2. Data Preparation and Cleaning

Boris Glavic
http://www.cs.iit.edu/~glavic/
http://www.cs.iit.edu/~cs520/
http://www.cs.iit.edu/~dbgroup/

2. Overview

Topics covered in this part
– Causes of Dirty Data
  – Constraint-based Cleaning
  – Outlier-based and Statistical Methods
  – Entity Resolution
  – Data Fusion

2. Causes of “Dirty” Data

Manual data entry or result of erroneous integration (cont.)
– Redundancy:
  • (ID: 1, City: Chicago, Zip: 60616)
  • (ID: 2, City: Chicago, Zip: 60616)
– Inconsistent references to entities
  • Dept. of Energy, DOE, Dep. Of Energy, …

2. Causes of “Dirty” Data

Manual data entry or result of erroneous integration
– Typos:
  • “Peter” vs. “Pteer”
  – Switching fields
    • “FirstName: New York, City: Pete”
  – Incorrect information
    • “City: New York, Zip: 60616”
  – Missing information
    • “City: New York, Zip: “

2. Cleaning Methods

Enforce Standards
– Applied in real world
– How to develop a standard not a fit for this lecture
– Still relies on no human errors

Constraint-based cleaning
– Define constraints for data
– “Make” data fit the constraints

Statistical techniques
– Find outliers and smoothen or remove
  • E.g., use a clustering algorithm
2. Overview

- Topics covered in this part
  - Causes of Dirty Data
  - Constraint-based Cleaning
  - Outlier-based and Statistical Methods
  - Entity Resolution
  - Data Fusion

2.1 Cleaning Methods

- Constraint-based cleaning
  - Choice of constraint language
  - Detecting violations to constraints
  - Fixing violations (automatically?)

2.1 Constraint Languages

- First work focused on functional dependencies (FDs)
- Extensions of FDs have been proposed to allow rules that cannot be expressed with FDs
  - E.g., conditional FDs only enforce the FD is a condition is met
  - More fine-grained control, e.g., zip -> city only if country is US
- Constraints that consider master data
  - Master data is highly reliable data such as a government issued zip, city lookup table

2.1 Example Constraints

<table>
<thead>
<tr>
<th>SSN</th>
<th>zip</th>
<th>city</th>
<th>name</th>
<th>boss</th>
<th>salary</th>
</tr>
</thead>
<tbody>
<tr>
<td>333-333-3333</td>
<td>60616</td>
<td>New York</td>
<td>Peter</td>
<td>Gert</td>
<td>50,000</td>
</tr>
<tr>
<td>333-333-9999</td>
<td>60615</td>
<td>Chicago</td>
<td>Gert</td>
<td>NULL</td>
<td>40,000</td>
</tr>
<tr>
<td>333-333-5599</td>
<td>60615</td>
<td>Schaumburg</td>
<td>Gertrud</td>
<td>Hans</td>
<td>10,000</td>
</tr>
<tr>
<td>333-333-6666</td>
<td>60616</td>
<td>Chicago</td>
<td>Hans</td>
<td>NULL</td>
<td>1,000,000</td>
</tr>
<tr>
<td>333-355-4343</td>
<td>60616</td>
<td>Chicago</td>
<td>Malcom</td>
<td>Hans</td>
<td>20,000</td>
</tr>
</tbody>
</table>

\[\forall \vec{x} : \neg(\phi(\vec{x}))\]

- Here we will look at FDs mainly and a bit at denial constraints
- Sometimes use logic-based notation introduced previously
2.1 Example Constraints

Example: Constraints Languages

<table>
<thead>
<tr>
<th>SSN</th>
<th>Zip</th>
<th>City</th>
<th>Name</th>
<th>Income</th>
</tr>
</thead>
<tbody>
<tr>
<td>333-333-3333</td>
<td>60616</td>
<td>New York</td>
<td>Peter</td>
<td>50,000</td>
</tr>
<tr>
<td>333-333-9999</td>
<td>60615</td>
<td>Chicago</td>
<td>Gert</td>
<td>40,000</td>
</tr>
<tr>
<td>333-333-5599</td>
<td>60615</td>
<td>Schaumburg</td>
<td>Gertrud</td>
<td>10,000</td>
</tr>
<tr>
<td>333-333-6666</td>
<td>60616</td>
<td>Chicago</td>
<td>Hans</td>
<td>1,000,000</td>
</tr>
<tr>
<td>333-355-4343</td>
<td>60616</td>
<td>Chicago</td>
<td>Malcom</td>
<td>20,000</td>
</tr>
</tbody>
</table>

Constraints

1. The zip code uniquely determines the city
2. Nobody should earn more than their direct superior
3. Salaries are non-negative

\[ \neg (E(x, y) \land E(x', y')) \]

2.1 Constraint based Cleaning Overview

- Define constraints
- Given database D
  - 1) Detect violations of constraints
    - We already saw example of how this can be done using queries. Here a bit more formal
  - 2) Fix violations
    - In most cases there are many different ways to fix the violation by modifying the database (called solution)
      - What operations do we allow: insert, delete, update
      - How do we choose between alternative solutions

2.1 Constraint based Cleaning Overview

- Study 1) + 2) for FDs
- Given database D
  - 1) Detect violations of constraints
    - We already saw example of how this can be done using queries. Here a bit more formal
  - 2) Fix violations
    - In most cases there are many different ways to fix the violation by modifying the database (called solution)
      - What operations do we allow: insert, delete, update
      - How do we choose between alternative solutions
2.1 Example Constraints

Example: Constraint Violations

<table>
<thead>
<tr>
<th>SSN</th>
<th>Zip</th>
<th>City</th>
<th>Name</th>
</tr>
</thead>
<tbody>
<tr>
<td>333-333-3333</td>
<td>60616</td>
<td>New York</td>
<td>Peter</td>
</tr>
<tr>
<td>333-333-9999</td>
<td>60615</td>
<td>Chicago</td>
<td>Gert</td>
</tr>
<tr>
<td>333-333-5599</td>
<td>60615</td>
<td>Schaumburg</td>
<td>Gertrud</td>
</tr>
<tr>
<td>333-333-6666</td>
<td>60616</td>
<td>Chicago</td>
<td>Hans</td>
</tr>
<tr>
<td>333-355-4343</td>
<td>60616</td>
<td>Chicago</td>
<td>Malcom</td>
</tr>
</tbody>
</table>

How to repair?

Deletion:
- remove some conflicting tuples
- quite destructive

Update:
- modify values to resolve the conflict
- equate RHS values (city here)
- disequate LHS value (zip)

Example: Constraint Repair

<table>
<thead>
<tr>
<th>SSN</th>
<th>Zip</th>
<th>City</th>
<th>Name</th>
</tr>
</thead>
<tbody>
<tr>
<td>333-333-3333</td>
<td>60616</td>
<td>New York</td>
<td>Peter</td>
</tr>
<tr>
<td>333-333-9999</td>
<td>60615</td>
<td>Chicago</td>
<td>Gert</td>
</tr>
<tr>
<td>333-333-5599</td>
<td>60615</td>
<td>Schaumburg</td>
<td>Gertrud</td>
</tr>
<tr>
<td>333-333-6666</td>
<td>60616</td>
<td>Chicago</td>
<td>Hans</td>
</tr>
<tr>
<td>333-355-4343</td>
<td>60616</td>
<td>Chicago</td>
<td>Malcom</td>
</tr>
</tbody>
</table>

2.1 Constraint based Cleaning Overview

- How to repair?
- Deletion:
  - remove some conflicting tuples
  - quite destructive
- Update:
  - modify values to resolve the conflict
  - equate RHS values (city here)
  - disequate LHS value (zip)
- Insertion?
  - Not for FDs, but e.g., FKs

2.1 Detecting Violations

- Given FD A -> B on R
  - Recall logical representation
  - For all X, X': R(X) and R(X') and A=A' -> B=B'
  - Only violated if we find two tuples where A=A', but B != B'
  - In datalog
    - Q(): R(X), R(X'), A=A', B!=B'
  - In SQL
    - SELECT EXISTS (SELECT *
      FROM R x, R y
      WHERE A=A' AND B<>B')
2.1 Example Constraints

Example: SQL Violation Detection

```sql
Relation: person(name, city, zip)
FD1: zip -> city

Violation Detection Query
SELECT EXISTS (SELECT *
FROM person x, person y
WHERE x.zip = y.zip
AND x.city <> y.city)
To know which tuples caused the conflict:
SELECT *
FROM person x, person y
WHERE x.zip = y.zip
AND x.city <> y.city)
```

2.1 Fixing Violations

- Principle of minimality
  - Choose solution that minimally modifies the database
- Updates:
  - Need a cost model
- Deletes:
  - Minimal number of deletes

2.1 Constraint Repair Problem

Definition: Constraint Repair Problem (restated)

Given set of constraints Σ and a database instance I which violates the constraints find a clean instance I’ (does not violate the constraints) with cost(I,I’) being minimal

- Cost metrics that have been used
  - Deletion + Insertion
    - \( \Delta(I, I') = (I - I') \cup (I' - I) \)
    - S-repair: minimize measure above under set inclusion
    - C-repair: minimize cardinality
  - Update
    - Assume distance metric \( d \) for attribute values

2.1 Cost Metrics

- Deletion + Insertion
  - \( \Delta(I, I') = (I - I') \cup (I' - I) \)
  - S-repair: minimize measure above under set inclusion
  - C-repair: minimize cardinality
- Update
  - Assume single relation R with uniquely identified tuples
  - Assume distance metric \( d \) for attribute values
  - Schema(R) = attributes in schema of relation R
  - \( t' \) is updated version of tuple \( t \)
  - Minimize: \( \sum_{t \in R} \sum_{A \in \text{Schema}(R)} d(t.A, t'.A) \)

2.1 Naïve FD Repair Algorithm

- Update
  - Assume single relation R with uniquely identified tuples
  - Assume distance metric \( d \) for attribute values
  - Schema(R) = attributes in schema of relation R
  - \( t' \) is updated version of tuple \( t \)
  - Minimize: \( \sum_{t \in R} \sum_{A \in \text{Schema}(R)} d(t.A, t'.A) \)
  - We focus on this one
  - This is NP-hard
    - Heuristic algorithm

- FD Repair Algorithm: 1. Attempt
  - For each FD \( X \rightarrow Y \) in Σ run query to find pairs of tuples that violate the constraint
  - For each pair of tuples \( t \) and \( t' \) that violate the constraint
    - update \( t.Y \) to \( t'.Y \)
    - choice does not matter because cost is symmetric, right?
2.1 Constraint Repair

Example: Constraint Repair

<table>
<thead>
<tr>
<th>ID</th>
<th>SSN</th>
<th>zip</th>
<th>city</th>
<th>name</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>60616</td>
<td>60615</td>
<td>Chicago</td>
<td>Hans</td>
</tr>
<tr>
<td>2</td>
<td>60615</td>
<td>60616</td>
<td>Chicago</td>
<td>Malcom</td>
</tr>
<tr>
<td>3</td>
<td>60615</td>
<td>60615</td>
<td>New York</td>
<td>Gertrud</td>
</tr>
<tr>
<td>4</td>
<td>60616</td>
<td>60615</td>
<td>Schaumburg</td>
<td>Gert</td>
</tr>
<tr>
<td>5</td>
<td>60615</td>
<td>60616</td>
<td>Chicago</td>
<td>Peter</td>
</tr>
</tbody>
</table>

t1 and t2: set t1.city = Schaumburg
t1 and t2: set t1.city = Chicago
t2 and t3: set t2.city = Schaumburg

Our updates may cause new violations!

2.1 Problems with the Algorithm

• FD Repair Algorithm: 2. Attempt

– I’ = I

– 1) For each FD X -> Y in \( \Sigma \) run query to find pairs of tuples that violate the constraint

– 2) For each pair of tuples \( t \) and \( t' \) that violate the constraint: \( t.X = t'.X \) and \( t.Y \neq t'.Y \)

  • update \( t.Y \) to \( t'.Y \)

  – choice does not matter because cost is symmetric, right?

  – Our updates may cause new violations!

2.1 Problems with the Algorithm

• FD Repair Algorithm: 2. Attempt

– I’ = I

– 1) For each FD X -> Y in \( \Sigma \) run query to find pairs of tuples that violate the constraint

– 2) For each pair of tuples \( t \) and \( t' \) that violate the constraint: \( t.X = t'.X \) and \( t.Y \neq t'.Y \)

  • update \( t.Y \) to \( t'.Y \)

  – choice does not matter because cost is symmetric, right?

  – 3) If we changed I’ goto 1)

• FD Repair Algorithm: 1. Attempt

– For each FD X -> Y in \( \Sigma \) run query to find pairs of tuples that violate the constraint

– For each pair of tuples \( t \) and \( t' \) that violate the constraint: \( t.X = t'.X \) and \( t.Y \neq t'.Y \)

  • update \( t.Y \) to \( t'.Y \)

  – choice does not matter because cost is symmetric, right?

2.1 Constraint Repair

Example: Constraint Repair

<table>
<thead>
<tr>
<th>ID</th>
<th>SSN</th>
<th>zip</th>
<th>city</th>
<th>name</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>60616</td>
<td>60615</td>
<td>Chicago</td>
<td>Hans</td>
</tr>
<tr>
<td>2</td>
<td>60615</td>
<td>60616</td>
<td>Chicago</td>
<td>Malcom</td>
</tr>
<tr>
<td>3</td>
<td>60615</td>
<td>60615</td>
<td>New York</td>
<td>Gertrud</td>
</tr>
<tr>
<td>4</td>
<td>60616</td>
<td>60615</td>
<td>Schaumburg</td>
<td>Gert</td>
</tr>
<tr>
<td>5</td>
<td>60615</td>
<td>60616</td>
<td>Chicago</td>
<td>Peter</td>
</tr>
</tbody>
</table>

t1 and t2: set t1.city = New York
t1 and t2: set t1.city = Schaumburg
t2 and t3: set t2.city = Chicago

May never terminate
2.1 Problems with the Algorithm

- FD Repair Algorithm: 2. Attempt
  - Even if we succeed the repair may not be minimal. There may be many tuples with the same X values
     - They all have to have the same Y value
     - Choice which to update matters!

- FD Repair Algorithm: 3. Attempt
  - Equivalence Classes
     - Keep track of sets of cells (tuple, attribute) that have to have the same values in the end (e.g., all Y attribute values for tuples with same X attribute value)
     - These classes are updated when we make a choice
     - Choose Y value for equivalence class using minimality, e.g., most common value
  - Observation
     - Equivalence Classes may merge, but never split if we only update RHS of all tuples with same X at once
     - Thus we can find an algorithm that terminates

- FD Repair Algorithm: 3. Attempt
  - Initialize:
     - Each cell in its own equivalence class
     - Put all cells in collection unresolved
  - While unresolved is not empty
     - Remove tuple t from unresolved
     - Pick FD X -> Y (e.g., random)
     - Compute set of tuples S that have same value in X
     - Merge all equivalence classes for all tuples in S and attributes in Y
     - Pick values for Y (update all tuples in S to Y)

2.1 Constraint Repair

- Example: Constraint Repair
  - Changes: t_1.city = Chicago
  - Not so cheap: set t_4.city and t_5.city = New York

2.1 Consistent Query Answering

- As an alternative to fixing the database which requires making a choice we could also leave it dirty and try to resolve conflicts at query time
  - Have to reason over answers to the query without knowing which of the possible repairs will be chosen
  - Intuition: return tuples that would be in the query result for every possible repair
2.1 Constraint Repair

Example: Constraint Repair

<table>
<thead>
<tr>
<th>SSN</th>
<th>zip</th>
<th>city</th>
<th>name</th>
</tr>
</thead>
<tbody>
<tr>
<td>333-333-3333</td>
<td>60616</td>
<td>New York</td>
<td>Peter</td>
</tr>
<tr>
<td>333-333-5599</td>
<td>60615</td>
<td>Schaumburg</td>
<td>Gertrud</td>
</tr>
<tr>
<td>333-333-9999</td>
<td>60615</td>
<td>Chicago</td>
<td>Hans</td>
</tr>
<tr>
<td>333-333-6666</td>
<td>60616</td>
<td>Chicago</td>
<td>Malcom</td>
</tr>
</tbody>
</table>

Cheaper: \( t_1.\text{city} = \text{Chicago} \)
Not so cheap: set \( t_4.\text{city} \) and \( t_5.\text{city} = \text{New York} \)

2.2 Statistical and Outlier

• Assumption
  – Errors can be identified as outliers

• How do we find outliers?
  – Similarity-based:
    • Object is dissimilar to all (many) other objects
    • E.g., clustering, objects not in cluster are outliers
  – Some type of statistical test:
    • Given a distribution (e.g., fitted to the data)
    • How probable is it that the point has this value?
    • If low probability -> outlier

2.3 Entity Resolution

• Entity Resolution (ER)
• Alternative names
  – Duplicate detection
  – Record linkage
  – Reference reconciliation
  – Entity matching
  – …

2.3 Entity Resolution

**Definition: Entity Resolution Problem**

Given sets of tuples \( A \) compute equivalence relation \( E(\bar{A}) \) which denotes that tuple \( t \) and \( t' \) represent the same entity.

• Intuitively, \( E \) should be based on how similar \( t \) and \( t' \) are
  – Similarity measure?

• \( E \) should be an equivalence relation
  – If \( t \) is the same as \( t' \) and \( t' \) is the same as \( t'' \) then \( t \) should be the same as \( t'' \)
2.3 Entity Resolution

Example: Two tuples (objects) that represent the same entity

<table>
<thead>
<tr>
<th>SSN</th>
<th>zip</th>
<th>city</th>
<th>name</th>
</tr>
</thead>
<tbody>
<tr>
<td>333-333-3333</td>
<td>60616</td>
<td>Chicago</td>
<td>Peter</td>
</tr>
<tr>
<td>3333333333</td>
<td>IL</td>
<td>60616</td>
<td>Peter</td>
</tr>
</tbody>
</table>

2.3 Entity Resolution

- Similarity based on similarity of attribute values
  - Which distance measure is appropriate?
  - How do we combine attribute-level distances?
  - Do we consider additional information?
    - E.g., foreign key connections
    - How similar should duplicates be?
    - E.g., fixed similarity threshold
    - How to guarantee transitivity of E
      - E.g., do this afterwards

Example: Per attribute similarity

<table>
<thead>
<tr>
<th>SSN</th>
<th>zip</th>
<th>city</th>
<th>name</th>
</tr>
</thead>
<tbody>
<tr>
<td>333-333-3333</td>
<td>60616</td>
<td>Chicago</td>
<td>Peter</td>
</tr>
<tr>
<td>3333333333</td>
<td>IL</td>
<td>60616</td>
<td>Peter</td>
</tr>
</tbody>
</table>

2.3 Entity Resolution – Distance Measures

- Edit-distance
  - Measures similarity of two strings
  - \( d(s, s') = \text{minimal number of insert, replace, delete operations (single character) that transform } s \text{ into } s' \)
  - Is symmetric (actually a metric)
    - Why?

Example: Edition Distance

Given two strings \( s, s' \), we define the edit distance \( d(s, s') \) as the minimum number of single character insert, replacements, deletions that transforms \( s \) into \( s' \).

Example:

\( \text{NEED} \rightarrow \text{STREET} \)

- Trivial solution: delete all chars in \( \text{NEED} \), then insert all chars in \( \text{STREET} \)
- Gives upper bound on distance \( \text{len}(<\text{NEED}>)+\text{len}(<\text{STREET}>)=10 \)

Example:

\( \text{NEED} \rightarrow \text{STREET} \)

- Minimal solution:
  - insert T
  - replace N with R
  - replace D with T
- \( d(\text{NEED, STREET}) = 4 \)
2.3 Entity Resolution

• Principal of optimality
  – Best solution of a subproblem is part of the best solution for the whole problem

• Dynamic programming algorithm
  – $D(i,j)$ is the edit distance between prefix of $len_i$ of $s$ and prefix of $len_j$ of $s'$
  – $D(len(s),len(s'))$ is the solution
  – Represented as matrix
  – Populate based on rules shown on the next slide

Recursive definition

– $D(i,0) = i$
  • Cheapest way of transforming prefix $s[i]$ into empty string is by deleting all $i$ characters in $s[i]$
– $D(0,j) = j$
  • Same holds for $s'[j]$
– $D(i,j) = \min \{ D(i-1,j) + 1, D(i,j-1) + 1, D(i-1,j-1) + d(i,j) \}$
  • $d(i,j) = 1$ if $s[i] \neq s'[j]$ and 0 else

Example:

NEED -> STREET

<table>
<thead>
<tr>
<th>0</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
<th>6</th>
</tr>
</thead>
<tbody>
<tr>
<td>N</td>
<td>1</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>E</td>
<td>2</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>E</td>
<td>3</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>D</td>
<td>4</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

NEED -> STREET

<table>
<thead>
<tr>
<th>0</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
<th>6</th>
</tr>
</thead>
<tbody>
<tr>
<td>N</td>
<td>1</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>E</td>
<td>2</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>E</td>
<td>3</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>D</td>
<td>4</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
2.3 Entity Resolution

Example:

NEED -> STREET

\[
\begin{array}{cccccc}
S & T & R & E & E & T \\
0 & 1 & 2 & 3 & 4 & 5 & 6 \\
N & 1 & 1 & 2 & 3 & 4 & \\
E & 2 & 2 & 2 & 3 & 3 & \\
E & 3 & 3 & 3 & 3 & 3 & \\
D & 4 & 4 & 4 & 4 & 4 & 4 & \\
\end{array}
\]

2.3 Entity Resolution – Distance Measures

- Other sequence-based measures for string similarity
  - Needleman-Wunsch
    - Missing character sequences can be penalized differently from character changes
  - Affine Gap Measure
    - Limit influence of longer gaps
    - E.g., Peter Friedrich Mueller vs. Peter Mueller
  - Smith-Waterman Measure
    - More resistant to reordering of elements in the string
    - E.g., Prof. Franz Mueller vs. F. Mueller, Prof.

- Other sequence-based measures for string similarity
  - Jaro-Winkler
    - Consider shared prefixes
    - Consider distance of same characters in strings
    - E.g., Johann vs. ojhan vs. ohannj
  - See textbook for details!
2.3 Entity Resolution – Distance Measures

- **Token-set based measures**
  - Split string into tokens
    - E.g., single characters
    - E.g., words if string represents a longer text
  - Potentially normalize tokens
    - E.g., word tokens replace word with its stem
    - Generating, generated, generates are all replaced with generate
  - Represent string as set (multi-set) of tokens

**Example:**
Input string: \( S = \) "the tokenization of strings is commonly used in information retrieval"
Set of tokens: \( \text{Tok}(S) = \{\text{commonly}, \text{in}, \text{information}, \text{is}, \text{of}, \text{retrieval}, \text{strings}, \text{the}, \text{tokenization}, \text{used}\}\)
Bag of tokens: \( \text{Bag}(S) = \{\text{commonly}:1, \text{in}:1, \text{information}:1, \text{is}:1, \text{of}:1, \text{retrieval}:1, \text{strings}:1, \text{the}:1, \text{tokenization}:1, \text{used}:1\}\)

- **Jaccard-Measure**
  - \( B_s = \text{Tok}(s) = \) token set of string \( s \)
  - Jaccard measures relative overlap of tokens in two strings
    - Number of common tokens divided by total number of tokens
    
    \[ d_{\text{jacc}}(s, s') = \frac{|B_s \cap B_{s'}|}{|B_s \cup B_{s'}|} \]

**Example:**
Input string: \( S = \) "nanotubes are used in these experiments to…"
\( S' = \) "we consider nanotubes in our experiments…"
\( S'' = \) "we prove that P=NP, thus solving …"

\( \text{Tok}(S) = \{\text{are}, \text{experiments}, \text{in}, \text{nanotubes}, \text{these}, \text{to}\}\)
\( \text{Tok}(S') = \{\text{consider}, \text{experiments}, \text{in}, \text{nanotubes}, \text{our}, \text{we}\}\)
\( \text{Tok}(S'') = \{\text{P=NP}, \text{prove}, \text{solving}, \text{that}, \text{thus}, \text{we}\}\)

\[ d_{\text{jacc}}(S, S') = 3 / 10 = 0.3 \]
\[ d_{\text{jacc}}(S, S'') = 0 / 13 = 0 \]
\[ d_{\text{jacc}}(S', S'') = 1 / 11 = 0.0909 \]

- **Other set-based measures**
  - **TF/IDF:** term frequency, inverse document frequency
    - Take into account that certain tokens are more common than others
    - If two strings (called documents for TF/IDF) overlap on uncommon terms they are more likely to be similar than if they overlap on common terms
      - E.g., the vs. carbon nanotube structure
2.3 Entity Resolution

- **TF/IDF**: term frequency, inverse document frequency
  - Represent documents as feature vectors
  - One dimension for each term
  - Value computed as frequency times IDF
  - Inverse of frequency of term in the set of all documents
  - Compute cosine similarity between two feature vectors
  - Measure how similar they are in term distribution (weighted by how uncommon terms are)
  - Size of the documents does not matter
  - See textbook for details

- **Entity resolution**
  - Concatenate attribute values of tuples and use string similarity measure
  - Loose information encoded by tuple structure
  - E.g., [Gender: male, Salary: 9000] -> “Gender: male, Salary: 9000” or -> “male, 9000”
  - Combine distance measures for single attributes
    - Weighted sum or more complex combinations
      - E.g., \( d(t, t') = w_1 \times d_A(t.A, t_0.A) + w_2 \times d_B(t.B, t_0.B) \)
    - Use quadratic distance measure
      - E.g., earth-movers distance

- **Weighted linear combination**
  - Say tuples have \( n \) attributes
  - \( w_c \) predetermined weight of an attribute
  - \( d_i(t, t') \): similarity measure for the \( i \)th attribute
  - Tuples match if \( d(t, t') > \beta \) for a threshold \( \beta \)

**Example:** Weighted sum of attribute similarities

<table>
<thead>
<tr>
<th>SSN</th>
<th>zip</th>
<th>city</th>
<th>name</th>
</tr>
</thead>
<tbody>
<tr>
<td>0112233456</td>
<td>60616</td>
<td>Chicago</td>
<td>Peter</td>
</tr>
<tr>
<td>0123456789</td>
<td>IL 60616</td>
<td>Petre</td>
<td></td>
</tr>
</tbody>
</table>

**Assumption:** SSNs and names are most important, city and zip are not very predictive

\( w_{SSN} = 0.4, w_{zip} = 0.05, w_{city} = 0.15, w_{name} = 0.4 \)

\[ d(t, t') = 0.4 \times 1 + 0.05 \times 0.8 + 0.15 \times 0 + 0.4 \times 0.6 \]
\[ = 0.4 + 0.04 + 0 + 0.24 \]
\[ = 0.68 \]
2.3 Entity Resolution

• Entity resolution
  – Rule-based approach
  – Learning-based approaches
  – Clustering-based approaches
  – Probabilistic approaches to matching
  – Collective matching

• Rule-based approach
  – Collection (list) of rules
  – if \( d_{name}(t,t') < 0.6 \) then unmatched
  – if \( d_{zip}(t,t') = 1 \) and \( t.country = USA \) then matched
  – if \( t.country \neq t'.country \) then unmatched

• Advantages
  – Easy to start, can be incrementally improved

• Disadvantages
  – Lot of manual work, large rule-bases hard to understand

• Learning-based approach
  – Build all pairs (\( t,t' \)) for training dataset
  – Represent each pair as feature vector from, e.g., similarities
  – Train classifier to return \{match,no match\}

• Advantages
  – Automated

• Disadvantages
  – Requires training data

• Clustering-based approach
  – Apply clustering method to group inputs
  – Typically hierarchical clustering method
  – Clusters now represent entities
    • Decide how to merge based on similarity between clusters

• Advantages
  – Automated, no training data required

• Disadvantages
  – Choice of cluster similarity critical
2.3 Entity Resolution

- Entity resolution
  - Rule-based approach
  - Learning-based approaches
  - Clustering-based approaches
  - Probabilistic approaches to matching
    - Collective matching
      • See text book

2.4 Data Fusion

- Data Fusion = how to combine (possibly conflicting) information from multiple objects representing the same entity
  - Choose among conflicting values
    • If one value is missing (NULL) choose the other one
    • Numerical data: e.g., median, average
    • Consider sources: have more trust in certain data sources
    • Consider value frequency: take most frequent value
    • Timeliness: latest value

Outline

0) Course Info
1) Introduction
2) Data Preparation and Cleaning
3) Schema matching and mapping
4) Virtual Data Integration
5) Data Exchange
6) Data Warehousing
7) Big Data Analytics
8) Data Provenance