Algorithms for Historical Record Linkage

Marijn Schraagen, Leiden Institute of Advanced Computer Science
Outline

- Record linkage definition
- Indexing edit distance
- Name cores
- Family reconstruction
- Spatio-temporal linkage
- Summary
Record linkage definition

• Record linkage is the problem of linking **different records** that contain information about the **same entity** without a **unique entity identifier**

• Examples:
  • Medical records from different hospitals about the same patient
  • Customer records from different online stores created by the same customer
  • Civil events (birth, marriage, death) for a person in historical sources
Historical record linkage

- Genlias/WieWasWie: Dutch historical civil certificates

- Available under a research license in digitized format

  marriage, 05-05-1875, Breda, Joris Dekker, Adriana Putters
Historical record linkage

• Example: linkage on marriage certificates

• A certificate contains several people: bride and groom, parents bride, parents groom

• The parents are also married, which is listed in a different record
Indexing edit distance

• Levenshtein distance between string $a$ and $b$: minimum number of character insertions, deletions and substitutions necessary to transform $a$ into $b$

• Uniform cost: no weighting for, e.g., string length, character confusion probability, position of error, character transpositions, number of errors, etc.

• However, for most datasets Levenshtein distance is useful to find related records based on string similarity between record fields
Indexing edit distance

• Every record in a database might be similar to any of the other records

• A full check of every possibility requires $n^2$ similarity computations for the number of records $n$ (quadratic complexity), and a single check takes $n \cdot m$ operations for string lengths $n$ and $m$ (also quadratic)

• For 10 million records of average length 20 this amounts to $40000000000000000$ operations (40 quadrillion)
Bit vector index

- Research goal: develop a more efficient method to find every pair of records within a reasonable edit distance

- General idea: create a tree of records where adjacency in the tree corresponds to small Levenshtein distance of records

- Tree is used as index on characters

- A record is represented as a bit vector where bits indicate presence of characters
A different bit results from an insertion, a deletion or one side of a substitution.

Not all insertions, deletions and substitutions are visible as bit difference.

- 0-1 differences: insertion or substitution
- 1-0 differences: deletion or substitution
• **0-1 differences**: insertion or substitution
  - Number of 0-1 differences \((a)\) \(\leq\) edit distance

• **1-0 differences**: deletion or substitution
  - Number of 1-0 differences \((b)\) \(\leq\) edit distance

• Edit distance \(\geq\) \(\max(a,b)\)
Example: Xandrea, th=1

Bit vector Xandrea: 11110100

- Incorrect 0: 0
- Incorrect 1: 0
Example: Xandrea, th=1

Bit vector Xandrea: 11110100

- Incorrect 0: 0
- Incorrect 1: 0
Example: Xandrea, th=1

Bit vector Xandrea: 11110100

- Incorrect 0: 1
- Incorrect 1: 0
Example: Xandrea, th=1

Bit vector Xandrea: 11110100

- Incorrect 0: 1
- Incorrect 1: 0
Example: Xandrea, th=1

Bit vector Xandrea: 11110100

- Incorrect 0: 2
- Incorrect 1: 0
Example: Xandrea, th=1

Bit vector Xandrea: 11110100

• Incorrect 0: 0
• Incorrect 1: 0
Example: Xandrea, th=1

Bit vector Xandrea: 11110100

- Incorrect 0: 1
- Incorrect 1: 0
Example: Xandrea, th=1

Bit vector Xandrea: 11110100

- Incorrect 0: 2
- Incorrect 1: 0
Example: Xandrea, th=1

Bit vector Xandrea: 11110100

- Incorrect 0: 1
- Incorrect 1: 0
Example: Xandrea, th=1

Bit vector Xandrea: 11110100

- Incorrect 0: 1
- Incorrect 1: 1
Example: Xandrea, th=1

Bit vector Xandrea: 11110100

- Incorrect 0: 2
- Incorrect 1: 1
Example: Xandrea, th=1

Bit vector Xandrea: 11110100

- Incorrect 0: 1
- Incorrect 1: 1
Example: Xandrea, th=1

Bit vector Xandrea: 11110100

- Incorrect 0: 0
- Incorrect 1: 0
Note: tree traversal is a simple operation, while distance computation of candidates is quadratic in the length of the strings
Indexing edit distance

• For a dataset of several million marriage certificates the method can check a few hundred to 1000+ records per second (depending on the threshold) against the full set

• Completeness of the results is guaranteed (i.e., mathematically proven)

• Demo application: create pedigrees from marriage records

• Tomorrow at Famillement
Indexing edit distance
Name cores

- Edit distance considers all characters equal
- However, some characters are more important than others
- Therefore, edit distance does not always capture name variation
- Examples:
  - Aaltje, Aal, Aalie, Altje, Aaltgijn, Aaltjen, Aeltina, Aeltje, Aaeltjen, Alina, Altijen
  - Bastiaan, Basjaan, Basta, Basteyen, Bastiaen, Bastiaens, Bastian, Bastjaan, Bastjaen, Bastjan, Bastjen, Bastyaen, Sebastiaan, Sebastiaen, Sebastian, Sebastianus
Name cores

- Research approach: compute a key for person names that contains only the important characters, i.e., the core of the name

- The key represents a name and variants of this name

- The key should be short and distinctive
  - Short: minimize spelling differences between variants
  - Distinctive: separate variants from non-variants

- Matching on keys is intended to be fast and to capture other aspects of name variation besides edit distance

- Learn from data, i.e., known variants, what the important characters are
Name cores

• Assumption: if a character is present in every variant of a name then this character is important

• Possible key: longest common subsequence (LCS) between known name variants

• Example:
  • Muhamed
  • Muhammad
  • Mohamed
  • Mohammed
  • Mehmed

• LCS: Mhmd
• Assumption: a character has certain features which predict whether or not this character will be part of the LCS

• Possible features:
  • Character (a-z)
  • Position in word
  • Word length
  • Number of syllables in word
  • Part of syllable: onset, nucleus, coda
  • Previous character
  • Next character
  • Distance to end of word
Model training

• Research goal: build a model for automatic name reduction for unseen names based on LCS keys of known variants in a training set

• Use features of characters in a machine learning setting
Model training

- Training data: 65,002 manually constructed name variant pairs
- LCS is constructed using all variants of a name
- Model: C4.5 decision tree
  - Other models have been investigated: Naïve Bayes, Support Vector Machine, Bayesian Network, 1-Nearest Neighbor classifier
  - Decision tree outperforms other models
Model training

- Training examples are constructed from a full name and the LCS
- Every character in the full name is a single training example
- A training example consists of values for several features of a character
Model training

- Example: Muhamed, LCS mhmd
- Construct 7 training examples
- Example for first \( m \): \([m,0,7,3,\text{onset},\#,u,6,1]\)
- Class: part of LCS (1) or not (0)

<table>
<thead>
<tr>
<th>Character</th>
<th>m</th>
<th>u</th>
<th>h</th>
<th>a</th>
<th>m</th>
<th>e</th>
<th>d</th>
</tr>
</thead>
<tbody>
<tr>
<td>Position</td>
<td>0</td>
<td>1</td>
<td>2</td>
<td>3</td>
<td>4</td>
<td>5</td>
<td>6</td>
</tr>
<tr>
<td>Word length</td>
<td>7</td>
<td>7</td>
<td>7</td>
<td>7</td>
<td>7</td>
<td>7</td>
<td>7</td>
</tr>
<tr>
<td>Syllables</td>
<td>3</td>
<td>3</td>
<td>3</td>
<td>3</td>
<td>3</td>
<td>3</td>
<td>3</td>
</tr>
<tr>
<td>Part of syllable</td>
<td>onset</td>
<td>nucleus</td>
<td>onset</td>
<td>nucleus</td>
<td>onset</td>
<td>nucleus</td>
<td>coda</td>
</tr>
<tr>
<td>Previous</td>
<td>#</td>
<td>m</td>
<td>u</td>
<td>h</td>
<td>a</td>
<td>m</td>
<td>e</td>
</tr>
<tr>
<td>Next</td>
<td>u</td>
<td>h</td>
<td>a</td>
<td>m</td>
<td>e</td>
<td>d</td>
<td>#</td>
</tr>
<tr>
<td>Distance to end</td>
<td>6</td>
<td>5</td>
<td>4</td>
<td>3</td>
<td>2</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>Class</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>1</td>
</tr>
</tbody>
</table>
For unseen names test examples can be constructed using the features.

The classification tree predicts 0 (not in name core) or 1 (in core).

Example tree:

```
Model application

<table>
<thead>
<tr>
<th>pos = 1</th>
<th>syllable part</th>
</tr>
</thead>
<tbody>
<tr>
<td>no</td>
<td></td>
</tr>
<tr>
<td></td>
<td>yes</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>pos &gt; 4</th>
</tr>
</thead>
<tbody>
<tr>
<td>no</td>
</tr>
<tr>
<td></td>
</tr>
<tr>
<td>yes</td>
</tr>
</tbody>
</table>

adjacent to [r,l]

<table>
<thead>
<tr>
<th>onset</th>
</tr>
</thead>
<tbody>
<tr>
<td>no</td>
</tr>
<tr>
<td></td>
</tr>
<tr>
<td>yes</td>
</tr>
</tbody>
</table>

nucleus

<table>
<thead>
<tr>
<th>no</th>
<th>1 (keep)</th>
</tr>
</thead>
<tbody>
<tr>
<td>yes</td>
<td>0 (discard)</td>
</tr>
</tbody>
</table>

coda

<table>
<thead>
<tr>
<th>pos &gt; 4</th>
</tr>
</thead>
<tbody>
<tr>
<td>no</td>
</tr>
<tr>
<td></td>
</tr>
<tr>
<td>yes</td>
</tr>
</tbody>
</table>

Harmen: 101000 = hr

H: pos = 1 (1)
a: pos ≠ 1, nucleus (0)
r: pos ≠ 1, coda, pos ≤ 4 (1)
m: pos ≠ 1, onset, adj. to r (0)
e: pos ≠ 1, nucleus (0)
n: pos ≠ 1, coda, pos > 4 (0)
Evaluation

- Using the name core model for record linkage
- Linking task: link couples consisting of two people, total four names
- For each name, predict the class of each character and construct the name core
- Append cores to construct a core sequence
- If two cores sequences are equal, then the corresponding records can be linked
**Sequence 1:** hr|but|jan|kolk
**Sequence 2:** hr|but|jan|kolk
**Match found**
Evaluation

- Full dataset contains 2.6 million records
- Evaluation on manually constructed test set of 6212 records
- Links are categorized as small edit distance (1-3) or large edit distance (≥4)

<table>
<thead>
<tr>
<th>edit distance</th>
<th>1-3</th>
<th>≥4</th>
</tr>
</thead>
<tbody>
<tr>
<td>amount of links</td>
<td>1730</td>
<td>134</td>
</tr>
<tr>
<td>precision</td>
<td>0.99</td>
<td>0.75</td>
</tr>
<tr>
<td>recall</td>
<td>0.78</td>
<td>0.15</td>
</tr>
<tr>
<td>F-measure</td>
<td>0.88</td>
<td>0.26</td>
</tr>
</tbody>
</table>
## Comparison to other methods

- Results of other linkage methods on the same test set

<table>
<thead>
<tr>
<th></th>
<th>Edit distance 1-3</th>
<th></th>
<th>Edit distance ≥ 4</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Precision</td>
<td>Recall</td>
<td>F-measure</td>
<td>Precision</td>
</tr>
<tr>
<td>Name cores</td>
<td>0.99</td>
<td>0.78</td>
<td>0.88</td>
<td>0.75</td>
</tr>
<tr>
<td>Traditional blocking</td>
<td>1.00</td>
<td>0.79</td>
<td>0.89</td>
<td>1.00</td>
</tr>
<tr>
<td>q-grams</td>
<td>0.99</td>
<td>0.91</td>
<td>0.95</td>
<td>0.97</td>
</tr>
<tr>
<td>Suffix array</td>
<td>0.36</td>
<td>0.84</td>
<td>0.48</td>
<td>0.13</td>
</tr>
<tr>
<td>Suffix array substring</td>
<td>0.15</td>
<td>0.97</td>
<td>0.24</td>
<td>0.09</td>
</tr>
<tr>
<td>Suffix array robust</td>
<td>0.31</td>
<td>0.93</td>
<td>0.45</td>
<td>0.18</td>
</tr>
<tr>
<td>Sorted array</td>
<td>0.36</td>
<td>0.92</td>
<td>0.47</td>
<td>0.21</td>
</tr>
<tr>
<td>Sorted array adaptive</td>
<td>0.80</td>
<td>0.90</td>
<td>0.84</td>
<td>0.63</td>
</tr>
<tr>
<td>th-Canopy clustering</td>
<td>0.99</td>
<td>0.86</td>
<td>0.92</td>
<td>0.98</td>
</tr>
<tr>
<td>nn-Canopy clustering</td>
<td>0.13</td>
<td>0.93</td>
<td>0.22</td>
<td>0.06</td>
</tr>
<tr>
<td>Sorted inverted index</td>
<td>0.14</td>
<td>0.83</td>
<td>0.49</td>
<td>0.28</td>
</tr>
<tr>
<td>th-String map</td>
<td>0.45</td>
<td>0.83</td>
<td>0.49</td>
<td>0.28</td>
</tr>
<tr>
<td>nn-String map</td>
<td>0.48</td>
<td>0.82</td>
<td>0.54</td>
<td>0.27</td>
</tr>
</tbody>
</table>
Comparison to other methods

• Note: most other methods are designed for high recall at the expense of precision (e.g., sorted array)
• Scalability of other methods is an issue, both in accuracy and computation time (e.g., q-grams)
Comparison to other methods

- Computation time by number of records
- Linear vs. quadratic (or worse) behaviour
Comparison to other methods

- Memory consumption by number of records
Method summary

- Train decision tree on character level features to extract a name core for any given input
  - Training set: manually constructed set of name variant pairs
  - Example: M,u,h,a,m,m,e,d → 1,0,1,0,1,0,0,1 → Mhmd

- Apply trained tree on a record linkage problem
  - Records consisting of 4 names
  - Create name core sequence by repeated application of the model
  - Compare to other name core sequences
A decision tree is a generalization over examples
Some leaf nodes may contain examples with conflicting labels
E.g., Sebastiaan, Bastiaan, Bastjen, LCS bastn, first nucleus in LCS
Define **quality** as the ratio of positive and negative examples

<table>
<thead>
<tr>
<th>pos = 1</th>
<th>syllable part</th>
</tr>
</thead>
<tbody>
<tr>
<td>no</td>
<td>yes</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>onset</th>
<th>nucleus</th>
</tr>
</thead>
<tbody>
<tr>
<td>no</td>
<td>0 (discard)</td>
</tr>
<tr>
<td>yes</td>
<td>1 (keep)</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>adjacent to [r,l]</th>
</tr>
</thead>
<tbody>
<tr>
<td>no</td>
</tr>
<tr>
<td>yes</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>coda</th>
<th>pos &gt; 4</th>
</tr>
</thead>
<tbody>
<tr>
<td>no</td>
<td>0 (discard)</td>
</tr>
<tr>
<td>yes</td>
<td>1 (keep)</td>
</tr>
</tbody>
</table>

Negative: e,i,a,a,i,a,a,e
Positive: a
Quality: 8/9 = 0.88
Leaf node quality

- Standard method: the majority class is assigned to the leaf node
- Quality parameter: set a threshold on the minimum proportion of positive examples
- Example
  - a leaf node containing 33 positive examples and 51 negative examples
  - proportion: 0.39 positive, 0.61 negative
  - standard method: a character that ends up in this leaf node is not included in the name core
  - threshold 0.3: a character that ends up in this leaf node is included in the name core
  - threshold 0.4: a character that ends up in this leaf node is not included in the name core
- Change the threshold to increase or decrease the number of name core matches
Name cores: conclusion

- Novel, morphologically motivated model of name variation
- Computational efficiency and high recall in discovering links with small edit distance
- Additional discovery of a significant amount of links with large edit distance
Family reconstruction

- Record linkage is generally based on pair-wise comparison of records
- Alternatively, multiple records can be linked simultaneously
- The consistency of the link graph can be used as matching criterion
- More information is included in a match decision
- Reduced dependence on string similarity measures
Family reconstruction

- Linkage task: reconstruction of nuclear families based on
  - marriage certificates (3.0 million)
  - birth certificates (4.2 million)

<table>
<thead>
<tr>
<th>Marriage, 1846</th>
<th>Birth, 1846</th>
<th>Birth, 1848</th>
<th>Birth, 1849</th>
<th>Birth, 1850</th>
</tr>
</thead>
<tbody>
<tr>
<td>Jan Vermeulen</td>
<td>Melis Vermeulen</td>
<td>Arnoldus Vermeulen</td>
<td>Jacobus Vermeulen</td>
<td>Johanna Vermeulen</td>
</tr>
<tr>
<td>Aaltje Rijn</td>
<td>Jan Vermeulen</td>
<td>Jan Vermeulen</td>
<td>Jan Vermeulen</td>
<td>Jan Vermeulen</td>
</tr>
<tr>
<td></td>
<td>Aaltje Rijn</td>
<td>Alida Rijn</td>
<td>Aaltje Rijn</td>
<td>Alida Rijn</td>
</tr>
</tbody>
</table>
Family reconstruction

- Every birth certificate can possibly be matched to a marriage certificate with the birth parents as bride and groom

<table>
<thead>
<tr>
<th>Marriage, 1846</th>
<th>Marriage, ≤ 1846</th>
<th>Marriage, ≤ 1848</th>
<th>Marriage, ≤ 1849</th>
<th>Marriage, ≤ 1850</th>
</tr>
</thead>
<tbody>
<tr>
<td>Jan Vermeulen</td>
<td>Jan Vermeulen</td>
<td>Jan Vermeulen</td>
<td>Jan Vermeulen</td>
<td>Jan Vermeulen</td>
</tr>
<tr>
<td>Aaltje Rijn</td>
<td>Aaltje Rijn</td>
<td>Alida Rijn</td>
<td>Aaltje Rijn</td>
<td>Alida Rijn</td>
</tr>
<tr>
<td>Birth, 1846</td>
<td></td>
<td>Birth, 1848</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Melis Vermeulen</td>
<td></td>
<td>Arnoldus Vermeulen</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Jan Vermeulen</td>
<td></td>
<td>Jan Vermeulen</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Aaltje Rijn</td>
<td></td>
<td>Alida Rijn</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Birth, 1849</td>
<td></td>
<td>Birth, 1849</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Jacobus Vermeulen</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Jan Vermeulen</td>
<td></td>
<td>Jan Vermeulen</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Aaltje Rijn</td>
<td></td>
<td>Alida Rijn</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Birth, 1850</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Johanna Vermeulen</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Jan Vermeulen</td>
<td></td>
<td>Jan Vermeulen</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Aaltje Rijn</td>
<td></td>
<td>Alida Rijn</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Family reconstruction

- Links based on exact matching can be assumed **if the event sequence is consistent**

- Consistency constraints: year and place of event

- Examples
  - marriage 1846, birth 1846, birth 1849: exact match on marriage couple and names of parents
  - birth 1848, birth 1850: exact match on names of parents
• Links based on approximate matching can be assumed if the event sequence is consistent

• Alida – Aaltje: approximate match (edit distance = 4, relative 4/5 for Alida and 4/6 for Aaltje)
Implementation analysis

- Storage and comparison of event sequences: linear in the number of records
- 3 million marriage certificates, 4 million birth certificates
- Implementation in C++ on a 3.16 GHz processor with 6GB memory
- Total processing time for full family reconstruction of the Netherlands in the 19th century: 46 minutes
Benchmark

- Comparison with manual family reconstructions from the Dutch town of Coevorden and surroundings
- Algorithm-generated families and benchmark families are matched using exact date and place of marriage
- Children are compared using year of birth
- Example:
  Total family size: 4
  Children matched: 3
## Benchmark results

<table>
<thead>
<tr>
<th>family size</th>
<th>not found</th>
<th>0</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
<th>6</th>
<th>7</th>
<th>8</th>
<th>9</th>
<th>10</th>
<th>11</th>
<th>12</th>
<th>13</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>10</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>1</td>
<td>12</td>
<td>0</td>
<td>104</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>2</td>
<td>13</td>
<td>1</td>
<td>23</td>
<td>109</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>3</td>
<td>8</td>
<td>0</td>
<td>5</td>
<td>16</td>
<td>99</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>4</td>
<td>18</td>
<td>1</td>
<td>8</td>
<td>10</td>
<td>12</td>
<td>84</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>5</td>
<td>13</td>
<td>0</td>
<td>8</td>
<td>3</td>
<td>7</td>
<td>17</td>
<td>74</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>6</td>
<td>11</td>
<td>1</td>
<td>8</td>
<td>2</td>
<td>7</td>
<td>8</td>
<td>22</td>
<td>83</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>7</td>
<td>6</td>
<td>0</td>
<td>6</td>
<td>1</td>
<td>4</td>
<td>3</td>
<td>7</td>
<td>13</td>
<td>55</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>8</td>
<td>7</td>
<td>0</td>
<td>3</td>
<td>4</td>
<td>4</td>
<td>3</td>
<td>1</td>
<td>3</td>
<td>9</td>
<td>35</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>9</td>
<td>9</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>2</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>3</td>
<td>3</td>
<td>22</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>10</td>
<td>4</td>
<td>1</td>
<td>3</td>
<td>2</td>
<td>1</td>
<td>2</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>2</td>
<td>6</td>
<td>14</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>11</td>
<td>3</td>
<td>0</td>
<td>3</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>4</td>
<td></td>
<td></td>
</tr>
<tr>
<td>12</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>2</td>
<td>1</td>
<td>2</td>
<td>0</td>
<td>2</td>
<td>4</td>
<td></td>
<td></td>
</tr>
<tr>
<td>≥13</td>
<td>1</td>
<td>1</td>
<td>2</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>10</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>1</td>
</tr>
</tbody>
</table>

| %           | 74.9      |     | 83.5|      |    |    |    |    |    |    |    |    |    |    |    |    |
Method advantages

• Family reconstruction using collective entity resolution
• Accuracy levels sufficient for practical use
• Computationally efficient
• Reduced dependence on string similarity measures
Method disadvantages

• Domain-specific constraints and thresholds

• Coarse candidate selection strategy
  • Can be improved at the expense of efficiency

• Advantage over traditional methods is small
  • 28% of all matches is non-exact
  • Only few cases with edit distance > 3
Spatio-temporal linkage

• Person names contain variation and errors which complicates record linkage

• Solution: do not use person names

• Search using only place, date (spatio-temporal information) and gender
Spatio-temporal linkage

- Sort certificate data using a key based on gender, birth date and birth place of an individual

- This combination of properties is highly selective for a large part of the Dutch population in the 19th century

- Use names as second sort key

- Consecutive records are likely to refer to the same person
Spatio-temporal linkage

- Key: V|18760830|groningen
- Note: placename sort is based on mapping to geographical coordinates
- Sorted database fragment:
  V|18760830|groningen|birth|Siersma||Martha|Siersma||Johannes|Biel||Catharina
  V|18760830|groningen|birth|Vries|de|Hendrikje|Vries|de|Derk|Bilijam||Grietje
  V|18760830|groningen|marriage|Vries|de|Hendrikje|Nienhuis||Lammert
  V|18760830|groningen|death|Houttuin||Beeka|Houttuin||Wiebe|Sijtsema||Eppoliena
- Birth and marriage of Hendrikje de Vries are automatically placed in consecutive rows after sort using only the first character of the last name in the sorting procedure
Spatio-temporal linkage

- Often an age is provided instead of an exact birth date
- Estimate the birth date by subtracting the age from the event date
- Compensate for lack of precision by using the last name and a first initial in the sorting key
- Sort on gender, year, place, last name, initial, estimated date

- High accuracy for consecutive records
- Records generally close by when not directly adjacent
- Influence of names limited, resulting in discovery of name variation
Spatio-temporal linkage

- WieWasWie web interface allows for spatio-temporal search
Summary

• Indexing character vectors provides all pairs of records within a threshold of Levenshtein edit distance for databases with millions of records using a regular desktop pc in a few hours in batch or real-time in a query application

• Using machine learning a model can be developed to generate core representations of person names which generalize over name variation

• In a collective entity resolution approach domain constraints can be used to confirm or reject a link between two records based on information from related records

• Spatio-temporal information reduces dependency on names

• Any questions?