Approximate Duplicate Detection

Helena Galhardas
IST, University of Lisbon and INESC-ID
Who am I?

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

Areas of Research:
- Databases
- Data Cleaning
- Information Extraction
- ETL (Extraction, Transformation, Loading)
Data journalism/Fact-checking
Context

Data journalism/Fact-checking needs Heterogeneous Data Integration
Context

Data journalism/Fact-checking needs

Heterogeneous Data Integration
Context

Data journalism/Fact-checking needs

Heterogeneous Data Integration needs

Approximate Duplicate Detection

(e.g., PERSON: Anne Martin and person: A. Martin)
Find records from different datasets that could be the same entity
The example refers to a data quality problem that is known under different names:

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

It is one of the data quality problems addressed by **data cleaning**
Other Data Quality Problems

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

- e.g., occupation=""

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

- e.g., Salary="-10"

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

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

- Approximate Duplicate Detection
Given two relational tables $R$ and $S$ with identical schema, we say tuple $r$ in $R$ matches a tuple $s$ in $S$ if they refer to the same real-world entity.

Those kind of pairs are called matches.

We want to find all such matches.
App. Duplicate Detection

R X S
Similarity measure
Algorithm

sim > θ
Duplicate

sim < δ
Non-duplicate
1st Challenge App. Duplicate Detection

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

- Use **string matching algorithms**
2nd Challenge App. Duplicate Detection

- **Efficiently** match a very large amount (tens of millions) of tuples
  - Record-set oriented matching: A potentially large set (or two sets) of records needs to be compared

- Aims at minimizing the number of tuple pairs to be compared and perform each of the comparisons efficiently
String matching – what is it?

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

Exs:
- “David R. Smith” and “David Smith”
- “1210 W. Dayton St, Madison WI” and “1210 West Dayton, Madison WI 53706”

- Formally:
  - Given two sets of strings \( x \) and \( y \), we want to find all pairs of strings \( (x, y) \), where \( x \in X \), \( y \in Y \) and such that \( x \) and \( y \) refer to the same real-world entity
  - These pairs are denoted matches
1\textsuperscript{st} Challenge String Matching

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

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

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

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

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

**Ex:** The cost of transforming string “David Smiths” into the string “Davidd Simth” is 4 where the required operations are:
  - Inserting character d (after David)
  - Substituting m by i
  - Substituting i by m
  - Deleting the last character of x, which is s

- Given two strings s1 and s2 and their edit distance, denoted by \(d(s_1, s_2)\), the similarity function can be given by:
  \[s(s_1, s_2) = 1 - \frac{d(s_1, s_2)}{\max(\text{length}(s_1), \text{length}(s_2))}\]

**Ex:** The similarity between “David Smiths” and “David Smiths” is 0.67
Computing the edit distance
(recurrence equation)

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

\[c(x_i,y_j) = 0 \text{ if } x_i = y_j, 1 \text{ otherwise}\]

\[d(0,0) = 0; d(i,0) = i; d(0,j) = j\]
Dynamic programming matrix

Computing the distance between “dva” and “dave”

<table>
<thead>
<tr>
<th></th>
<th>y0</th>
<th>y1</th>
<th>y2</th>
<th>y3</th>
<th>y4</th>
</tr>
</thead>
<tbody>
<tr>
<td>x0</td>
<td>0</td>
<td>1</td>
<td>2</td>
<td>3</td>
<td>4</td>
</tr>
<tr>
<td>x1</td>
<td>d</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>2</td>
</tr>
<tr>
<td>x2</td>
<td>v</td>
<td>2</td>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>x3</td>
<td>a</td>
<td>3</td>
<td>2</td>
<td>1</td>
<td>2</td>
</tr>
</tbody>
</table>

Cost of computing \( d(x,y) \) is \( \text{length}(x) \times \text{length}(y) \)
Example

- Surtday
- Saturday

<table>
<thead>
<tr>
<th></th>
<th>S</th>
<th>A</th>
<th>T</th>
<th>U</th>
<th>R</th>
<th>D</th>
<th>A</th>
<th>Y</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>1</td>
<td>2</td>
<td>3</td>
<td>4</td>
<td>5</td>
<td>6</td>
<td>7</td>
<td>8</td>
</tr>
<tr>
<td>S</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>2</td>
<td>3</td>
<td>4</td>
<td>5</td>
<td>6</td>
</tr>
<tr>
<td>U</td>
<td>2</td>
<td>1</td>
<td>1</td>
<td>2</td>
<td>2</td>
<td>3</td>
<td>4</td>
<td>5</td>
</tr>
<tr>
<td>R</td>
<td>3</td>
<td>2</td>
<td>2</td>
<td>3</td>
<td>2</td>
<td>3</td>
<td>4</td>
<td>5</td>
</tr>
<tr>
<td>T</td>
<td>4</td>
<td>3</td>
<td>3</td>
<td>3</td>
<td>3</td>
<td>3</td>
<td>4</td>
<td>5</td>
</tr>
<tr>
<td>D</td>
<td>5</td>
<td>4</td>
<td>4</td>
<td>3</td>
<td>3</td>
<td>4</td>
<td>3</td>
<td>5</td>
</tr>
<tr>
<td>A</td>
<td>6</td>
<td>5</td>
<td>4</td>
<td>4</td>
<td>4</td>
<td>4</td>
<td>3</td>
<td>4</td>
</tr>
<tr>
<td>Y</td>
<td>7</td>
<td>6</td>
<td>5</td>
<td>5</td>
<td>5</td>
<td>5</td>
<td>4</td>
<td>3</td>
</tr>
</tbody>
</table>

- Edit distance = 3
- Optimal alignment:

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

- Developed mainly to compare short strings, such as first and last names
- Given two strings \( x \) and \( y \),
  - Find the common chars \( x_i \) and \( y_i \) such that
    \[ x_i = y_j \text{ and } |i - j| \leq \min\{|x|, |y|\}/2 \]
  - **Common chars**: Those that are identical and are positionally “close to one another”
  - The number of common chars \( x_i \) in \( x \) and \( y_i \) in \( y \) is the same – it is called \( c \)
  - Compare the \( i \)-th common character of \( x \) with the \( i \)-th common character of \( y \). If they don’t match, then there is a **transposition**. Number of transpositions is \( t \)
  - Compute the Jaro score as:
    \[
    \text{jaro}(x,y) = \frac{1}{3} \left[ \frac{c}{|x|} + \frac{c}{|y|} + \frac{(c - t/2)}{c} \right]
    \]
- Cost of computation: \( O(|x||y|) \)
Example

- \( x = \text{jon}; \ y = \text{john} \)
  - \( c = 3 \)
  - Common character sequence in \( x \) and \( y \) is: \( \text{jon} \)
  - Nb. transpositions, \( t = 0 \)
  - \( \text{Jaro}(x,y) = \frac{1}{3} \left( \frac{3}{3} + \frac{3}{4} + \frac{3}{3} \right) = 0.917 \)
  - Similarity according to edit distance \( (x,y) = 0.75 \)

- \( x = \text{jon}; \ y = \text{ojhn} \)
  - \( C = 3 \)
  - Common character sequence in \( x \): \( \text{jon} \)
  - Common character sequence in \( y \): \( \text{ojn} \)
  - \( t = 2 \)
  - \( \text{Jaro}(x,y) = \frac{1}{3} \left( \frac{3}{3} + \frac{3}{4} + \frac{3-2}{2}/3 \right) = 0.81 \)
Jaro-Winkler measure

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

\[
\text{Jaro-Winkler}(x,y) = (1 - PL*PW) \times \text{jaro}(x,y) + PL*PW
\]
Set-based similarity measures

- View the strings as sets or multi-sets of tokens, and use set-related properties to compute similarity scores
  - Overlap measure
  - Jaccard measure
  - TF/IDF measure

- Several ways of generating tokens from strings
  - Words in the string (delimited by space char)
    - Tokens of “david smith”: {david, smith}
  - Q-grams: substrings of length q that are present in the string
    - 3-grams of “david”: {#da, dav, avi, vid, id#}
Overlap measure

Let $B_x$ and $B_y$ be the sets of tokens generated for strings $x$ and $y$

- **Overlap measure**: returns the number of common tokens

$$O(x, y) = |B_x \cap B_y|$$

Ex: $x = \text{dave}$; $y = \text{dav}$

- Set of all 2-grams of $x$: $B_x = \{\#d, \text{da, av, ve, e}\}$
- Set of all 2-grams of $y$: $B_y = \{\#d, \text{da, av, v}\}$
- $O(x,y) = 3$
Jaccard similarity score between two strings $x$ and $y$ is:

\[ J(x, y) = \frac{|B_x \cap B_y|}{|B_x \cup B_y|} \]

Ex: $x$ = “dave” with $B_x = \{#d, da, av, ve, e#\}$
$y$ = “dav” with $B_y = \{#d, da, av, v#\}$

\[ J(x, y) = \frac{3}{6} \]
TF/IDF measure

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

Ex: \( x = \text{"Apple Corporation, CA"} \)

\( y = \text{"IBM Corporation, CA"} \)

\( z = \text{"Apple Corp."} \)

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

- Each string is converted into a bag of terms (a document in IR terminology)
  Ex: x = aab; y=ac; z=a
  string x is converted into document Bx={a, a, b}
- For every term t and document d, compute:
  - Term frequency, $\text{tf}(t, d)$: number of times t occurs in d
    - $\text{tf}(a, x) = 2$
  - Inverse document frequency, $\text{idf}(t)$: total number of documents in the collection divided by the number of documents that contain t
    - $\text{idf}(a) = 3/3$
More definitions

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

\[ x = \text{aab}; \ y = \text{ac}; \ z = \text{a} \]
\[ B_x = \{\text{a, a, b}\}; \ B_y = \{\text{a, c}\}; \ B_z = \{\text{a}\} \]
\[ \text{tf}(\text{a, x}) = 2; \ \text{tf}(\text{b, x}) = 1; \ \ldots \ \text{tf}(\text{c, z}) = 0 \]
\[ \text{idf}(\text{a}) = 3/3 = 1; \ \text{idf}(\text{b}) = 3/1 = 3; \ \text{idf}(\text{c}) = 3/1 \]

<table>
<thead>
<tr>
<th></th>
<th>a</th>
<th>b</th>
<th>c</th>
</tr>
</thead>
<tbody>
<tr>
<td>\text{vx}</td>
<td>2</td>
<td>3</td>
<td>0</td>
</tr>
<tr>
<td>\text{vy}</td>
<td>3</td>
<td>0</td>
<td>3</td>
</tr>
<tr>
<td>\text{vz}</td>
<td>3</td>
<td>0</td>
<td>0</td>
</tr>
</tbody>
</table>

with \( v_d(t) = \text{tf}(t, d) \cdot \text{idf}(t) \)
Computing the TF/IDF similarity score

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

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

\[
S(p, q) = \frac{\sum_{t \in T} v_p(t) \cdot v_q(t)}{\sqrt{\sum_{t \in T} v_p(t)^2} \cdot \sqrt{\sum_{t \in T} v_q(t)^2}}
\]

**Ex:**

\[
s(x, y) = \frac{2 \cdot 3}{\sqrt{(2^2+3^2)} \cdot \sqrt{(3^2+3^2)}}
\]
Observations

- TF/IDF similarity score between two strings is **high** if they share many frequent terms (terms with high TF scores), unless these terms also commonly appear in other strings in the collection (terms with low IDF scores)

- An alternative score computation for dampening the TF and IDF components by a log factor is:
  
  $$v_d(t) = \log( tf(t,d) + 1 ).\log(idf(t))$$

  - With $v_d(t)$ normalized to length 1:
    
    $$v_d(t) = v_d(t) / \sqrt{\sum_{t \in T} v_d(t)^2}$$
Hybrid similarity measures

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

- Match strings based on their sound
  - Specially effective in matching names (e.g., “Meyer” and “Meier”), which are often spelled in different ways but sound the same

- Most commonly used similarity measure: **soundex**
  - Maps a surname \( x \) into a four-character code that captures the sound of the name
  - Two surnames are considered similar if they share the same code
Mapping a surname into a code (1)

Ex: $x = \text{Ashcraft}$

1. Keep the first letter of $x$ as the first letter of the code
   Ex: First letter of $x$ is A

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

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

3. Replace each sequence of identical digits by the digit itself
   - A226a13 -> A26a13

4. Drop all the non-digit letters, except the first one. Return the first four letters as the soundex code
   - A26a13 -> A261
Observations about the soundex code

- Is always a letter followed by three digits, padded by 0 if there are not enough digits
  Ex: soundex of Sue is S000

- Hashes similar sounding consonants (such as B, F, P, and V) into the same digit, which means it maps similar sounding names into the same soundex code
  Ex: Both Robert and Rupert map into R163

- Is not perfect
  Ex: fails to map Gough and Goff into the same code

- Widely used to match names in census records, vital records, genealogy databases
  - Works well for names from different origins
  - Doesn’t work well for Asian names, because the discriminating power of these names is based on vowels that are ignored by the code
A better phonetic similarity measure

- Metaphone
  - A string is converted into a code with variable size
  - It takes into account English pronouncing rules

- Improved versions of the algorithm:
  - Double metaphone
  - Metaphone 3
Outline String Matching

✓ Similarity measures
  ✓ Sequence-based
  ✓ Set-based
  ✓ Hybrid
  ✓ Phonetic

➢ Scaling up string matching
  ➢ Inverted index over strings
  ➢ Size filtering
  ➢ Prefix filtering
Recap. Challenge (2)

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

for each string \( x \in X \)
  
  for each string \( y \in Y \)
    
    if \( s(x,y) \geq t \), return \((x,y)\) as a matched pair

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

- To develop a method \texttt{FindCands} to quickly find the string that may match a given string \(x\)
- Then, use the following algorithm:

\[
\text{for each string } x \in X \\
\quad \text{use a method } \texttt{FindCands} \text{ to find a candidate set } Z \supseteq Y \\
\quad \text{for each string } y \in Z \\
\quad \quad \text{if } s(x,y) \geq t, \text{ return } (x,y) \text{ as a matched pair}
\]

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

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

- Explained using the Jaccard and Overlap measures
Inverted index over strings

1. Converts each string $y \in Y$ into a document $D(y)$, then builds an inverted index $I_y$ over these documents

2. Given a term $t$, use $I_y$ to quickly find the list of documents created from $Y$ that contain $t$, which gives the strings $y \in Y$ that contain $t$
Example

Two sets of strings X and Y to be matched:

Set X
1: {lake, mendota}
2: {lake, monona, area}
3: {lake, mendota, monona, dane}

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

<table>
<thead>
<tr>
<th>Terms in Y</th>
<th>ID Lists</th>
</tr>
</thead>
<tbody>
<tr>
<td>area</td>
<td>5, 6</td>
</tr>
<tr>
<td>lake</td>
<td>4, 6</td>
</tr>
<tr>
<td>mendota</td>
<td>6</td>
</tr>
<tr>
<td>monona</td>
<td>4, 5, 6</td>
</tr>
<tr>
<td>research</td>
<td>5</td>
</tr>
<tr>
<td>university</td>
<td>4</td>
</tr>
</tbody>
</table>

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

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

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

\[ J(x, y) = \frac{|x \cap y|}{|x \cup y|} \]

\[ 1/J(x, y) \geq |y|/|x| \geq J(x, y) \] (can be proved)

If \( x \) and \( y \) match, then

\[ J(x, y) \geq t \]

So:

\[ 1/t \geq |y|/|x| \geq t \iff |x|/t \geq |y| \geq |x|.t \]

Given a string \( x \in X \), only the strings that satisfy this Equation can possibly match \( x \).
Example

\[ x = \{\text{lake, mendota}\} \]
\[ t = 0.8 \]

Using the equation: \[ |x|/t \geq |y| \geq |x|.t \]
If \( y \in Y \) matches \( x \), we must have:
\[ 2/0.8 = 2.5 \geq |y| \geq 2*0.8 = 1.6 \]
So, none of the strings in the set \( Y \) satisfies this constraint!

Set \( Y \)

4: \{lake, monona, university\}
5: \{monona, research, area\}
6: \{lake, mendota, monona, area\}
B-tree index

- Procedure `FindCands` builds a B-tree over the sizes of strings in $\mathcal{Y}$
- Given a string $x \in X$, it uses the index to find strings in $\mathcal{Y}$ that satisfy equation:
  \[ |x|/t \geq |y| \geq |x| \cdot t \]
- Returns that set of strings that fit the size constraint
  - Effective when there is significant variability in the number of tokens in the strings $X$ and $\mathcal{Y}$
Outline

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

- **Approximate Duplicate Elimination (Data Fusion)**
Find records from different datasets that could be the same entity
1st Challenge

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

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

- Treat each tuple as a string and apply string matching algorithms
- **Exploit the structured nature of data** – *hand-crafted matching rules*
- **Automatically discover matching rules from training data** – *supervised learning*
- Iteratively assign tuples to clusters, no need of training data – *clustering*
- Model the matching domain with a probability distribution and reason with the distribution to take matching decisions – *probabilistic approaches*
- Exploit correlations among tuple pairs to match them all at once – *collective matching*
2nd Challenge

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

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

- And for minimizing the time taken to match each pair
  - **Short-circuiting** the matching process – exit immediately if one pair of attributes doesn’t match
Record-set oriented matching techniques

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

- And for minimizing the time taken to match each pair
  - **Short-circuiting** the matching process – exit immediately if one pair of attributes doesn’t match
Outline Record-oriented Matching

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

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

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

\[ \text{sim}(x, y) = \sum_{i=1}^{n} \alpha_i \cdot \text{sim}_i(x, y) \]

that returns the similarity score between two tuples \( x \) and \( y \), where:

- \( n \) is the nb attributes in each table \( X \) and \( Y \)
- \( \text{sim}_i(x, y) \) is the similarity score between the i-th attributes of \( x \) and \( y \)
- \( \alpha_i \) is a pre-specified weight indicating the importance of the i-th attribute to the total similarity score

\[ \alpha_i \text{ in } [0,1]; \sum_{i=1}^{n} \alpha_i = 1 \]

- If \( \text{sim}(x, y) \geq \beta \) we say tuples \( x \) and \( y \) match
Example

Table X

<table>
<thead>
<tr>
<th>Name</th>
<th>SSN</th>
<th>Addr</th>
</tr>
</thead>
<tbody>
<tr>
<td>Jack Lemmon</td>
<td>430-871-8294</td>
<td>Maple St</td>
</tr>
<tr>
<td>Harrison Ford</td>
<td>292-918-2913</td>
<td>Culver Blvd</td>
</tr>
<tr>
<td>Tom Hanks</td>
<td>234-762-1234</td>
<td>Main St</td>
</tr>
<tr>
<td>...</td>
<td>...</td>
<td>...</td>
</tr>
</tbody>
</table>

Table Y

<table>
<thead>
<tr>
<th>Name</th>
<th>SSN</th>
<th>Addr</th>
</tr>
</thead>
<tbody>
<tr>
<td>Tom Hanks</td>
<td>234-162-1234</td>
<td>Main Street</td>
</tr>
<tr>
<td>Kevin Spacey</td>
<td>928-184-2813</td>
<td>Frost Blvd</td>
</tr>
<tr>
<td>Jack Lemon</td>
<td>430-817-8294</td>
<td>Maple Street</td>
</tr>
<tr>
<td>...</td>
<td>...</td>
<td>...</td>
</tr>
</tbody>
</table>

- To match names, define a similarity function $\text{sim}_{\text{Name}}(x,y)$ based on the Jaro-Winkler distance.
- To match SSNs, define a function $\text{sim}_{\text{SSN}}(x,y)$ based on edit distance, etc.
- $\text{sim}(x,y) = 0.3*\text{sim}_{\text{Name}}(x,y) + 0.3*\text{sim}_{\text{SSN}}(x,y) + 0.2*\text{sim}_{\text{Addr}}(x,y)$
Complex matching rules (1)

- Linearly weighted matching rules do not work well when encoding more complex matching knowledge
  - Ex: two persons match if their names match approximately and either the SSN matches exactly or otherwise the addresses must match exactly
- Modify the similarity functions
  - Ex: \( \text{sim}'_{\text{SSN}}(x,y) \) returns true only if the SSN match exactly; analogous with \( \text{sim}'_{\text{Address}}(x,y) \)
- And then the matching rule would be:
  
  \[
  \begin{align*}
  \text{If } & \text{sim}_{\text{name}}(x,y) < 0.8 \text{ then return “no match”} \\
  \text{Else if } & \text{sim}'_{\text{SSN}}(x,y) = \text{true then return “match”} \\
  \text{Else if } & \text{sim}'_{\text{SSN}}(x,y) \geq 0.9 \text{ and } \text{sim}'_{\text{Address}}(x,y) = \text{true then return “match”} \\
  \text{Else return “no match”}
  \end{align*}
  \]
This kind of rules are often written in a high-level declarative language.
Easier to understand, debug, modify and maintain.

Still, it is labor intensive to write good matching rules.
Or not clear at all how to write them.
Or difficult to set the parameters $\alpha, \beta$. 
Learning-based matching

- Supervised learning
  - can also be unsupervised (clustering)

- Idea: learn a matching model $M$ from the training data, then apply $M$ to match new tuple pairs.

- Training data has the form:
  $$T = \{(x_1, y_1, l_1), (x_2, y_2, l_2), \ldots, (x_n, y_n, l_n)\}$$
  where each triple $(x_i, y_i, l_i)$ consists of a tuple pair $(x_i, y_i)$ and a label $l_i$ with value “yes” if $x_i$ matches $y_i$ and “no” otherwise.
Training (1)

- Define a set of features $f_1, f_2, \ldots, f_m$ thought to be potentially relevant to matching
  - each $f_i$ quantifies one aspect of the domain judged possibly relevant to matching the tuples
  - Each feature $f_i$ is a function that takes a tuple pair $(x, y)$ and produces a numerical, categorical, or binary value.

- The learning algorithm will use the training data to decide which features are in fact relevant
Training (2)

- Convert each training example \((x_i, y_i, l_i)\) in the set \(T\) into a pair:
  \[
  (\langle f_1(x_i, y_i), f_2(x_i, y_i), \ldots f_m(x_i, y_i) \rangle, c_i)
  \]
  where \(v_i = \langle f_1(x_i, y_i), f_2(x_i, y_i), \ldots f_m(x_i, y_i) \rangle\)
  is a feature vector that encodes the tuple pair \((x_i, y_i)\)
in terms of the features and \(c_i\) is an appropriately transformed version of label \(l_i\).

- Training set \(T\) is converted into a new training set \(T'\):
  \[
  \{(v_1, c_1), (v_2, c_2), \ldots, (v_n, c_n)\}
  \]
  and then we apply a learning algorithm such as SVM or Decision Trees to \(T'\) to learn a matching model \(M\).
Matching

- Given a new pair \( (x, y) \), transform it into a feature vector
  \[ v = <f_1(x, y), f_2(x, y), \ldots, f_m(x, y)> \]

- And then apply model \( M \) to predict whether \( x \) matches \( y \)
**Example**

**Table X**

<table>
<thead>
<tr>
<th>Name</th>
<th>Phone</th>
<th>City</th>
<th>State</th>
</tr>
</thead>
<tbody>
<tr>
<td>Dave Smith</td>
<td>(608) 395 9462</td>
<td>Madison</td>
<td>WI</td>
</tr>
<tr>
<td>Joe Wilson</td>
<td>(408) 123 4265</td>
<td>San Jose</td>
<td>CA</td>
</tr>
<tr>
<td>Dan Smith</td>
<td>(608) 256 1212</td>
<td>Middleton</td>
<td>WI</td>
</tr>
</tbody>
</table>

**Table Y**

<table>
<thead>
<tr>
<th>Name</th>
<th>Phone</th>
<th>City</th>
<th>State</th>
</tr>
</thead>
<tbody>
<tr>
<td>David D. Smith</td>
<td>395 9462</td>
<td>Madison</td>
<td>WI</td>
</tr>
<tr>
<td>Daniel W. Smith</td>
<td>256 1212</td>
<td>Madison</td>
<td>WI</td>
</tr>
</tbody>
</table>

**Matches:**

- \((x_1, y_1)\)
- \((x_3, y_2)\)

**Goal:** learn a linearly weighted rule to match \(x\) and \(y\)

\[
sim(x, y) = \sum_{i=1}^{n} \alpha_i \cdot \sim_i(x, y)
\]
Consider 6 possibly relevant features:

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

\[ \langle v_1, c_1 \rangle = \langle [f_1(x_1, y_1), f_2(x_1, y_1), f_3(x_1, y_1), f_4(x_1, y_1), f_5(x_1, y_1), f_6(x_1, y_1)], 1 \rangle \]
\[ \langle v_2, c_2 \rangle = \langle [f_1(x_2, y_2), f_2(x_2, y_2), f_3(x_2, y_2), f_4(x_2, y_2), f_5(x_2, y_2), f_6(x_2, y_2)], 1 \rangle \]
\[ \langle v_3, c_3 \rangle = \langle [f_1(x_3, y_3), f_2(x_3, y_3), f_3(x_3, y_3), f_4(x_3, y_3), f_5(x_3, y_3), f_6(x_3, y_3)], 0 \rangle \]

- **Goal:** learn the weight \( \alpha_i \), with \( i \) in [1, 6] that gives a linearly weighted matching rule of the form:
  \[ \text{sim}(x, y) = \sum_{i=1}^{6} \alpha_i \cdot f_i(x, y) \]

- Perform a **least-squares linear regression** on the transformed data set for finding the weights \( \alpha_i \) that minimize the squared error:
  \[ \sum_{i=1}^{3} \left( c_i - \sum_{j=1}^{6} \alpha_j \cdot f_j(v_i) \right)^2 \]
  where \( c_i \) is the label associated with feature vector \( v_i \) and \( f_j(v_i) \) is the j-th element of feature vector \( v_i \)

- Learn \( \beta \) from the training set by setting it to the value that lets us minimize the number of incorrect matching predictions.
Advantages/inconvenients
supervised learning

- **Advantages:**
  - Can automatically examine a large set of features to select the most useful ones
  - Can construct very complex rules, very difficult to construct in rule-based learning

- **Inconvenients:**
  - Requires a large number of training examples which can be labor intensive to obtain
Outline Record-oriented matching

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

- Scaling up record-oriented matching
  - Sorting: Sorted Neighborhood Method (SNM)
Record Pairs as Matrix

<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>
Number of comparisons: All pairs

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

400 comparisons
Reflexivity of Similarity

380 comparisons
Symmetry of Similarity

<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>

190 comparisons
Complexity

Problem: Too many comparisons!
- 10,000 customers => 49,995,000 comparisons
  - \((n^2 - n) / 2\)
  - Each comparison is already expensive.

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

- Partition the records (horizontally) and compare pairs of records only within a partition
  - Ex1: Partitioning by first two zip-digits
    - Ca. 100 partitions in Germany
    - Ca. 100 customers per partition
    - => 495,000 comparisons
  - Ex2: Partition by first letter of surname
  - ...

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

190 comparisons

WebClaimExplain Seminar Jan 2018
Blocking by ZIP

32 comparisons
Sorted Neighbourhood Method - SNM (or Windowing)

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

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

Example:

<table>
<thead>
<tr>
<th>First</th>
<th>Last</th>
<th>Address</th>
<th>ID</th>
<th>Key</th>
</tr>
</thead>
<tbody>
<tr>
<td>Sal</td>
<td>Stolfo</td>
<td>123 First Street</td>
<td>45678987</td>
<td>STLSAL123FRST456</td>
</tr>
</tbody>
</table>
2. Sort Data

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

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

- What is the optimal window size while
  - Maximizing accuracy
  - Minimizing computational cost

- The effectiveness of the SNM highly depends on the key selected to sort the records
  - A key is defined to be a sequence of a subset of attributes
  - Keys must provide sufficient discriminating power
## Example of Records and Keys

<table>
<thead>
<tr>
<th>First</th>
<th>Last</th>
<th>Address</th>
<th>ID</th>
<th>Key</th>
</tr>
</thead>
<tbody>
<tr>
<td>Sal</td>
<td>Stolfo</td>
<td>123 First Street</td>
<td>45678987</td>
<td>STLSAL123FRST456</td>
</tr>
<tr>
<td>Sal</td>
<td>Stolfo</td>
<td>123 First Street</td>
<td>45678987</td>
<td>STLSAL123FRST456</td>
</tr>
<tr>
<td>Sal</td>
<td>Stolpho</td>
<td>123 First Street</td>
<td>45678987</td>
<td>STLSAL123FRST456</td>
</tr>
<tr>
<td>Sal</td>
<td>Stiles</td>
<td>123 Forest Street</td>
<td>45654321</td>
<td>STLSAL123FRST456</td>
</tr>
</tbody>
</table>
Equational Theory - Example

- Two names are spelled nearly identically and have the same address
  - It may be inferred that they are the same person

- Two social security numbers are the same but the names and addresses are totally different
  - Could be the same person who moved
  - Could be two different people and there is an error in the social security number
A simplified rule in English

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

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

- In complex problems, an **expert’s assistance** is needed to write the equational theory.
Looses some matching pairs

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

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

- Widen the scanning window size, w

- Execute several independent runs of the SNM
  - Use a different key each time
  - Use a relatively small window
  - Call this the **Multi-Pass approach**
Multi-pass approach

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

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

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

From the example:

789912345 Kathi Kason 48 North St. (A)
879912345 Kathy Kason 48 North St. (B)
879912345 Kathy Smith 48 North St. (C)
Example of multi-pass matches

Pass 1 (Lastname discriminates)
KSNKAT48NRTH789 (Kathi Kason 789912345 )
KSNKAT48NRTH879 (Kathy Kason 879912345 )

Pass 2 (Firstname discriminates)
KATKSN48NRTH789 (Kathi Kason 789912345 )
KATKSN48NRTH879 (Kathy Kason 879912345 )

Pass 3 (Address discriminates)
48NRTH879KSNKAT (Kathy Kason 879912345 )
48NRTH879SMTKAT (Kathy Smith 879912345 )
References

Thank You!

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