Web Data Integration

Identity Resolution
The Data Integration Process

- Data Collection
- Schema Mapping
  - Data Translation
- Identity Resolution
- Data Quality Assessment
  - Data Fusion
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
8. References
1. Introduction

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

<p>| | | | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>DB1</td>
<td>CID1243</td>
<td>Chris Miller</td>
<td>12/20/1982</td>
<td>Bardon Street; Melville</td>
</tr>
<tr>
<td>DB2</td>
<td>344278</td>
<td>Christian Miller</td>
<td>2/20/1982</td>
<td>7 Bardon St., Melville</td>
</tr>
<tr>
<td>DB2</td>
<td>427859</td>
<td>Chris Miller</td>
<td>12/14/1973</td>
<td>Bardon St., Madison</td>
</tr>
</tbody>
</table>

- The problem appears whenever
  1. data from multiple sources is combined
  2. a single data source is cleaned (de-duplicated)
- Various commercial tools are available, especially for CRM domain
Example: Records from Multiple Data Sources

Google search for "Iphone 5 smartphone"

- **Apple iPhone 5s Smartphone** - 4G - 16 GB 16 GB - WCDMA (UMTS... €558.99 from 50+ shops
  - **5 stars** 30 product reviews
  - Apple • Smartphone • iOS • 10,2 cm display • 4G LTE • UMTS • GSM • 8 MP Kamera • Quadband

- **Apple iPhone 5s Smartphone** - 4G - 16 GB 16 GB - WCDMA (UMTS... €361.00 from 50+ shops
  - **5 stars** 31 product reviews
  - Apple • Smartphone • iOS • 10,2 cm display • 4G LTE • UMTS • GSM • 8 MP Kamera • Quadband

- **Apple iPhone 5s Smartphone** - 4G - 32 GB 32 GB - WCDMA (UMTS... €629.00 from 50+ shops
  - **5 stars** 30 product reviews
  - Apple • Smartphone • iOS • 10,2 cm display • 4G LTE • UMTS • GSM • 8 MP Kamera • Quadband
Negative Effects of Duplicates in Single Data Source

1. Unnecessary memory and processing power consumption

2. Queries give you wrong results
   - Number of customers != SELECT COUNT(*) FROM customer

3. 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
- Mixed and split citation problem
- Object identification
- Match
- Deduplication
- Reference matching
- Fuzzy match
- Object consolidation
- Entity resolution
- Approximate match
- Entity clustering
- Identity uncertainty
- Merge/purge
- Reference reconciliation
- Hardening soft databases
- Householding
- Doubles
The Two Central Challenges of Identity Resolution

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

- **Solution:** Matching Rules
  - Compare multiple attributes of the records using attribute-specific similarity measures

<table>
<thead>
<tr>
<th>Name</th>
<th>Date</th>
<th>Address</th>
</tr>
</thead>
<tbody>
<tr>
<td>Chris Miller</td>
<td>12/20/1982</td>
<td>Bardon Street; Melville</td>
</tr>
<tr>
<td>Christian Miller</td>
<td>2/20/1982</td>
<td>7 Bardon St., Melville</td>
</tr>
<tr>
<td>Chris Miller</td>
<td>12/14/1973</td>
<td>Bardon St., Madison</td>
</tr>
</tbody>
</table>

- **Questions:**
  1. Which attributes are relevant for the comparison?
  2. What is the right similarity measure for each attribute?
  3. How to combine/weight similarity scores from different attributes?
A Wide Range of Similarity Measures Exists

**Similarity Measures**

- **Edit-based**
  - Damerau-Levenshtein
  - Levenshtein
  - Smith-Waterman
  - Smith-Waterman-Gotoh

- **Token-based**
  - Jaccard
  - Dice
  - Cosine Similarity
  - Monge-Elkan
  - Soft TF-IDF

- **Hybrid**
  - Monge-Elkan
  - Soft TF-IDF

- **Type-specific**
  - Numerical Attributes
  - Dates/Times
  - Geo-Coordinates

- **Phonetic**
  - Soundex
  - Kölner Phonetik
  - Metaphone
  - Double Metaphone

**Numerical Attributes**

- Dates/Times
The Two Central Challenges of Identity Resolution

• **Challenge 2**: Data sets are large
  • Quadratic complexity: Comparing every pair of records is too expensive

• **Solution**: Blocking algorithms
  • 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 sim score between records x and y as a linearly weighted combination of individual attribute sim scores

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

- We declare x and y matched if $sim(x, y) \geq \beta$ for a pre-specified threshold $\beta$, and not matched otherwise.

  - Variation: Human manually reviews pair $(x,y)$ if $\alpha \leq sim(x, y) < \beta$. 
Example Matching Rule

\[
sim(x, y) = 0.3s_{\text{name}}(x, y) + 0.3s_{\text{phone}}(x, y) + 0.1s_{\text{city}}(x, y) + 0.3s_{\text{state}}(x, y)
\]

- \(s_{\text{name}}(x, y)\): using the Jaro-Winkler similarity measure
- \(s_{\text{phone}}(x, y)\): based on edit distance between \(x\)'s phone (after removing area code) and \(y\)'s phone
- \(s_{\text{city}}(x, y)\): based on edit distance
- \(s_{\text{state}}(x, y)\): based on exact match; yes \(\Rightarrow 1\), no \(\Rightarrow 0\)
2.2 More Complex Matching Rules

- Appropriate when we have more specific domain knowledge.

- **Example 1:** Two persons match if names match approximately and either phones match exactly or addresses match exactly
  1. If $s_{name}(x,y) < 0.8$ then return “not matched”
  2. Otherwise if $e_{phone}(x,y) = true$ then return “matched”
  3. Otherwise if $e_{city}(x,y) = true$ and $e_{state}(x,y) = true$ 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 $s_{name}(x,y) > 0.7$ and
  • $\max (s_{genID}(x,y), s_{componentID}(x,y), s_{structureID}(x,y)) = 1$
  • 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
  - Customers: Orders, addresses

- Example: The movie names look quite similar to edit distance measure

<table>
<thead>
<tr>
<th>ID</th>
<th>Film</th>
<th>ID</th>
<th>Actor</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Star Wars 1</td>
<td>1</td>
<td>Ewan McGregor</td>
</tr>
<tr>
<td>2</td>
<td>Star Wars 4</td>
<td>2</td>
<td>Natalie Portman</td>
</tr>
<tr>
<td>3</td>
<td>Star War 1</td>
<td>3</td>
<td>Mark Hamill</td>
</tr>
<tr>
<td>4</td>
<td></td>
<td>4</td>
<td>Harrison Ford</td>
</tr>
<tr>
<td>5</td>
<td></td>
<td>5</td>
<td>Ewan McGregor</td>
</tr>
<tr>
<td>6</td>
<td></td>
<td>6</td>
<td>Natalie Portman</td>
</tr>
</tbody>
</table>
Example: Matching Films

![Graph showing precision vs recall for matching films with and without actors. The graph compares the performance of two methods: one without actors (red line) and one with actors (green line). The graph shows that the method with actors generally has higher precision at various recall levels compared to the method without actors.](image-url)
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
    - Mannheim
    - e.g. using Google Translate API or other translation software

Petrovski, Bryl, Bizer: Integrating Product Data from Websites offering Microdata Markup. DEOS, 2014.
Example: Complex Matching Rule including Preprocessing

http://silk-framework.com
2.5 Local versus Global Matching

- **Input:** A matrix containing entity similarities
- **Output:** A set of correspondences connecting matching entity pairs

- **Local Matching**
  - consider all pairs above threshold as matches
  - implies: One entity can be matched with several other entities
  - makes sense for duplicate detection within single data source

- **Global Matching**
  - enforce constraint that one entity in data set A should only be matched to one entity in data set B
  - makes sense for distinct 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)
Summary: The Entity Matching Process

1. Gather Data for Matching
2. Normalize Attribute Values
3. Apply Attribute-specific Similarity Measures
4. Combine Similarity Scores
5. Decide Match/Non-Match
6. 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 $\rightarrow$ 10,000 comparisons
    - 10,000 customers $\rightarrow$ 100 million comparisons
    - 1,000,000 customers $\rightarrow$ 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 comparisons 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 overlooked 😞
Number of comparisons: All pairs

Complexity: $n^2$

<table>
<thead>
<tr>
<th></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>
<th>14</th>
<th>15</th>
<th>16</th>
<th>17</th>
<th>18</th>
<th>19</th>
<th>20</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</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>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>2</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>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>3</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>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>4</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>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>5</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>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>6</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>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>7</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>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>8</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>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>9</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>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<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>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>11</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>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>12</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>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>13</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>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>14</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>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>15</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>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>16</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>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>17</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>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>18</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>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>19</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>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>20</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>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

20 records ➔ 400 comparisons
**Reflexivity of Similarity**

Complexity: $n^2 - n$

<table>
<thead>
<tr>
<th></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>
<th>14</th>
<th>15</th>
<th>16</th>
<th>17</th>
<th>18</th>
<th>19</th>
<th>20</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</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>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>2</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>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>3</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>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>4</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>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>5</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>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>6</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>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>7</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>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>8</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>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>9</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>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<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>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>11</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>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>12</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>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>13</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>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>14</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>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>15</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>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>16</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>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>17</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>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>18</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>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>19</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>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>20</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>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

**Similarity is reflexive:** \( \text{sim}(x, x) = 1 \)

380 comparisons
Symmetry of Similarity

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

Similarity is symmetric:
\[ \text{sim}(x,y) = \text{sim}(y,x) \]

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 ~1000 time 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

<table>
<thead>
<tr>
<th></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>
<th>14</th>
<th>15</th>
<th>16</th>
<th>17</th>
<th>18</th>
<th>19</th>
<th>20</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</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>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>2</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>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>3</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>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>4</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>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>5</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>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>6</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>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>7</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>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>8</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>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>9</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>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<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>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>11</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>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>12</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>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>13</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>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>14</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>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>15</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>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>16</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>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>17</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>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>18</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>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>19</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>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>20</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>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

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 solution: 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 compared
  - Pairs might not compared as their blocking key values differ
    - typo in zip code, customer has moved
  - Possible solution: Use only first letters as they often contain less typos

- Example combining both tricks

<table>
<thead>
<tr>
<th>FirstName</th>
<th>Name</th>
<th>Adresse</th>
<th>ID</th>
<th>Blocking Key</th>
</tr>
</thead>
<tbody>
<tr>
<td>Sal</td>
<td>Stolpho</td>
<td>123 First St.</td>
<td>456780</td>
<td>STOSAL</td>
</tr>
<tr>
<td>Mauricio</td>
<td>Hernandez</td>
<td>321 Second Ave</td>
<td>123456</td>
<td>HERMAU</td>
</tr>
</tbody>
</table>
### 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

<table>
<thead>
<tr>
<th>First Name</th>
<th>Last Name</th>
<th>Address</th>
<th>SSN</th>
</tr>
</thead>
<tbody>
<tr>
<td>Mauricio</td>
<td>Hernandez</td>
<td>321 Second Ave</td>
<td>123456</td>
</tr>
<tr>
<td>All</td>
<td>Stolpo</td>
<td>123 First St.</td>
<td>456780</td>
</tr>
<tr>
<td>Sal</td>
<td>Stolpo</td>
<td>123 First St.</td>
<td>456780</td>
</tr>
<tr>
<td>Sal</td>
<td>Stelfo</td>
<td>123 First Street</td>
<td>456789</td>
</tr>
</tbody>
</table>

**Key**

<table>
<thead>
<tr>
<th>Key</th>
</tr>
</thead>
<tbody>
<tr>
<td>123HER</td>
</tr>
<tr>
<td>456STO</td>
</tr>
<tr>
<td>456STO</td>
</tr>
<tr>
<td>456STE</td>
</tr>
</tbody>
</table>
The Sorted Neighborhood Method (SNM)

Window size = 4

Complexity:
1. Key generation: $O(n)$
2. Sorting: $O(n \times \log(n))$
3. Comparisons: $O(n \times w)$

54 comparisons

+ no problems with different bucket sizes
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, last name, soundex value of last name

- **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 \( w \)
  2. Merge sets of matches found in each run
    - Less efficient, but much more effective than single-pass
4. Evaluation

- **Ground Truth** for the evaluation: Manually label a set of pairs as matches or non-matches, including corner cases
  - True positives (TP): Correctly discovered duplicates
  - False positives (FP): Incorrectly discovered duplicates
  - True negatives (TN): Correctly avoided pairs
  - False negatives (FN): Missed duplicates

- **Precision** = $\frac{TP}{TP + FP}$
  - discovered correct matches / declared matches
  - fraction of declared matches that are correct

- **Recall** = $\frac{TP}{TP + FN}$
  - discovered correct matches / all matches
  - fraction of all correct matches that are found
Precision & Recall

**True matches**
- True positives
- False negatives
- False positives
- True negatives

**Declared matches**

**All pairs**

**Precision** = \( \frac{\text{True positives}}{\text{Declared matches}} \)

**Recall** = \( \frac{\text{True positives}}{\text{True matches}} \)

**F-Measure** = \( \frac{2 \cdot \text{Precision} \cdot \text{Recall}}{\text{Precision} + \text{Recall}} \)

**Accuracy** is not a good measure as true negatives usually dominate overall result.
F-Measure versus Average of Precision & Recall

Harmonic mean („F-Measure“) emphasizes the importance of small values, whereas arithmetic mean („Average“) is affected more by outliers that are unusually large.

\[
z = \frac{1}{2} (x + y)
\]

\[
z = \frac{2 \cdot (x \cdot y)}{x + y}
\]
F-Measure Graph

Optimal Threshold
Efficiency Measures

- Besides of the quality of the matching rule, the quality of the blocking method is also important.

- Solution 1: Runtime measurements
  - but: Different hardware, difficult repeatability

- Solution 2: Measure how well/poor the blocking method filters the candidates
  - how many true positives are missed?

- Reduction Ratio = \[ \frac{\text{pairs}_{\text{without Blocking}}}{\text{pairs}_{\text{after Blocking}}} \]

- Pairs Completeness = \[ \frac{\text{TP}}{\text{FN} + \text{TP}} = \frac{\text{matches}_{\text{after Blocking}}}{\text{all matches}} \]

- Pairs Quality = \[ \frac{\text{TP}}{\text{TP} + \text{TN}} = \frac{\text{matches}_{\text{after Blocking}}}{\text{all pairs}_{\text{selected by Blocking}}} \]
Evaluating Identity Resolution

Precision  \rightarrow  Similarity threshold  \rightarrow  Recall

Similarity measure  \rightarrow  Efficiency  \rightarrow  Partition/window size
Standard 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

<table>
<thead>
<tr>
<th>Match task</th>
<th>Source size (#entities)</th>
<th>Mapping size (#correspondences)</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Source 1</td>
<td>Source 2</td>
</tr>
<tr>
<td>Bibliographic</td>
<td>DBLP-ACM</td>
<td>2,616</td>
</tr>
<tr>
<td></td>
<td>DBLP-Scholar</td>
<td>64,263</td>
</tr>
<tr>
<td>E-commerce</td>
<td>Amazon-GoogelProducts</td>
<td>1,363</td>
</tr>
<tr>
<td></td>
<td>Abt-Buy</td>
<td>1,081</td>
</tr>
</tbody>
</table>


2. Further datasets: CORA (bibliographic), restaurants, CDs, movies

F-Measure for Bibliographic and E-Commerce Data

5. Similarity Measures – In Detail

- **Edit-based**
  - Damerau-Levenshtein
  - Levenshtein
  - Smith-Waterman
  - Smith-Waterman-Gotoh

- **Token-based**
  - Jaccard
  - Dice
  - Cosine Similarity
  - Monge-Elkan
  - Soft TF-IDF

- **Hybrid**
  - Jaro-Winkler
  - Words / n-grams
  - Soft TF-IDF

- **Type-specific**
  - Numerical Attributes
  - Dates/Times
  - Geo-Coordinates

- **Phonetic**
  - Soundex
  - Kölner Phonetik
  - Metaphone
  - Double Metaphone
Similarity Measures within the Entity Matching Process

1. Gather Data for Matching
2. Normalize Attribute Values
3. Apply Attribute-specific Similarity Measures
4. Combine Similarity Scores
5. Decide Match/Non-Match
6. Cluster Records based on Correspondences

But do not forget about the importance of the first two steps!
Similarity and Distance Measures

- Similarity is a rather universal but vague concept: \( \text{sim}(x,y) \)
  - \( x \) and \( y \) can be strings, numbers, geo coordinates, records, images, songs, ...

- Normalized: \( \text{sim}(x,y) \in [0,1] \)
  - \( \text{sim}(x,y) = 1 \) for exact match
  - \( \text{sim}(x,y) = 0 \) for "completely different" \( x \) and \( y \)
  - \( 0 < \text{sim}(x,y) < 1 \) for some approximate similarity

- Distance measures
  - Reflexive: \( \text{dist}(x,x) = 0 \)
  - Positive: \( \text{dist}(x,y) \geq 0 \)
  - Symmetric: \( \text{dist}(x,y) = \text{dist}(y,x) \)
  - Triangular inequality: \( \text{dist}(x,z) \leq \text{dist}(x,y) + \text{dist}(y,z) \)

- Converting distances to similarities
  - \( \text{sim}(x,y) = 1 - \text{dist}(x,y) \) if \( \text{dist}(x,y) \in [0,1] \)
  - \( \text{sim}(x,y) = 1/(\text{dist}(x,y)+1) \) if \( \text{dist}(x,y) \in [0,\infty] \)
5.1 Edit-based String Similarity Measures

- **Edit-based**
  - Damerau-Levenshtein
  - Levenshtein
  - Smith-Waterman
  - Smith-Waterman-Gotoh

- **Type-specific**
  - Numerical attributes
  - Dates/Times

- **Similarity Measures**
  - Jaccard
  - Dice
  - Cosine Similarity
  - Monge-Elkan
  - Soft TF-IDF

- **Token-based**
  - Words/n-grams

- **Phonetic**
  - Soundex
  - Kölner Phonetik
  - Metaphone
  - Double Metaphone

- **Hybrid**
  - Smith-Waterman-Gotoh
  - Hamming
  - Jaro-Winkler
  - Jaro
  - Damerau-Levenshtein
  - Levenshtein
  - Monge-Elkan
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:**
  - `levenshtein('Table', 'Cable') = 1` (1 Substitution)
  - `levenshtein('Table', 'able') = 1` (1 Deletion)
- Levenshtein distance is often called "**edit distance**"
  - as it is the most widely used edit-based measure
Levenshtein Similarity

\[ sim_{Levenshtein} = 1 - \frac{Levenshtein\text{ }Distance}{\max(\left| s_1 \right|, \left| s_2 \right|)} \]

<table>
<thead>
<tr>
<th>(s_1)</th>
<th>(s_2)</th>
<th>Levenshtein Distance</th>
<th>(sim_{Levenshtein})</th>
</tr>
</thead>
<tbody>
<tr>
<td>Jones</td>
<td>Johnson</td>
<td>4</td>
<td>0.43</td>
</tr>
<tr>
<td>Paul</td>
<td>Pual</td>
<td>2</td>
<td>0.5</td>
</tr>
<tr>
<td>Paul Jones</td>
<td>Jones, Paul</td>
<td>11</td>
<td>0</td>
</tr>
</tbody>
</table>
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 complexity 😞
  - Levenshtein distance is calculated using dynamic programming
  - time 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, town 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

\[ \text{sim}_{jaro} = \frac{1}{3} \left( \frac{m}{|x|} + \frac{m}{|y|} + \frac{m - t}{m} \right) \]

\( m = 4 \quad t = 1 \)

\[ \text{sim}_{jaro} = \frac{1}{3} \cdot \left( \frac{4}{4} + \frac{4}{4} + \frac{4 - 1}{4} \right) \approx 0.92 \]

\( m = 4 \quad t = 0 \)

\[ \text{sim}_{jaro} = \frac{1}{3} \cdot \left( \frac{4}{5} + \frac{4}{7} + \frac{4 - 0}{4} \right) \approx 0.79 \]
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
  
  \[
  \begin{align*}
  &\text{if } \text{sim}_{\text{jaro}} \leq 0.7 : \text{sim}_{\text{jarowinkler}} = \text{sim}_{\text{jaro}} \\
  \text{otherwise} : \quad \text{sim}_{\text{jarowinkler}} = \text{sim}_{\text{jaro}} + l \cdot p \cdot (1 - \text{sim}_{\text{jaro}})
  \end{align*}
  \]

- \(l\) : 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:

  \[
  \begin{align*}
  s_1 &= PAUL \quad s_2 = PUAL \\
  \text{sim}_{\text{jaro}} &= 0.92 \\
  l &= 1 \\
  p &= 0.1 \\
  \text{sim}_{\text{jarowinkler}} &= 0.92 + 1 \cdot 0.1 \cdot (1 - 0.92) = 0.928
  \end{align*}
  \]

  \[
  \begin{align*}
  s_1 &= JONES \quad s_2 = JOHNSON \\
  \text{sim}_{\text{jaro}} &= 0.79 \\
  l &= 2 \\
  p &= 0.1 \\
  \text{sim}_{\text{jarowinkler}} &= 0.79 + 2 \cdot 0.1 \cdot (1 - 0.79) = 0.832
  \end{align*}
  \]
5.2 Token-based String Similarity Measures

- **Edit-based**
  - Damerau-Levenshtein
  - Levenshtein
  - Smith-Waterman
  - Smith-Waterman-Gotoh

- **Token-based**
  - Jaccard
  - Dice
  - Words / n-grams
  - Cosine Similarity
  - Monge-Elkan
  - Soft TF-IDF

- **Type-specific**
  - Numerical attributes
  - Dates/Times

- **Phonetic**
  - Soundex
  - Kölner Phonetik
  - Metaphone
  - Double Metaphone

- **Hybrid**
  - Jaro-Winkler
  - Monge-Elkan

- **Similarity Measures**
Token-based Similarity

Token-based measures can deal with the different order of words in longer 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’
- 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 $n$
  - Sliding window over string
  - $n=2$: Bigrams
  - $n=3$: Trigrams
  - Variation: Pad with $n-1$ special characters
    - Emphasizes beginning and end of string
  - Variation: Include positional information to weight similarities

- Goals:
  1. Deal with typos and different order of words
  2. Reduce the time complexity compared to Levenshtein

<table>
<thead>
<tr>
<th>String</th>
<th>Bigrams</th>
<th>Padded bigrams</th>
<th>Positional bigrams</th>
<th>Trigrams</th>
</tr>
</thead>
<tbody>
<tr>
<td>gail</td>
<td>ga, ai, il</td>
<td>g, ga, ai, il, l</td>
<td>(ga,1), (ai,2), (il,3)</td>
<td>gai, ail</td>
</tr>
<tr>
<td>gayle</td>
<td>ga, ay, yl, le</td>
<td>g, ga, ay, yl, le</td>
<td>(ga,1), (ay,2), (yl,3), (le,4)</td>
<td>gay, ayl, yle</td>
</tr>
<tr>
<td>peter</td>
<td>pe, et, te, er</td>
<td>p, pe, et, te, er</td>
<td>(pe,1), (et,2), (te,3), (er,4)</td>
<td>pet, ete, ter</td>
</tr>
<tr>
<td>pedro</td>
<td>pe, ed, dr, ro</td>
<td>p, pe, ed, dr, ro</td>
<td>(pe,1), (ed,2), (dr,3), (ro,4)</td>
<td>ped, edr, dro</td>
</tr>
</tbody>
</table>
Token-based Similarity Measures

- Can be applied to words or n-grams

- **Overlap Coefficient:**\( sim_{\text{overlap}}(x, y) = \frac{|tok(x) \cap tok(y)|}{\min(|tok(x)|, |tok(y)|)} \)
  - example: Useful for attribute label matching where attribute labels might contain units of measurements or years

- **Jaccard Coefficient:**
  \[
  sim_{\text{jaccard}}(x, y) = \frac{|tok(x) \cap tok(y)|}{|tok(x)| + |tok(y)| - |tok(x) \cap tok(y)|} = \frac{|tok(x) \cap tok(y)|}{|tok(x) \cup tok(y)|}
  \]
  - widely used general purpose similarity measure for tokens

- **Dice's Coefficient:**\( sim_{\text{dice}}(x, y) = \frac{2 \cdot |tok(x) \cap tok(y)|}{|tok(x)| + |tok(y)|} \)
5.3 Hybrid String Similarity Measures

- **Edit-based**
  - Damerau-Levenshtein
  - Levenshtein
  - Smith-Waterman
  - Smith-Waterman-Gotoh

- **Token-based**
  - Jaccard
  - Dice
  - Cosine Similarity

- **Type-specific**
  - Numerical attributes
  - Dates/Times
  - Geo-Coordinates

- **Phonetic**
  - Soundex
  - Kölner Phonetik
  - Metaphone
  - Double Metaphone

- **Hybrid**
  - Monge-Elkan
  - Soft TF-IDF
Monge-Elkan Similarity

- Hybrid similarity measures split strings into tokens and apply internal similarity function to compare tokens
  - Goal: Find best match for each token

- Can deal with typos and different order of words

- \[ \text{sim}_{MongeElkan}(x, y) = \frac{1}{|x|} \sum_{i=1}^{|x|} \max_{j=1,|y|} \text{sim}'(x[i], y[j]) \]
  - \(|x|\) is number of tokens in \(x\)
  - \(\text{sim}'\) is internal similarity function (e.g. Levenshtein or Jaro)

- If strings contain just one token each
  - \(\text{sim}_{MongeElkan}(x, y) = \text{sim}'(x, y)\)

- Complexity: Quadratic in number of tokens 😞
Monge-Elkan – Example

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

- Peter Christen vs. Christian Pedro
  - \( sim_{jaro}(peter, christian) = 0.3741 \)
  - \( sim_{jaro}(peter, pedro) = 0.7333 \)
  - \( sim_{jaro}(christen, christian) = 0.8843 \)
  - \( sim_{jaro}(christen, pedro) = 0.4417 \)

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

**Standard Jaccard**

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

- if strings contain multiple words, choose words as tokens

**Extended Jaccard**

- use internal similarity function (e.g. Levenshtein or Jaro) to calculate similarity between all pairs of tokens
- consider tokens as shared if similarity is above threshold
- shared tokens: \( S = \{(x_i, y_j) | x_i \in tok(x) \land y_j \in tok(y) : sim'(x_i, y_j) \geq \theta \} \)
- unique tokens: \( U_{tok(x)} = \{x_i | x_i \in tok(x) \land y_j \in tok(y) \land (x_i, y_j) \notin S \} \)

\[ sim_{jaccard\_ext}(x, y) = \frac{|S|}{|S| + |U_{tok(x)}| + |U_{tok(y)}|} \]
5.4 Phonetic String Similarity Measures

- **Edit-based**
  - Damerau-Levenshtein
  - Levenshtein
  - Smith-Waterman
  - Smith-Waterman-Gotoh

- **Token-based**
  - Jaccard
  - Dice
  - Cosine Similarity
  - Monge-Elkan
  - Soft TF-IDF

- **Hybrid**
  - Monge-Elkan
  - Soft TF-IDF

- **Type-specific**
  - Numerical attributes
  - Dates/Times
  - Geo-Coordinates

- **Phonetic**
  - Soundex
  - Kölner Phonetik
  - Metaphone
  - Double Metaphone
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.

<table>
<thead>
<tr>
<th>Digit</th>
<th>Letters</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>B, F, P, V</td>
</tr>
<tr>
<td>2</td>
<td>C, G, J, K, Q, S, X, Z</td>
</tr>
<tr>
<td>3</td>
<td>D, T</td>
</tr>
<tr>
<td>4</td>
<td>L</td>
</tr>
<tr>
<td>5</td>
<td>M, N</td>
</tr>
<tr>
<td>6</td>
<td>R</td>
</tr>
</tbody>
</table>

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

J525 also: Jenkins, Jansen, Jameson
Soundex on WolframAlpha

Input interpretation:

Soundex | Levenshtein

Soundex code:

L152

Soundex-close English words:

Livingstone | lebensraum | Livingston | lovemaking

Computed by Wolfram Mathematica
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

<table>
<thead>
<tr>
<th>Letter</th>
<th>Context</th>
<th>Code</th>
</tr>
</thead>
<tbody>
<tr>
<td>A, E, I, J, O, U, Y</td>
<td>0</td>
<td></td>
</tr>
<tr>
<td>H</td>
<td>-</td>
<td></td>
</tr>
<tr>
<td>B</td>
<td>1</td>
<td></td>
</tr>
<tr>
<td>P</td>
<td>not before H</td>
<td></td>
</tr>
<tr>
<td>D, T</td>
<td>not before C, S, Z</td>
<td></td>
</tr>
<tr>
<td>F, V, W</td>
<td>3</td>
<td></td>
</tr>
<tr>
<td>P</td>
<td>before H</td>
<td></td>
</tr>
<tr>
<td>G, K, Q</td>
<td></td>
<td></td>
</tr>
<tr>
<td>C</td>
<td>in the initial sound before A, H, K, L, O, Q, R, U, X</td>
<td></td>
</tr>
<tr>
<td></td>
<td>before A, H, K, O, Q, U, X but not after S, Z</td>
<td></td>
</tr>
<tr>
<td>X</td>
<td>not after C, K, Q</td>
<td></td>
</tr>
<tr>
<td>L</td>
<td>5</td>
<td></td>
</tr>
<tr>
<td>M, N</td>
<td>6</td>
<td></td>
</tr>
<tr>
<td>R</td>
<td>7</td>
<td></td>
</tr>
<tr>
<td>S, Z</td>
<td></td>
<td></td>
</tr>
<tr>
<td>C</td>
<td>after S, Z</td>
<td></td>
</tr>
<tr>
<td></td>
<td>in the initial sound, but not before A, H, K, L, O, Q, R, U, X</td>
<td></td>
</tr>
<tr>
<td></td>
<td>not before A, H, K, O, Q, U, X</td>
<td></td>
</tr>
<tr>
<td>D, T</td>
<td>before C, S, Z</td>
<td></td>
</tr>
<tr>
<td>X</td>
<td>after C, K, Q</td>
<td></td>
</tr>
</tbody>
</table>
5.5 Data Type Specific Similarity Measures

- **Type-specific**
  - Numerical attributes
  - Dates/Times

- **Edit-based**
  - Damerau-Levenshtein
  - Levenshtein
  - Smith-Waterman
  - Smith-Waterman-Gotoh
  - Hamming
  - Jaro
  - Jaro-Winkler

- **Token-based**
  - Words/n-grams
  - Jaccard
  - Dice

- **Phonetic**
  - Soft TF-IDF
  - Monge-Elkan
  - Cosine Similarity
  - Metaphone
  - Double Metaphone

- **Hybrid**
  - Smith-Waterman-Gotoh
  - Double Metaphone
  - Soundex
  - Kölner Phonetik
  - Geo-Coordinates
**Numerical Comparison**

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

- \( \text{sim}_\text{num.abs}(n, m) = \begin{cases} 1 - \left( \frac{|n-m|}{d_{\text{max}}} \right) & \text{if } |n - m| < d_{\text{max}} \\ 0 & \text{else} \end{cases} \)
  
  - Linear extrapolation between 0 and \( d_{\text{max}} \)
  - \( d_{\text{max}} = \) maximal numeric distance in which numbers should be considered similar

- **Example:**
  - \( d_{\text{max}} = $1,000 \)
  - \( \text{sim}_\text{num.abs}(2,000, 2,500) = 1 - \frac{500}{1,000} = 0.5 \)
  - \( \text{sim}_\text{num.abs}(200,000, 200,500) = 1 - \frac{500}{1,000} = 0.5 \)

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

- \( \text{sim}_\text{num.perc}(n, m) = \begin{cases} 1 - \left( \frac{pc}{pc_{\text{max}}} \right) & \text{if } pc < pc_{\text{max}} \\ 0 & \text{else} \end{cases} \)
  
  - \( pc = \frac{|n-m|}{\text{max}(|n|, |m|)} \cdot 100 \) is percentage difference
  - \( pc_{\text{max}} = 33\% \) is the maximal percentage that should be tolerated
  - \( \text{sim}_\text{num.perc}(2,000, 2,500) = 1 - \frac{20}{33} = 0.394 \) because \( pc = \frac{|2,000-2,500|}{2,500} \cdot 100 = 20\% \)
  - \( \text{sim}_\text{num.perc}(200,000, 200,500) = 1 - \frac{0.25}{33} = 0.993 \) because \( pc = \frac{500}{200,500} \cdot 100 = 0.25\% \)
Time and Space Comparisons

- **Dates**
  - convert dates into days after year 0 \( \rightarrow \) integer
  - afterwards use \( \text{sim}_{\text{num\_abs}} \)

- **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
5.6 Implementations of Similarity Measures

- SecondString Library
  - Java package that we will use in the exercise
  - supports all basic string comparisons
  - MongeElkan, SoftTFIDF
  - [http://sourceforge.net/projects/secondstring/](http://sourceforge.net/projects/secondstring/)

- SimMetrics Library
  - alternative Java package
  - supports all basic string comparisons
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 rule 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
  - feature value = similarity score
  - if you want the learning algorithm to decide which similarity metric fits best for a specific attribute pair, you generate multiple features for the pair
    - Levenshtein($x$.name, $y$.name)
    - Jaro($x$.name, $y$.name)
    - Jaro-Winkler($x$.name, $y$.name)
Example: Feature Generation

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

- \(s_1\) and \(s_2\) use Jaro-Winkler and 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 \((v_i, c_i)\)
   - \(v_i = f_1(x_i, y_i), \ldots, f_m(x_i, y_i)\) is a feature vector that encodes \((x_i, y_i)\)
in terms of the features (list of similarity values)
   - \(c_i\) is an appropriately transformed version of label \(l_i\)
     (e.g., yes/no or 1/0, depending on learning algorithm used afterwards)
   - thus \(T\) is transformed into \(T' = \{(v_1, c_1), \ldots, (v_n, c_n)\}\)

2. apply a learning algorithm to \(T'\) to learn a matching model \(M\)
   - e.g. linear regression, SVMs, Naïve Bayes, decision trees, …

   - Training data should
     1. be balanced: Contain same amount of matches and non-matches
     2. contain corner cases as they are most informative
       - Star Wars 1 vs. Star Wars 2, Mannheim vs. Ludwigshafen
   - Avoid overfitting by not testing on training data
     - using split validation or cross-validation
Example: Learning a Linearly Weighted Matching Rule

- Goal: Learn rule $\text{sim}(a, b) = \sum_{i=1}^{6} \alpha_i \ast s_i(a, b)$
- 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 $\alpha_i$ that minimize the squared error

  $$\sum_{i=1}^{3} (c_i - \sum_{j=1}^{6} \alpha_j \ast s_j(v_i))^2$$
Example: Learning a Decision Tree

Advantage:
The decision tree learning algorithm automatically selects the features that are useful
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 \(\Rightarrow\) labor intensive and limits the number of features we 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 number 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: \(F_1 > 0.95\) after labeling less 10-50 pairs

Learning Linkage Rules within the Silk Workbench

1. Labeling Pairs of Movies

<table>
<thead>
<tr>
<th>Source: DBpedia</th>
<th>Target: linkedmdb</th>
<th>Score</th>
<th>Correct?</th>
</tr>
</thead>
<tbody>
<tr>
<td><a href="http://dbpedia.org/resource/Topaz_281969_film%29">http://dbpedia.org/resource/Topaz_281969_film%29</a></td>
<td>mdb.org/resource/film/230</td>
<td>-4.1%</td>
<td></td>
</tr>
<tr>
<td><a href="http://purl.org/dc/terms/title%3E">http://purl.org/dc/terms/title&gt;</a> Topaz</td>
<td>1945</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

2. Learned Linkage Rule

http://silk-framework.com
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 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.
References

- All Aspects of Identity Resolution
  - Peter Christen: Data Matching. Springer 2012.

- Blocking
References

- **Similarity Measures**

- **Matching Systems and their Evaluation**

- **Learning Matching Rules**