relatively small window
  - Call this the Multi-Pass approach

Multi-pass approach

- Each independent run of the Multi-Pass approach will produce a set of pairs of records
  - Although one field in a record may be in error, another field may not

- Transitive closure can be applied to those pairs to be merged

Transitive closure example

```
IF A similar to B
  AND B similar to C
     THEN A similar to C
```

From the example:

789912345 Kathi Kason 48 North St. (A)
879912345 Kathy Kason 48 North St. (B)
879912345 Kathy Smith 48 North St. (C)

Example of multi-pass matches

Pass 1 (Lastname discriminates)

KSNKAT48NRTH789 (Kathi Kason 789912345)

KSNKAT48NRTH879 (Kathy Kason 879912345)

Pass 2 (Firstname discriminates)

KATKSN48NRTH789 (Kathi Kason 789912345)

KATKSN48NRTH879 (Kathy Kason 879912345)

Pass 3 (Address discriminates)

48NRTH879KSNKAT (Kathy Kason 879912345)

48NRTH879SMTKAT (Kathy Smith 879912345)

References

- [Batini2006] “Data Quality: Concepts, Methodologies and Techniques”, C. Batini and M. Scannapieco, Springer-Verlag, 2006
- [Fan2012] “Foundations of Data Quality Management”, W. Fan and F. Geerts, 2012
- [Christen2012] “Data Matching”, Peter Christen, Springer.
- [Naumann2010] “An Introduction to Duplicate Detection”, F. Naumann and Melanie Herschel, Morgan Claypool Publishers.
- [Doan2012] “Principles of Data Integration” by AnHai Doan, Alon Halevy, Zachary Ives, 2012.

Thank You!

Hope to see you in Lisbon for EDBT/ICDT 2019…