Web Data Integration

Identity Resolution
The Data Integration Process

1. Data Collection / Extraction
2. Schema Mapping
   - Data Translation
3. Identity Resolution
4. 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
1. Introduction

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

- The problem appears whenever
  1. a single data source is cleaned (deduplicated)
  2. data from multiple sources is integrated
Negative Effects of Duplicates in Single Data Source

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

- Duplicate detection
- Deduplication
- Entity resolution
- Identity uncertainty
- Hardening soft databases

- Record linkage
- Reference matching
- Doubles
- Fuzzy match
- Approximate match
- Merge/purge

- Data matching
- Object identification
- Householding
- Object consolidation
- Match
- Reference reconciliation

Mixed and split citation problem
The Two Central Challenges of Identity Resolution

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

<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>

- **Solution:** Entity Matching
  - compare multiple attributes of the records using attribute-specific similarity measures

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

We will discuss them later …

- **Edit-based**
  - Hamming
  - Levenshtein
  - Jaro
  - Jaro-Winkler

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

- **Datatype-specific**
  - Numbers
  - Dates/Times
  - Geo-Coordinates

- **Sets of Entities**
  - fastText
  - BERT

- **Embedding-based**
  - Monge-Elkan
  - Soft TF-IDF

- **Phonetic**
  - Soundex
  - Kölner Phonetik

- **Hybrid**
  - Jaccard
The Two Central Challenges of Identity Resolution

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

**Challenge 1:** Representations of the same real-world entity are not identical
2.1 Linearly Weighted Matching Rules

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

- 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 Non-Linear Matching Rules

- Often better than linear rules, but require specific domain knowledge.

- **Example 1:** Two persons match if names match approximately and either phones match exactly or addresses match exactly
  1. if $\text{sim}_{\text{name}}(x,y) < 0.8$ then return “not matched”
  2. otherwise if $\text{equal}_{\text{phone}}(x,y) = \text{true}$ then return “matched”
  3. otherwise if $\text{equal}_{\text{city}}(x,y) = \text{true}$ and $\text{equal}_{\text{state}}(x,y) = \text{true}$ then return “matched”
  4. otherwise return “not matched”

- **Example 2:** Two genes match if their names match approximately and any of the different, alternative gene identifiers match exactly (deals with missing values)
  - if $\max(\text{equal}_{\text{genID}}(x,y), \text{equal}_{\text{componentID}}(x,y), \text{equal}_{\text{structureID}}(x,y)) = 1$
  - and $\text{sim}_{\text{name}}(x,y) > 0.7$
  - then return “matched”
2.3 Data Gathering for Matching

- Not only values of the records to be compared, but also values of related records are relevant for the similarity computation
  - Movies: Actors
  - CDs: Songs
  - Persons: Spouse, children, employer, publications

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

<table>
<thead>
<tr>
<th>ID</th>
<th>Film</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Star Wars 1</td>
</tr>
<tr>
<td>2</td>
<td>Star Wars 4</td>
</tr>
<tr>
<td>3</td>
<td>Star War 1</td>
</tr>
</tbody>
</table>

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

![Graph comparing precision and recall for matching films with and without actors. The red line represents 'without actors' and the green line represents 'with actors'.]
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:
    - Dssdn# dfErrn# Dl# F<9; 2D# 419qfk# dsxws
    - Dssdn# dfErrn# Dl# 40q/# lwhe# Fruh# 8 # 193JK }/# E/# lrrq# 31:

- Translation using external services
  - Geocoding
    - translate addresses into geo-coordinates and compare coordinates afterwards
    - e.g., using Google Geocoding API
  - Translate into target language
    - מנהיים ➔ Mannheim
    - e.g., using Google Translate API or other translation software
Example: Complex Matching Rule including Preprocessing
2.5 Local versus Global Matching

- Input: A matrix containing record similarities
- Output: A set of correspondences connecting pairs of matching records

- Local Matching
  - consider all pairs above threshold as matches
  - implies that one record can be matched with several other records
  - makes sense for duplicate detection within single data source

- Global Matching
  - enforce constraint that one record in data set A should only be matched to one record in data set B
  - makes sense for data sources that do not contain duplicates
  - Approaches:
    1. Bipartite pairs with the maximal sum of similarity values
    2. Stable marriage (see Chapter Schema Mapping)
2.6 Cluster Records using Pairwise Correspondences

- **Goal:** Create groups of records describing the same real-world entity from pairwise correspondences
  - relevant for matching multiple data sources and for the deduplication of a single source

- **Simple Approach:** Connected Components
  - transitive closure of pairwise correspondences
  - problem: correspondences might be inconsistent as they result from separate local decisions

- **Smarter Approach:** Correlation Clustering
  - cuts graph into coherent groups by minimizing disagreement with pairwise correspondences
    - Cohesion penalty: Non-matching records in cluster
    - Correlation penalty: Removing correspondences
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 pairs of records that are compared by
  1. avoiding *unnecessary comparisons* (next 3 slides)
    - no negative effect, but faster 😊
  2. applying *blocking methods* that further reduce the number of comparisons
    - negative effect: True matches might be missed 😞
Number of comparisons: All pairs

Complexity: $n^2$

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>

- Applies to duplicate detection use case
- but not to two data sources use case
### Symmetry of Similarity

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

<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>

Still quadratic 😞

190 comparisons
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
    - $\Rightarrow$ 495,000 comparisons instead of 49,995,000
    - algorithm $\sim$100 times faster
    - matches with wrong zip code might be missed
  - partition books by publisher
  - partition people by first two characters of surname

- Blocking is also called hashing or partitioning
Standard Blocking

32 comparisons

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

p xfk#dvwhu#
wkdq#4<3
frp sdulvrqv

0 p ljkwp lvv
P dwfkhv 😞
Choosing a Good Blocking Key

- **Reduction ratio** depends on effectiveness of blocking key
  - high: if records are equally distributed over buckets
  - low: if majority of the records end up in one bucket
    - example: 90% of all customers are from Mannheim
  - possible workaround: build sub-buckets using a second blocking attribute
    - block houses by zip first. Afterward, block within each bucket by street name

- **Recall** depends on actually matching pairs being kept (compared)
  - pairs might not compared as their blocking key values differ
    - typo in zip code, customer has moved
  - possible workaround: use only first letters as they often contain less typos

- Example combining both workarounds

<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>FirstName</th>
<th>Surname</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>Stolpho</td>
<td>123 First St.</td>
<td>456780</td>
</tr>
<tr>
<td>Sal</td>
<td>Stolpho</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>

<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>
### Complexity:

1. Key generation: $O(n)$
2. Sorting: $O(n \times \log(n))$
3. Comparisons: $O(n \times w)$
Challenges when Applying the SNM

- **Choice of Blocking Key**
  - SNM assumes that records that are likely to match fall within the window
  - Thus, key should be strongly "discriminative" and bring together records that are likely to match, and pushes apart records that are not
    - example keys: social sec, student ID, two characters of first + surname

- **Choice of Window Size**
  - Depends on the types and frequency of the errors/typos in the data
  - Practical experience: w = 20 often a good compromise

- **Workaround: Use Multi-Pass Approach**
  1. Run SNM several times with different blocking keys
    - use simple keys and a small w, e.g. 1. social sec, 2. two characters first + surname
  2. Merge sets of matches found in each run
    - Less efficient, but much more effective than single-pass
3.3 Token Blocking for Textual Attributes

- Identifying attributes are often rather textual, e.g.
  - Product names: Samsung Galaxy S10 SM-G975, 128GB, 8GB RAM
  - Names of local business: Wong Restaurant, Hoy Wong Greenwich

- Token Blocking
  - builds an inverted index that associates every token with all entities containing it in their attribute values
  - using only the identifying attribute or a concatenation of multiple attributes
  - afterwards, all pairs that sharing at least one (or more) tokens are compared

- N-Gram Blocking
  - variation of token blocking that uses character n-grams in order to deal with typos
  - n=3: men, end, edo, ...

\[
\begin{array}{|c|c|c|}
\hline
\text{Terms in Y} & \text{ID Lists} \\
\hline
\text{area} & 5 \\
\text{lake} & 4, 6 \\
\text{mendota} & 6 \\
\text{monona} & 4, 5, 6 \\
\text{research} & 5 \\
\text{university} & 4 \\
\hline
\end{array}
\]
4. Evaluation

- You need **ground truth (gold standard)** for the evaluation
- To create a gold standard, manually label a set of record pairs as **matches** or **non-matches** including **corner cases**
- Rule of thumb for creating a **suitable** gold standard with **acceptable** manual effort:
  1. match records using several simple matching techniques (similar to multi-pass blocking) and sort record pairs according to their similarity
  2. use existing information about matches (e.g. ISBN or GTIN numbers that exist in multiple sources)
  3. **manually** verify a fair amount of the resulting pairs (e.g. 500 pairs) including
     1. matching record pairs (randomly chosen, 20% of GS)
     2. corner cases (30% of GS)
     3. non-matching record pairs (randomly chosen, 50% of GS)
Evaluation Metrics: Precision, Recall & F1

Accuracy is not a good metric as true negatives usually dominate overall result.
F1-Measure Graph
Gold Standard Pairs versus All Pairs

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

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

- Option 1: Runtime measurements
  - but: different hardware, replicability problematic

- Option 2: Measure how well/poor the blocking method filters the candidate pairs
  - by which ratio does the blocking method reduce the number of comparisons?
  - how many true positives are missed?

- Reduction Ratio = \(1 - \frac{\text{pairs}_{\text{after Blocking}}}{\text{pairs}_{\text{before Blocking}}}\)

- Pairs Completeness = \(\frac{\text{matches}_{\text{after Blocking}}}{\text{matches}_{\text{before Blocking}}}\)

- Pairs Quality = \(\frac{\text{matches}_{\text{after Blocking}}}{\text{all pairs}_{\text{selected by Blocking}}}\)
Evaluating Identity Resolution

Precision  Similarity threshold  Recall

Similarity measure  Efficiency  Partition/window size
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


2. Magellan/Deepmatcher Evaluation Datasets

   - https://github.com/anhaidgroup/deepmatcher/blob/master/Datasets.md

3. WDC Training Dataset and Gold Standard for Large-Scale Product Matching

   - http://webdatacommons.org/largescaleproductcorpus/
F-Measure for Bibliographic and E-Commerce Data

Outline – Identity Resolution Part 2

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

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 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, images, songs, persons, ...

- 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 \( y \)

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

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

- Measures the dissimilarity of two strings
- Measures the **minimum number of edits** needed to transform one string into the other

- Allowed edit operations:
  1. **insert** a character into the string
  2. **delete** a character from the string
  3. **replace** one character with a different character

- Examples:
  - `levensthein('table', 'cable') = 1` (1 substitution)
  - `levensthein('Chris Bizer', 'Bizer, Chris') = 11` (10 substitution, 1 deletion)

- Levenshtein distance is often called **“edit distance”**
  - as it is the most widely used edit-based measure
**Levenshtein Similarity**

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

<table>
<thead>
<tr>
<th>$s_1$</th>
<th>$s_2$</th>
<th>Levenshtein Distance</th>
<th>$sim_{\text{Levenshtein}}$</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 runtime complexity 😞
  - Levenshtein distance is calculated using dynamic programming
  - runtime complexity $O(|x| \cdot |y|)$
Jaro Similarity

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

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

2. Look for swapped adjacent characters
   - transposition: ‘pe’ vs. ‘ep’
   - \( t \) : number of transpositions

\[
sim_{jaro} = \frac{1}{3} \left( \frac{m}{|x|} + \frac{m}{|y|} + \frac{m - t}{m} \right)
\]
Jaro Similarity – Example

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

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

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

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

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

\(s_1\): PAUL

\(s_2\): PAUL

\(s_1\): JONES

\(s_2\): JOHNSON
Winkler Similarity

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

- Let $p$ be the length of the common prefix of $x$ and $y$.

$$sim_{winkler}(x, y) = sim_{jaro}(x, y) + (1 - sim_{jaro}(x, y)) \frac{p}{10}$$
  - = 1 if common prefix is $\geq 10$
Jaro-Winkler Similarity

- Extension of Jaro similarity considering a common prefix

\[
\text{if } \text{sim}_{\text{jaro}} \leq 0.7 : \text{sim}_{\text{jarowinkler}} = \text{sim}_{\text{jaro}} \\
\text{otherwise : } \text{sim}_{\text{jarowinkler}} = \text{sim}_{\text{jaro}} + l \cdot p \cdot (1 - \text{sim}_{\text{jaro}})
\]

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

- Examples:

\[
\begin{align*}
    s_1 &= \text{PAUL} & s_2 &= \text{PUAL} \\
    \text{sim}_{\text{jaro}} &= 0.92 \quad &
    \text{sim}_{\text{jaro}} &= 0.79 \quad &
    \text{sim}_{\text{jarowinkler}} &= 0.79 + 2 \cdot 0.1 \cdot (1 - 0.79) = 0.832 \\
    l &= 1 &
    l &= 2 &
    p &= 0.1 \\
    \text{sim}_{\text{jarowinkler}} &= 0.92 + 1 \cdot 0.1 \cdot (1 - 0.92) = 0.928
\end{align*}
\]
5.2 Token-based String Similarity Measures
Token-based Similarity

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

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

- Split string into short substrings of length $n$
  - by sliding a length $n$ window over the string
  - $n=2$: Bigrams
  - $n=3$: Trigrams
  - Variation: pad with $n - 1$ special characters
    - Emphasizes beginning and end of string
    - Variation: include positional information in order to weight similarities later

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

- Can be applied to words or n-grams

- **Overlap Coefficient:** $\text{sim}_{\text{overlap}}(x, y) = \frac{|\text{tok}(x) \cap \text{tok}(y)|}{\min(|\text{tok}(x)|,|\text{tok}(y)|)}$
  - example: useful for attribute label matching if one label might contain additional information, such as units of measurements or years

- **Jaccard Coefficient:**
  
  $\text{sim}_{\text{jaccard}}(x, y) = \frac{|\text{tok}(x) \cap \text{tok}(y)|}{|\text{tok}(x)| + |\text{tok}(y)| - |\text{tok}(x) \cap \text{tok}(y)|} = \frac{|\text{tok}(x) \cap \text{tok}(y)|}{|\text{tok}(x) \cup \text{tok}(y)|}$
  - focuses of both strings as all unique tokens are considered
  - widely used general purpose similarity measure for tokens

- Speeding up the calculation using an inverted index, see
  - Doan, Halevy: Principles of Data Integration, Chapter 4.3
Rare tokens are often more distinguishing and thus more relevant for determining the similarity of two strings.

TF/IDF weighting gives less weight to common tokens (domain-specific stopwords).

<table>
<thead>
<tr>
<th></th>
<th>Samsung</th>
<th>Galaxy</th>
<th>S9</th>
<th>S4</th>
<th>32GB</th>
<th>64GB</th>
</tr>
</thead>
<tbody>
<tr>
<td>p1</td>
<td>0</td>
<td>0</td>
<td>0.04</td>
<td>0</td>
<td>0</td>
<td>0.12</td>
</tr>
<tr>
<td>p2</td>
<td>0</td>
<td>0</td>
<td>0.04</td>
<td>0</td>
<td>0.04</td>
<td>0</td>
</tr>
<tr>
<td>p3</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0.12</td>
<td>0.04</td>
<td>0</td>
</tr>
</tbody>
</table>

Cosine similarity

- popular similarity measure for comparing weighted term vectors

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

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

$$\cos(d_1, d_2) = \frac{d_1 \cdot d_2}{\|d_1\| \|d_2\|}.$$
5.3 Hybrid String Similarity Measures
Monge-Elkan Similarity

- hybrid similarity measures split strings into tokens and apply internal similarity function to compare tokens
- can deal with typos and different order of words
- Monge-Elkan similarity searches for the best match for each token of the first string $x$ in the second string $y$

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

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

- focuses on first string $x$, as length of $y$ does not matter
- runtime complexity: quadratic in number of tokens ☹
Monge-Elkan – Example

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

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

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

- uses an internal similarity function (e.g. Levenshtein or Jaro) to calculate similarity between all pairs of tokens
- considers tokens as shared if similarity is above threshold
  - shared tokens: \( S = \{(x_i, y_j) | x_i \in tok(x) \land y_j \in tok(y): sim'(x_i, y_j) \geq \theta\} \)
  - unique tokens: \( U_{tok(x)} = \{x_i | x_i \in tok(x) \land y_j \in tok(y) \land (x_i, y_j) \notin S\} \)
- calculates overall similarity as
  \[
  sim_{jaccad\_ext}(x, y) = \frac{|S|}{|U_{tok(x)}| + |U_{tok(y)}| - |S|}
  \]
- focuses of both strings as all unique tokens are considered
  - as opposed to Monge-Elkan which focuses on tokens of first string
5.4 Phonetic String Similarity Measures
Soundex 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
- PUAL: P400
- JONES: J520
- JOHNSON: J525

J525 also: Jenkins, Jansen, Jameson
Kölner Phonetik

- Like Soundex, but aimed at German last names
- Letters get different codes based on the context
- Code length is not restricted
- Multiple occurrences of the same code and „0“ are removed

Examples:
- PAUL: 15
- PUAL: 15
- JONES: 68
- JOHNSON: 686

<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></td>
<td>-</td>
</tr>
<tr>
<td>H</td>
<td></td>
<td>-</td>
</tr>
<tr>
<td>B</td>
<td></td>
<td>1</td>
</tr>
<tr>
<td>P</td>
<td>not before H</td>
<td>2</td>
</tr>
<tr>
<td>D, T</td>
<td>not before C, S, Z</td>
<td>3</td>
</tr>
<tr>
<td>F, V, W</td>
<td></td>
<td>4</td>
</tr>
<tr>
<td>P</td>
<td>before H</td>
<td>48</td>
</tr>
<tr>
<td>G, K, Q</td>
<td>in the initial sound before A, H, K, L, O, Q, R, U, X</td>
<td>5</td>
</tr>
<tr>
<td>C</td>
<td>before A, H, K, O, Q, U, X but not after S, Z</td>
<td>6</td>
</tr>
<tr>
<td>X</td>
<td>not after C, K, Q</td>
<td>7</td>
</tr>
<tr>
<td>L</td>
<td></td>
<td>8</td>
</tr>
<tr>
<td>M, N</td>
<td></td>
<td></td>
</tr>
<tr>
<td>R</td>
<td></td>
<td></td>
</tr>
<tr>
<td>S, Z</td>
<td>after S, Z</td>
<td></td>
</tr>
<tr>
<td>C</td>
<td>in the initial sound, but not before A, H, K, L, O, Q, R, 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 Embedding-based String Similarity Measures
Embedding-based String Similarity

- Embeddings represent words as points in a multidimensional vector space
  - the calculation of embeddings exploits that semantically related words appear in similar contexts in large text corpora (distributional similarity)
- Similarity of two embeddings
  - cosine similarity, inner product
- Similarity of two sequences of embeddings
  - word movers distance
  - neural networks (LSTMs, BERT, SBERT)
- Embeddings are successfully used for
  - schema matching
  - blocking before entity matching
  - as foundation for supervised entity matching methods
5.6 Data Type Specific Similarity Measures
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

Example:
- \( \text{sim}_{\text{num.perc}}(2,000, 2,500) = 1 - \frac{200}{33} = 0.394 \) because \( pc = \frac{|2,000 - 2,500|}{2,500} \cdot 100 = 20\% \)
- \( \text{sim}_{\text{num.perc}}(200,000, 200,500) = 1 - \frac{250}{33} = 0.992 \) because \( pc = \frac{500}{200,500} \cdot 100 = 0.25\% \)
Time and Space Comparisons

- **Dates**
  - convert dates into days after year 0 $\rightarrow$ 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

- **More Similarity Measures for other Data Types**
  - Tan, Steinbach, Kumar: Introduction to Data Mining. Chapter 4
  - e.g. shopping baskets $\rightarrow$ vector of asymmetric binary variables $\rightarrow$ Jaccard
Summary
6. Learning Matching Rules

- **Problem**
  It is hard for humans to write good matching rules, as this requires a lot of knowledge about the data set and matching techniques
  - What kind of typos and other errors are contained in the data?
  - Which string similarity measure fits which attribute?
  - How to set similarity thresholds?
  - How to weight different attributes?

- **Possible solution**
  1. Manually label a certain amount of pairs as matches and non-matches
  2. Use machine learning to generate matching rule from this training data

- **Advantage**
  - The human does what she is good at: Understand the data
  - The computer does what it is good at: Learn detailed rules from examples
Training Data and Feature Generation

- **Training data**: \( T = \{(x_1, y_1, l_1), \ldots, (x_n, y_n, l_n)\} \), where
  - each \((x_i, y_i)\) is a record pair and
  - \(l_i\) is a label: “yes” if \(x_i\) matches \(y_i\) and “no” otherwise

- **Feature Generation**
  - define a set of features \(f_1, \ldots, f_m\), each quantifying one aspect of the domain judged possibly relevant to matching the records
  - feature = similarity measure applied to attribute pair
    - after normalizing both values
  - if you want the learning algorithm to decides which similarity measure fits best for a specific attribute pair, you generate multiple features for the pair
    - Levenshtein\((x.name, y.name)\)
    - Jaro\((x.name, y.name)\)
    - Jaccard\((\text{tokens}(x.name, y.name))\)
  - Feature engineering requires domain-knowledge, e.g. for value normalization
Example: Feature Generation

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

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

- \( s_1 \) 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 \((v_i, l_i)\)
   - \(v_i = f_1(x_i, y_i), \ldots, f_m(x_i, y_i)\) is a feature vector that encodes \((x_i, y_i)\) in terms of the features (list of similarity values)
   - thus \(T\) is transformed into \(T' = \{(v_1, l_1), \ldots, (v_n, l_n)\}\)

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

3. Optimize the hyperparameters of the learning algorithm
   - using training, validation (!), and test set
Goal: Learn rule \( \text{sim}(a, b) = \sum_{i=1}^{6} \alpha_i \cdot s_i(a, b) \)

Perform a least-squares linear regression on training data

\[
\begin{align*}
v_1 &= <[s_1(a_1, b_1), s_2(a_1, b_1), s_3(a_1, b_1), s_4(a_1, b_1), s_5(a_1, b_1), s_6(a_1, b_1)], 1> \\
v_2 &= <[s_1(a_2, b_2), s_2(a_2, b_2), s_3(a_2, b_2), s_4(a_2, b_2), s_5(a_2, b_2), s_6(a_2, b_2)], 1> \\
v_3 &= <[s_1(a_3, b_3), s_2(a_3, b_3), s_3(a_3, b_3), s_4(a_3, b_3), s_5(a_3, b_3), s_6(a_3, b_3)], 0>
\end{align*}
\]

to find weights \( \alpha_i \) that minimize the squared error

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

- Tree-based models **often perform better** than linear models
- The tree learning algorithm automatically selects the most discriminative features
- Always also test random forests and XGBoost
Example: Deep Learning of Matching Models

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

- Often outperform linear and tree-based matching models for less structured textual data given enough training pairs
  * e.g. product titles and descriptions, not numeric sensor data
How to Assemble Good Training Data?

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

- The more training data the better!
  - remember the learning curve

- Try to reduce labeling effort by
  - reusing existing information about matches
e.g. ISBN or GTIN numbers, owl:sameAs
  = weak supervision as quality is often questionable

Discussion Learning-based Approaches

- **Pros** compared to writing matching rules by hand
  - when writing rules by hand, you must manually decide if a particular feature is useful ➔ labor intensive and limits the number of features you can consider
  - learning-based approaches can automatically examine a large number of features

- **Cons**
  - you need to label training examples
  - you don’t know which examples matter to the algorithm and thus might label an unnecessary large amount of examples in order to cover the relevant corner-cases

- **Alternative**
  - use Active Learning in order to let the algorithm decide which examples matter
  - practical experience: Often $F_1 > 0.9$ after labeling less than 300 pairs

7. Combining Entity and Schema Matching

Often both entity and schema correspondences are unknown:

- Matching offers by e-shops to a central product catalog
  - Which product category? Which product? Which product feature?
- Matching Web tables to a central knowledge base
  - Which ontology class? Which instance? Which property?

Approach: Combine entity and schema matching in an iterative fashion

1. Compare entity names to generate candidate entity matches (Star Wars 1-6)
2. Determine class per table using voting (Class: Movie)
3. Employ duplicate-based schema matching to align attributes (attributes: name, year, director, producer)
4. Re-rank entity candidates based on attribute value similarity (matching rule: Similar name and similar year and similar director)
5. Go back to step 3 until correspondences stabilize

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

50 Years of Entity Linkage

Rule-based and stats-based
- Blocking: e.g., same name
- Matching: e.g., avg similarity of attribute values
- Clustering: e.g., transitive closure, etc.

Supervised learning
- Random forest for matching
  F-msr: >95% w. ~1M labels
- Active learning for blocking & matching
  F-msr: 80%-98% w. ~1000 labels

~2000 (Early ML)
Sup / Unsup learning
- Matching: Decision tree, SVM
  F-msr: 70%-90% w. 500 labels
- Clustering: Correlation clustering, Markov clustering

~2015 (ML)

2018 (Deep ML)
Deep learning
- Deep learning
- Entity embedding

Dong: ML for Entity Linkage. DI&ML tutorial at SIGMOD 2018.
References

- Peter Christen: Data Matching. Springer 2012.