Philipps Universität Marburg

Summer 2022

Data Integration

Thorsten Papenbrock

Duplicates

Duplicate: Two representations (e.g. records, objects, XML structures) that represent the same real-world entity or concept

- Example:

Name	Street	Number
Ernie	Sesamstr.	2
Fienchen	Sesamstr.	1
Bert	Sesamstr.	2
Samson	Sesamstr.	1
Tiffy	Sesamstr.	3
Kermit	Sesamstr.	6
Grobi	Sesamstr.	4
Krümelmonster	Sesamstr.	8
Mumpitz	Sesamstr.	7
Oscar	Sesamstr.	5
Bibo	Sesamstr.	9
Graf Zahl	Sesamstr.	9¾
Kermitt	Sesamstr.	6
Finchen	Sesamstr.	1
Rumpel	Sesamstr.	1.5

Duplicates

Duplicate: Two representations (e.g. records, objects, XML structures) that represent the same real-world entity or concept

- Example:

Name	Street	Number
Ernie	Sesamstr.	2
Fienchen	Sesamstr.	1
Bert	Sesamstr.	2
Samson	Sesamstr.	1
Tiffy	Sesamstr.	3
Kermit	Sesamstr.	6
Grobi	Sesamstr.	4
Krümelmonster	Sesamstr.	8
Mumpitz	Sesamstr.	7
Oscar	Sesamstr.	5
Bibo	Sesamstr.	9
Graf Zahl	Sesamstr.	9¾
Kermitt	Sesamstr.	6
Finchen	Sesamstr.	1
Rumpel	Sesamstr.	1.5

Duplicates

Philipps Universität Marburg

Duplicate: Two representations (e.g. records, objects, XML structures) that represent the same real-world entity or concept

- Typically not equal but similar (w.r.t. values, attribute sets, references)
- Example:

- Origin examples:
  - Data integration: data silos, data sharing, data discovery
  - Errors in data entry: typos, redundancies, transmission
  - Fraudulent actions: manipulation, human mistakes

Duplicates

Duplicate: Two representations (e.g. records, objects, XML structures) that represent the same real-world entity or concept

- Typically not equal but similar (w.r.t. values, attribute sets, references)
- Effects:
  - Wrong decision making
    - Actions are carried out multiple times per entity
  - Inaccurate statistics
    - Number of entities is counted to high
  - Poorly trained machine learning models
    - Duplicate entries introduce bias and misclassifications
  - Software failures
    - Unique constraints may be violated
  - ...

All this costs a lot of money in practice!

Token-based Similarity: n-grams

k

- The function $to\ (x)$ splits the string x into short substrings of length n by sliding a window of size n over x; every slide creates an n-gram.
  - n=2: Bigrams; n=3: Trigrams
  - Number of n-grams = $|x|$ – n + 1
- Variation 1: Pad with n – 1 special characters
  - ➢ Emphasizes beginning and end of string
- Variation 2: Include positional information
  - ➢ Useful to weight tokens by their positions

"Thorsten Papenbrock"

"Th", "ho", "or", …

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: n-grams

WolframAlpha computational intelligence.

n-grams "levenshtein"

NATURAL LANGUAGE | MATH INPUT | EXTENDED KEYBOARD | EXAMPLES | UPLOAD | RANDOM

Assuming the input is referring to string encodings | Use "n" as a variable instead

Input interpretation

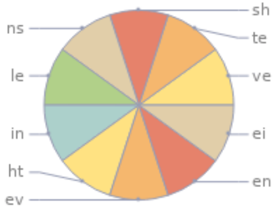
n–grams	levenshtein

Character-level bigrams

le | ev | ve | en | ns | sh | ht | te | ei | in

Character-level bigrams frequency pie chart

Show bar chart



String Similarity

Levenshtein Distance

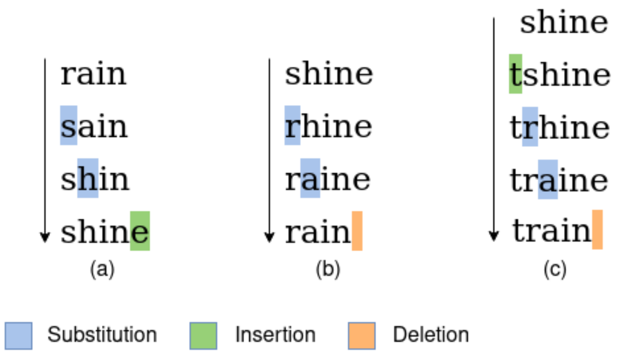
This is the challenge!

Definition:

- $dist_{levenshtein}(x,y)$ = minimum number of edits (insert, delete, replace) that transform the string x into string y
- The most popular metric for describing the edit-distance of strings

Operations:

1. insert a character into the string
2. delete a character from the string
3. replace (substitute) a character with a different character



Examples:

- $dist_{levenshtein}$('table', 'cable') = 1
  (1 replace)
- $dist_{levenshtein}$('Thorsten Papenbrock', ' Papenbrock, Thorsten') = 19
  (9 deletes + 10 inserts)

String Similarity

Levenshtein Distance

Calculation:

- Calculating the minimum edit-distance is a case for dynamic programming
- Optimality principle:
  - Any minimum edit-distance of two substrings must be part of the best overall solution.
- Dynamic programming algorithm:
  1. Initialize a matrix M of size (|*x*|+1) x (|*y*|+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: $dist_{levenshtein}(x, y) = M_{|x|,|y|}$

String Similarity

Levenshtein Distance

Philipps Universität Marburg

Calculation example:

		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					

$M_{i,0} = i$ and $M_{0,j} = j$

		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					

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

		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

Every path leads to the same optimal solution!

Levenshtein Distance

Similarity:

- $sim_{levenshtein}(x,y) = 1 - dist_{levenshtein}(x,y) / max(|x|,|y|)$

x	y	Lev. Distance	Lev. Similarity
Jones	Johnson	4	0.43
Paul	Pual	2	0.5
Paul Jones	Jones, Paul	11	0

Discussion:

- Robust against different forms and positions of typos
- Easy to understand and implement
- Unsupervised similarity metric
- Sensitive to word order changes
- Expensive to calculate

Levenshtein Distance

Complexity:

- Time: $O(|x| \cdot |y|)$ (calculate entire matrix)
- Space: $O(\min(|x|,|y|))$ (store only two rows of the matrix)

Properties:

- $0 \leq dist_{levenshtein}(x,y) \leq max(|x|,|y|)$
- $|\,|x| - |y|\,| \leq dist_{levenshtein}(x,y)$

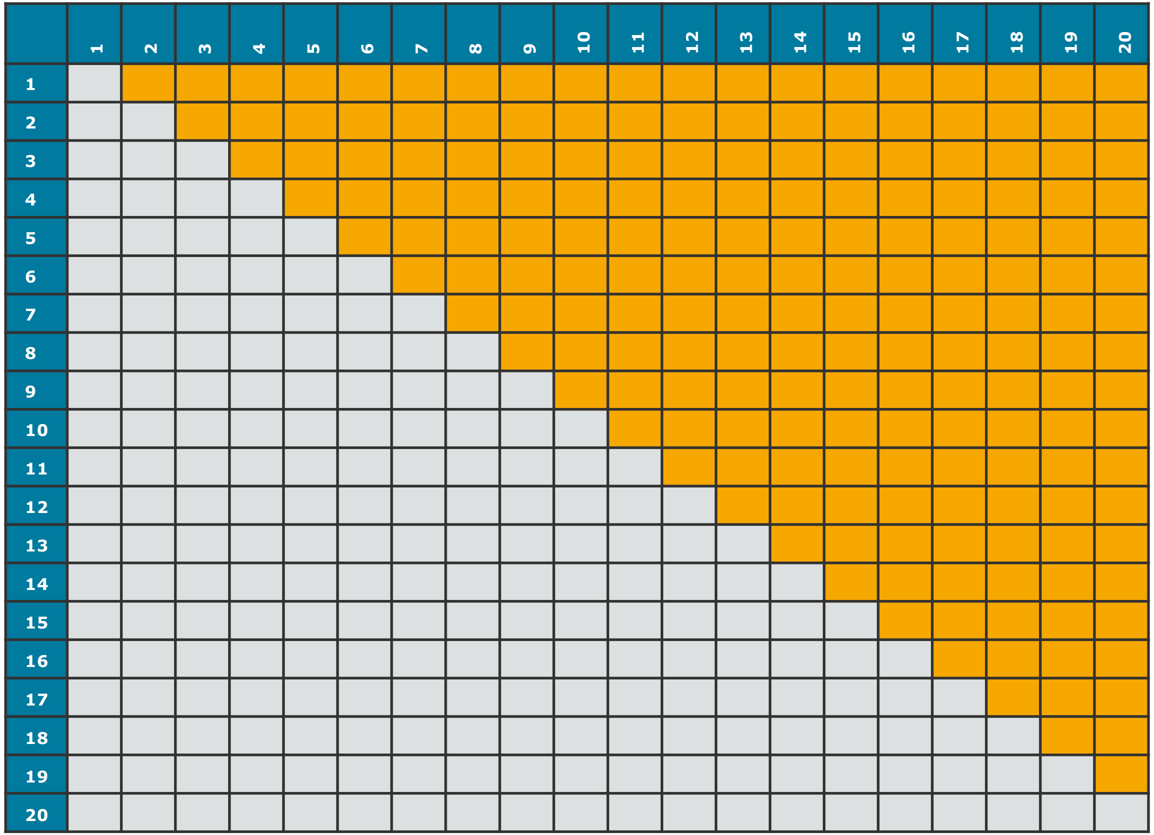
Good lower-bound estimate to possibly skip the exact distance computation!

Cost models (extension): Assign different costs to edit-operations.

- By operation:
  - E.g. replace costs 0.5 but insert/delete cost 1.0 to punish string length changes.
- By character:
  - E.g. OCR (m ≃ n, 1 ≃ l) or keyboard (a ≃ s) or brain (6 ≃ 9) or biology (a ≃ t)

Sorted Neighborhood Method (SNM)

M. A. Hernández, S. J. Stolfo
Real-world Data is Dirty: Data Cleansing and The Merge/Purge Problem.
1998

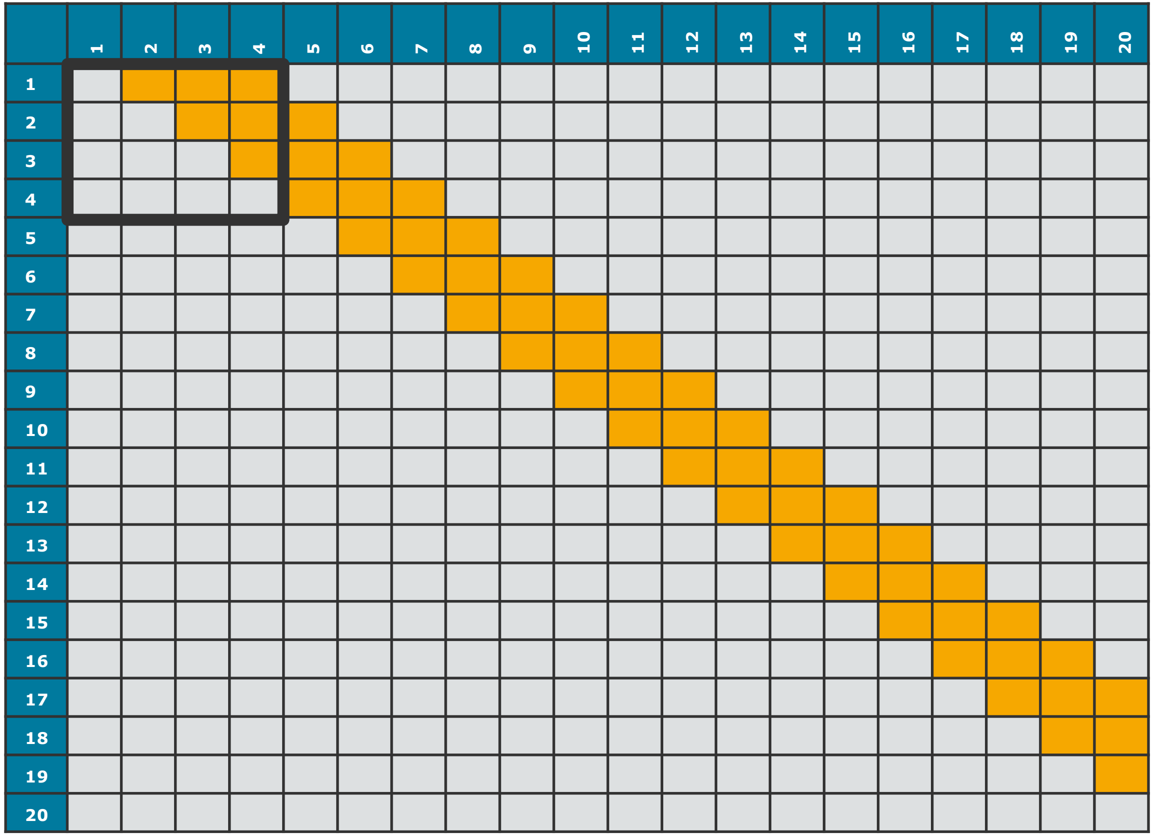


Sorted Neighborhood Method

- Sort tuples so that similar tuples are close to each other.
- Compare tuples only within a small neighborhood (= window).

Sorted Neighborhood Method (SNM)

M. A. Hernández, S. J. Stolfo
Real-world Data is Dirty: Data Cleansing and The Merge/Purge Problem.
1998



Sorted Neighborhood Method

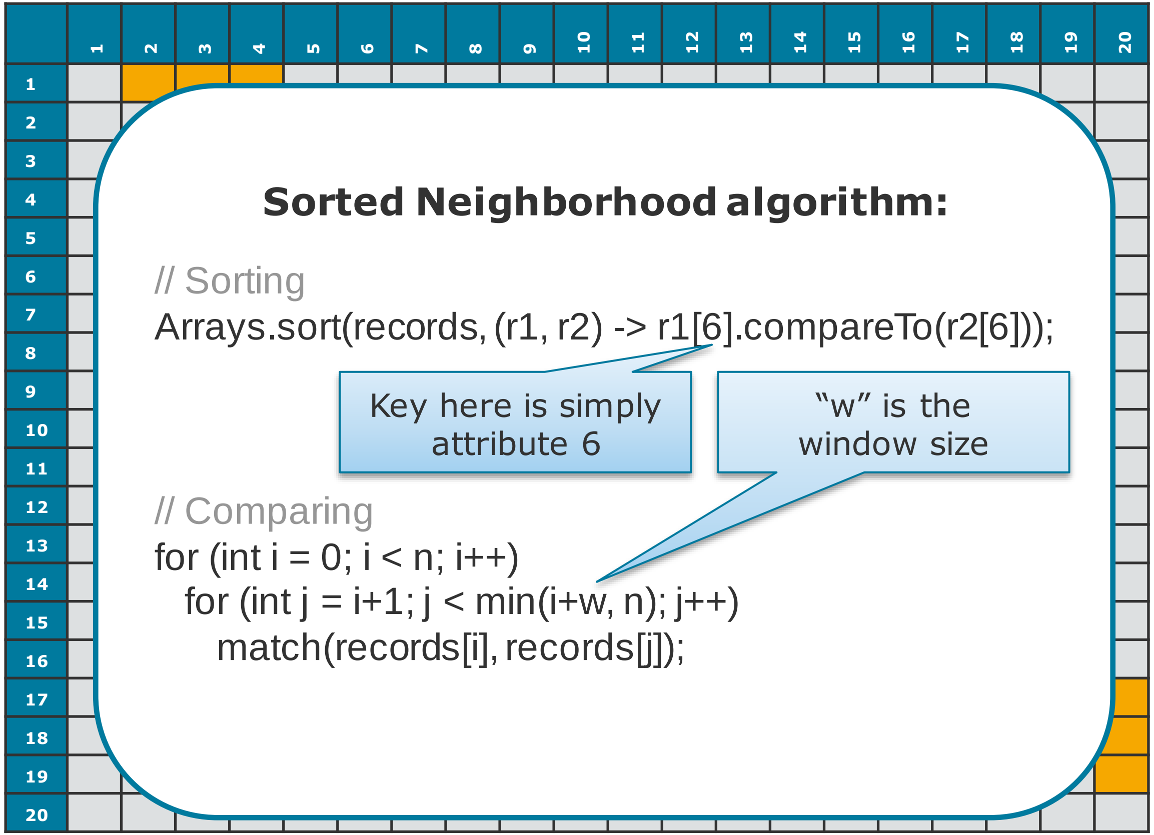
- Sort tuples so that similar tuples are close to each other.
- Compare tuples only within a small neighborhood (= window).

1. Generate a key
   - E.g. SSN
   - E.g. Name[1-3] + Age + …
2. Sort entire relation by the key
   - Sort records physically or create a sorted index.
3. Slide a window over sorted tuples
   - Compare all pairs of tuples within the sliding window.

Sorted Neighborhood Method (SNM)

M. A. Hernández, S. J. Stolfo
Real-world Data is Dirty: Data Cleansing and The Merge/Purge Problem.
1998

Philipps Universität Marburg



Sorted Neighborhood algorithm:

```
// Sorting
Arrays.sort(records, (r1, r2) -> r1[6].compareTo(r2[6]));

// Comparing
for (int i = 0; i < n; i++)
   for (int j = i+1; j < min(i+w, n); j++)
      match(records[i], records[j]);
```

Key here is simply attribute 6

"w" is the window size

Sorted Neighborhood Method

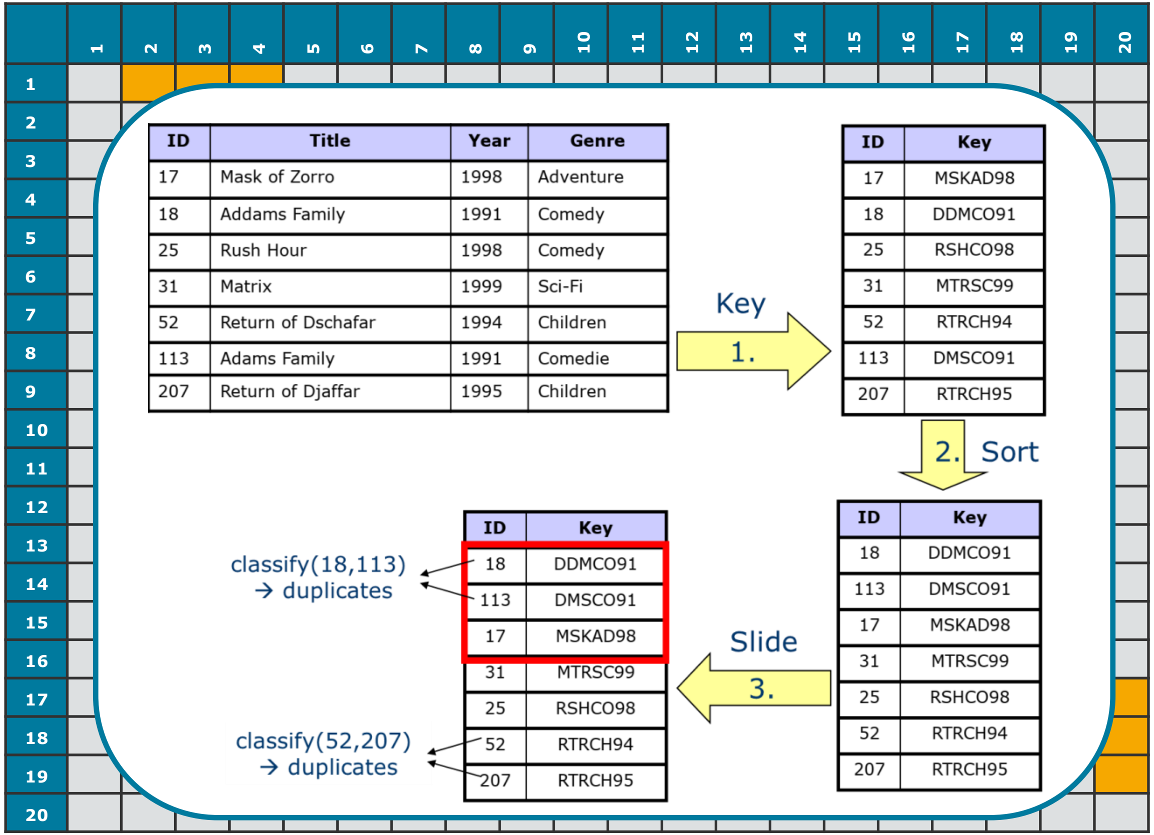
- Sort tuples so that similar tuples are close to each other.
- Compare tuples only within a small neighborhood (= window).

1. Generate a key
   - E.g. SSN
   - E.g. Name[1-3] + Age + …
2. Sort entire relation by the key
   - Sort records physically or create a sorted index.
3. Slide a window over sorted tuples
   - Compare all pairs of tuples within the sliding window.

Sorted Neighborhood Method (SNM)

M. A. Hernández, S. J. Stolfo
Real-world Data is Dirty: Data Cleansing and The Merge/Purge Problem. 1998

Philipps Universität Marburg



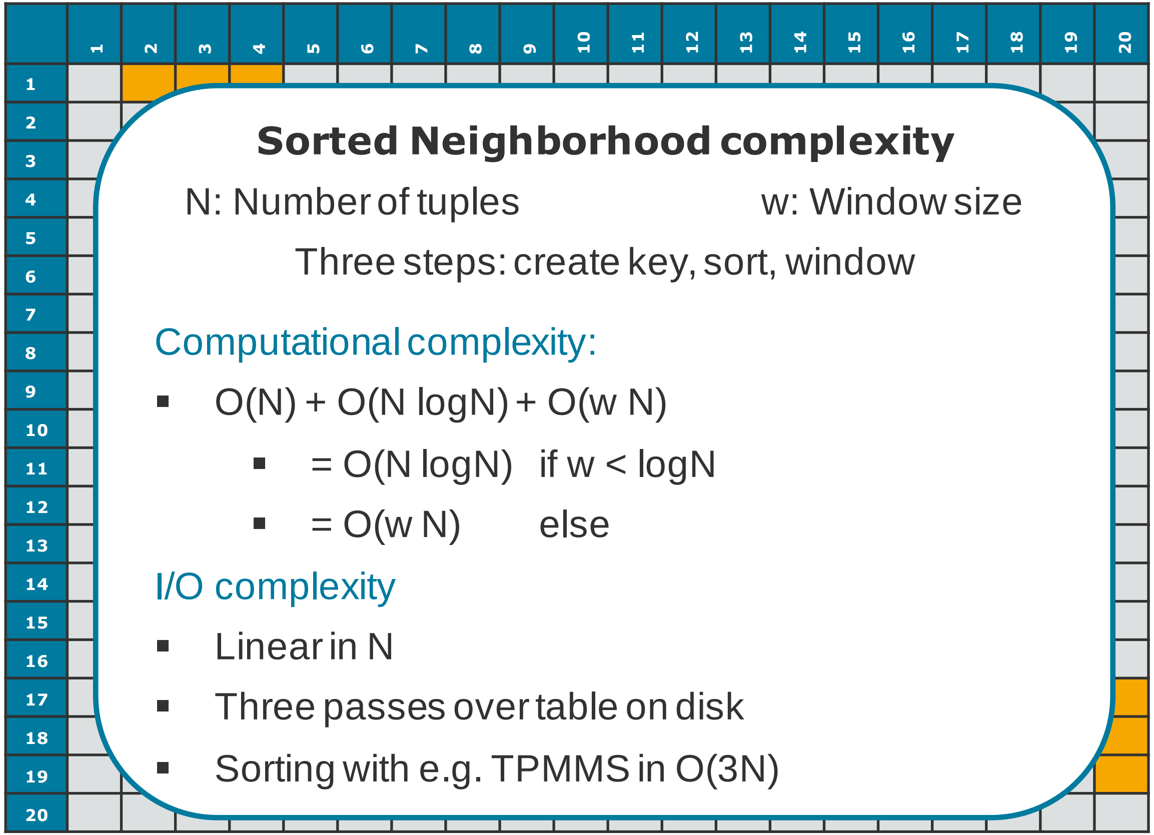
Sorted Neighborhood Method

- Sort tuples so that similar tuples are close to each other.
- Compare tuples only within a small neighborhood (= window).

1. Generate a key
   - E.g. SSN
   - E.g. Name[1-3] + Age + …
2. Sort entire relation by the key
   - Sort records physically or create a sorted index.
3. Slide a window over sorted tuples
   - Compare all pairs of tuples within the sliding window.

Sorted Neighborhood Method (SNM)

M. A. Hernández, S. J. Stolfo
Real-world Data is Dirty: Data Cleansing and The Merge/Purge Problem. 1998



Sorted Neighborhood complexity

N: Number of tuples w: Window size

Three steps: create key, sort, window

Computational complexity:

- O(N) + O(N logN) + O(w N)
  - = O(N logN) if w < logN
  - = O(w N) else

I/O complexity

- Linear in N
- Three passes over table on disk
- Sorting with e.g. TPMMS in O(3N)

Sorted Neighborhood Method

- Sort tuples so that similar tuples are close to each other.
- Compare tuples only within a small neighborhood (= window).

1. Generate a key
   - E.g. SSN
   - E.g. Name[1-3] + Age + …
2. Sort entire relation by the key
   - Sort records physically or create a sorted index.
3. Slide a window over sorted tuples
   - Compare all pairs of tuples within the sliding window.