Schema Matching and Mapping

Helena Galhardas
DEI IST

References

- Chapter 5 (all sections except 5.6.2, 5.9), “Principles of Data Integration” by AnHai Doan, Alon Halevy, Zachary Ives
- R.J. Miller, L.M. Haas, and M. Hernandez. “Schema Matching as Query Discovery”. In VLDB, 2000
- Slides of the course: CIS 550 – Database & Information Systems, Univ. Pennsylvania, Zachary Ives
What we have studied before?
- **Formalisms** for specifying source descriptions and how to use these descriptions to reformulate queries

What is the goal now?
- **Set of techniques** that helps a designer create semantic matchings and mappings that enable the creation of source descriptions for a particular data integration application
  - Heuristic task
  - Idea is to reduce the time it takes

---

**Motivating example**

- **DVD vendor schema**
  - Movies(id, title, year)
  - Products(mid, releaseDate, releaseCompany, basePrice, rating, saleLocID)
  - Locations(lid, name, taxRate)

- **Online Aggregator Schema**
  - Item(name, releaseInfo, classification, price)
Schema matching

- Relates a set of elements in schema S to a set of elements in schema T, without specifying in detail the exact nature of the relationship
- Simplest: one-to-one
  - Ex: `Movies.title ≈ Items.name; Products.rating ≈ Items.classification`
- One-to-many: relates one element in schema S to multiple elements in schema T
  - Ex: `Items.price ≈ Products.basePrice*(1+Locations.taxRate)`

Schema mapping

- **Query expression** that relates a schema S with a schema T
  - Ex1: the following semantic mapping:
    ```sql
    select name as title from Items
    ```
  - Ex2: and this one in reverse direction
    ```sql
    select (basePrice*(1+taxRate)) as price
    from Products, Locations
    where Products.saleLocID = Locations.lid
    ```
Process of creating schema mappings

1) **Schema matching**: Correspondences between elements of two schemas

   Ex: the `title/rating` in the vendor schema corresponds to the `title/classification` in the aggregator schema

2) **Creating schema mappings from the correspondences** (and filling in missing details): Specifies the transformations that have to be applied to the source data in order to produce the target data

   Ex: To compute the value of `price` in the aggregator schema, have to join the `Product` table with `Locations` table, using `saleLocID = lid`, and add the appropriate local tax given by `taxRate`

Challenges of schema matching and mapping (1)

- The semantics is not fully captured in schemas
  - Schema mapping system needs to understand the intended semantics of schema
  - But schema does not completely describe its full meaning

- Schema clues may be unreliable
  - Formal spec (names, structure, types, data values) and text descriptions can be unreliable (ex: two attributes name referring to different real-world concepts)
Challenges of schema matching and mapping (2)

- **Intended semantics can be subjective**
  - Depending on the designers, it may be difficult to find a consensus about whether two attributes match

- **Correctly combining the data is difficult**
  - Difficult to elaborate schema mappings from schema matchings
  - The designer must find the correct way to combine the data values of the various attributes, typically using a join and filtering conditions
  - To decide the right combination, the designer often must examine a large amount of data in both schemas – error-prone and labour process

---

Schema matching system

Goal: to **automatically** create a set of correspondences/matches between the two schemas $S$ and $T$

- Given two schemas $S$ and $T$ and being **schema elements** the table/attribute names in the schema,
  - a **correspondence $A \leadsto B$** states that a set of elements $A$ in $S$ maps to a set of attributes $B$ in $T$
  - Ex: for the target relation **Item**, there are the following correspondences:
    - **Products.title $\approx$ Item.title**
    - **Movie.year $\approx$ Item.year**
    - **$(Products.basePrice \times (1+ Locations.taxRate)) $ $\approx$ $ Item.price$**

- One solution is to provide a graphical interface to the designer; here we focus on how to **automatically create those correspondences** that are then validated by the user
Heuristics

- No single heuristic is guaranteed to yield accurate results

1. Examining the similarities between names of schema elements (e.g., releaseInfo of Aggregator schema and releaseDate or releaseCompany of schema Dvd-vendor)
2. Examining similarities between data values when available
3. Examine how attributes are used in queries, etc

None of the clues by itself is enough!

Example of the Need to Exploit Multiple Types of Information

<table>
<thead>
<tr>
<th>listed-price</th>
<th>contact-name</th>
<th>contact-phone</th>
<th>office</th>
<th>comments</th>
</tr>
</thead>
<tbody>
<tr>
<td>$250K</td>
<td>James Smith</td>
<td>(305) 729 0831</td>
<td>(305) 616 1822</td>
<td>Fantastic house</td>
</tr>
<tr>
<td>$320K</td>
<td>Mike Doan</td>
<td>(617) 253 1429</td>
<td>(617) 112 2315</td>
<td>Great location</td>
</tr>
<tr>
<td>......</td>
<td>......</td>
<td>......</td>
<td>......</td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>listed-price</th>
<th>contact-agent</th>
<th>extra-info</th>
</tr>
</thead>
<tbody>
<tr>
<td>$350K</td>
<td>(305) 634 9435</td>
<td>Beautiful yard</td>
</tr>
<tr>
<td>$230K</td>
<td>(617) 335 4243</td>
<td>Close to Seattle</td>
</tr>
</tbody>
</table>

- If use only names
  - contact-agent matches either contact-name or contact-phone

- If use only data values
  - contact-agent matches either contact-phone or office

- If use both names and data values
  - contact-agent matches contact-phone
Example of the Need to Exploit Multiple Types of Information

<table>
<thead>
<tr>
<th>listed-price</th>
<th>contact-name</th>
<th>contact-phone</th>
<th>office</th>
<th>comments</th>
</tr>
</thead>
<tbody>
<tr>
<td>$250K</td>
<td>James Smith</td>
<td>(305) 729 0831</td>
<td>(305) 616 1822</td>
<td>Fantastic house</td>
</tr>
<tr>
<td>$320K</td>
<td>Mike Doan</td>
<td>(617) 253 1429</td>
<td>(617) 112 2315</td>
<td>Great location</td>
</tr>
</tbody>
</table>

- If use only names
  - contact-agent matches either contact-name or contact-phone

- If use only data values
  - contact-agent matches either contact-phone or office

- If use both names and data values
  - contact-agent matches contact-phone

---

Example of the Need to Exploit Multiple Types of Information

<table>
<thead>
<tr>
<th>listed-price</th>
<th>contact-name</th>
<th>contact-phone</th>
<th>office</th>
<th>comments</th>
</tr>
</thead>
<tbody>
<tr>
<td>$350K</td>
<td></td>
<td>(206) 634 9435</td>
<td></td>
<td>Beautiful yard</td>
</tr>
<tr>
<td>$230K</td>
<td></td>
<td>(617) 335 4243</td>
<td></td>
<td>Close to Seattle</td>
</tr>
</tbody>
</table>

- If use only names
  - contact-agent matches either contact-name or contact-phone

- If use only data values
  - contact-agent matches either contact-phone or office

- If use both names and data values
  - contact-agent matches contact-phone
Components of a typical schema matching system

Matcher: predicts correspondences based on clues available in schema and data
Combiner: combines the predictions of the basic matchers into a single similarity matrix
Constraint enforcer: applies domain knowledge and constraints to prune the possible matches
Match selector: chooses the best match or matches from the similarity matrix

Example of similarity matrix

**BookVendor:**
- **Books** (ISBN, publisher, pubCountry, title, review)
- **Inventory** (ISBN, quantity, location)

**Distributor:**
- **Items** (code, name, brand, origin, desc)
- **InStore** (code, availQuant)

<table>
<thead>
<tr>
<th>attribute</th>
<th>Item</th>
<th>code</th>
<th>name</th>
<th>brand</th>
<th>origin</th>
<th>desc</th>
<th>InStore</th>
<th>code</th>
<th>availQuant</th>
</tr>
</thead>
<tbody>
<tr>
<td>ISBN</td>
<td>0.2</td>
<td>0.9</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0.1</td>
</tr>
<tr>
<td>title</td>
<td>0.25</td>
<td>0.1</td>
<td>0.8</td>
<td>0.1</td>
<td>0.15</td>
<td>0.2</td>
<td>0.15</td>
<td>0</td>
<td>0.15</td>
</tr>
<tr>
<td>review</td>
<td>0.15</td>
<td>0</td>
<td>0.6</td>
<td>0.1</td>
<td>0.55</td>
<td>0.05</td>
<td>0.05</td>
<td>0</td>
<td>0.05</td>
</tr>
<tr>
<td>ISBN</td>
<td>0.25</td>
<td>0.9</td>
<td>0</td>
<td>0</td>
<td>0.15</td>
<td>0.15</td>
<td>0.9</td>
<td>0</td>
<td>0.9</td>
</tr>
<tr>
<td>quantity</td>
<td>0</td>
<td>0.1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0.75</td>
<td>0</td>
<td>0.75</td>
</tr>
<tr>
<td>location</td>
<td>0.1</td>
<td>0.6</td>
<td>0.6</td>
<td>0.7</td>
<td>0.06</td>
<td>0.3</td>
<td>0.2</td>
<td>0</td>
<td>0.2</td>
</tr>
</tbody>
</table>
Creating schema mappings from matches

- Create the actual mappings from the matches (correspondences)
  - Find how tuples from one source can be translated into tuples in the other
- **Challenge:** there may be more than one possible way of joining the data
  
  Ex: To compute the value of `price` in the aggregator schema, may join the `Products` table with `Locations` table, using `saleLocID = lid`, and add the appropriate local tax given by `taxRate`, or join `Products` with `Movies` to obtain the origin of director, and compute the `price` based on the taxes in the director’s country of birth

Outline

- Matchers
- Combining match predictions
- Applying constraints and domain knowledge to candidate schema matches
- Match selector
- Applying machine learning techniques to enable the schema matcher to learn
- From matches to mappings
Components of a schema matcher

- **Matchers**
  - **Input:**
    - pair of schemas S1 and S2, with elements A and B, respectively
    - Additional available information, such as data instances or text descriptions
  - **Output:**
    - Similarity matrix that assigns to every pair of elements (Ai, Bj) a number between 0 and 1 predicting whether Ai corresponds to Bj
Classes of basic matchers

**Name-based matchers**: based on comparing names of schema elements

**Instance (data)-based matchers**: based on inspecting data instances
- Must look at large amounts of data
- Slower, but efficiency can be improved
- More precise

- For specific domains, it is possible to develop more specialized and effective matchers

Name-based matchers

- Compare the names of the elements, hoping that the names convey the true semantics of elements

- **Challenge**: to find effective *distance/similarity measures* reflecting the distance/similarity of element names
  - Names are never written in exactly the same way
  - One of the string matching algorithms studied can be used
Normalization

- Element names can be composed of **acronyms** or **short phrases** to express their meanings.
- Replaces a single token by several tokens that can be compared.
  - Element names should be normalized before applying distance measures.
- Some **normalization techniques**:
  - Split names according to certain delimiters (AgentAddress1 into Agent, Address and 1).
  - Expand known abbreviations (cust into customer).
  - Expand a string with its synonyms (price and cost).
  - Remove articles, propositions and conjunctions (and, in, at).

Instance (data) -based matchers

- Data instances, if available, convey the meaning of a schema element, more than its name.
  - Use them for predicting correspondences between schema elements.
- **Techniques**:
  1. **Recognizers**: employ dictionaries, regexps, or rules.
     - Develop a set of rules for inferring common types from the format of the data values, e.g., phone numbers, prices, zip codes, location names, etc.
  2. **Overlap matchers**
     - Examine the overlap of values between attributes.
  3. **Classifiers**: Text-field analysis
     - Use learning techniques.
Value overlap

- Measuring the overlap of values in the two elements
- Applies to categorical elements: whose values range in some finite domain (e.g., movie ratings, country name)

**Jaccard coefficient**: fraction of the values for the two elements that can be an instance for both of them
- Also defined as: conditional probability of a value being an instance of both elements given that it is an instance of one of them

\[
JaccardSim(e_1, e_2) = \Pr(e_1 \cap e_2 \mid e_1 \cup e_2) = \frac{|D(e_1) \cap D(e_2)|}{|D(e_1) \cup D(e_2)|}
\]

where \(D(e)\) is the set of values for element \(e\)

Example

- Use Jaccard measure to build a data-based matcher between DVD-VENDOR and AGGREGATOR

- AGGREGATOR **name** refers to DVD **title**, 
  DVD-VENDOR **name** refers to sale location, 
  DVD-VENDOR **title** refers to DVD title

- So, low score for (name, name), high score for (name, title)
Text classifiers

- Applies to elements whose values are longer texts (e.g., house descriptions)
  - Their values can vary drastically
  - The probability of finding the exact string for both elements is very low

- **Idea:** to compare the general topics these text fields are about
  - Builds classifiers on one schema and uses them to classify the elements of the other schema

---

**Text classifiers**

**Classifier for a concept C:** algorithm that identifies instances of C from those that are not
- Creates an internal model based on training examples, i.e., positive examples that are known to be instances of C and negative examples that are known not to be instances of C
- Given an example e, the classifier applies its model to decide whether e is an instance of C
- Assigns a probability to the prediction (confidence)

- Positive and negative examples are strings for text classifiers
- Examples of text classifiers: Naive Bayes classifier, Decision trees, Support Vector Machines
Common strategy of using classifiers

- For each element $s_i$ of schema $S$, want to train classifier $C_i$ to recognize instances of $s_i$
- To do this, need positive and negative training examples
  - take all data instances of $s_i$ (that are available) to be positive examples
  - take all data instances of other elements of $S$ to be negative examples
- Train $C_i$ on the positive and negative examples

Common strategy of using classifiers (cont.)

- Then, we can use $C_i$ to compute similarity score between $s_i$ and each element $t_j$ of schema $T$
- To do this, apply $C_i$ to data instances of $t_j$
  - for each data instance, $C_i$ produces a number in $[0,1]$ that is the confidence that the instance is indeed an instance of $s_i$
- Now, need to aggregate the confidence scores of all the instances (of $t_j$) to return a single confidence score (as the sim score between $s_i$ and $t_j$)
- A simple way to do so is to compute the average score over all instances of $t_j$
Using Classifiers: An Example

- **SCHEMA S**
  - current-showing
  - address: Lord of the Rings Madison WI Mountain View CA
  - phone: (608) 695 2311 (650) 277 1358

- **SCHEMA T**
  - name: ... Milwaukee WI ... Palo Alto CA Philadelphia PA
  - location

- Apply Classifier C to each data instance of T.location

- $s_i$ is address, $t_j$ is location
- Sim scores are 0.9, 0.7, and 0.5, respectively for the three instances of T.location ➔ return average score of 0.7 as sim score between address and location

Components of a schema matcher

- Match selector
- Constraint enforcer
- Combiner
- Matcher 1 ...
- Matcher n
Combining match predictions (1)

- Merges the similarity matrices output by matchers into a single one
  - Simple matchers: average, minimum, maximum
  - More complex: hand-crafted scripts or weighted-sum combiners

- If matching system uses k matchers to predict scores between element si of schema S and element tj of schema T, the score between these two elements computed by the combiner is:

\[
combined(i,j) = \left[ \frac{\sum_{m=1}^{k} matchScore(m,i,j)}{k} \right]
\]

where \(matchScore(m,i,j)\) is the score between si and tj produced by the mth matcher.

Example

Name-based matcher:
- name ≈ (name:1, title:0.2)
- releaseInfo ≈ (releaseDate:0.5, releaseCompany: 0.5)
- price ≈ (basePrice:0.8)

Instance-based matcher:
- name ≈ (name:0.2, title:0.8)
- releaseInfo ≈ (releaseDate:0.7)
- classification ≈ (rating:0.6)
- price ≈ (basePrice:0.2)

Average combiner:
- name ≈ (name:0.6, title:0.5)
- releaseInfo ≈ (releaseDate:0.6, releaseCompany:0.25)
- classification ≈ (rating:0.3)
- price ≈ (basePrice:0.5)
Combining match predictions (2)

Output: a similarity matrix that combines the predictions of the base matchers

- For every pair \((i, j)\), want a value between 0 and 1, \(\text{Combined}(i, j)\) that gives a single prediction about the correspondence between \(A_i\) and \(B_j\)

Possible types of combiners:
- Max used when we trust in a matcher that outputs a high value
- Avg used otherwise
- use hand-crafted scripts
  - e.g., if \(s_i\) is address, return the score of the data-based matcher otherwise, return the average score of all matchers
- multi-step combination functions and give weights to matchers
- the combiner itself can be a learner, which learns how to combine the scores of the matchers

Components of a schema matcher

1. Match selector
2. Constraint enforcer
3. Combiner
4. Matcher 1
5. ... Matcher n
Applying constraints and domain knowledge to candidate matches

- The designer may have domain-specific knowledge helpful in the process of schema matching
- Expressed as a set of constraints that enable pruning candidate matches
  - **Hard constraints**: must be applied; schema matcher will not output any match that violates them
  - **Soft constraints**: more heuristic nature; may be violated in some schemas; nb of violated should be minimized
  - A cost is associated to each constraint: infinite for hard constraints; any positive number for soft constraints
- Enforcer searches through the space of all match combinations produced by the combiner to find the one with the highest aggregated confidence score satisfying the constraints

**Example**

BookVendor:
- **Books**(ISBN, publisher, pubCountry, title, review)
- **Inventory**(ISBN, quantity, location)

Distributor:
- **Items**(code, name, brand, origin, desc)
- **InStore**(code, availQuant)

**Constraints:**
- C1: if $A \approx \text{Items.code}$, then $A$ is a key. Cost = $\infty$
- C2: If $A \approx \text{Items.desc}$, then any random sample of 100 instances of $A$ is such that $\text{Average(length}(A)) \geq 20$. Cost = 1.5
- C3: If more than half of the attributes of table $T$ match those of Table $V$, then $T \approx V$. Cost = 1
Algorithms for applying constraints to the similarity matrix

- Applying constraints with A* search
  - Guarantees to find the optimal solution
  - Computationally more expensive
- Applying constraints with local propagation
  - Faster
  - May get stuck in a local minimum

Components of a schema matcher

![Components of a schema matcher diagram]
Match selector

Input: similarity matrix for schemas S and T
Output: a schema match or the top few matches

- Simplest selection strategy: thresholding
  - all attribute pairs with sim not less than a threshold are returned as matches
  - Ex: given the matrix
    - name = <title: 0.5>
    - releaseInfo = <releaseDate: 0.6>
    - classification = <rating: 0.3>
    - price = <basePrice: 0.5>
    - given threshold 0.5, return matches name = title, etc.

- More sophisticated strategies: compute the top few matches

---

One of the algorithms behind

- The match selection problem can be formulated as an instance of finding a stable marriage
  - Elements of S1: men; elements of S2: women
  - Sim(i,j): the degree to which Ai and Bj desire each other
- Goal: find a stable match (marriage) between men and women
  - A match is unstable if there are Ai -> Bj and Ak -> Bl, such that
    - sim(i, l) > sim(i, j) and sim(i, l) > sim(k, l)
  - If these couples existed then Ai and Bl would want to be matched together
- To produce a schema match without unhappy couples do:
  - Match = {}
  - Repeat:
    - Let (i, j) be the highest value in sim such that Ai and Bj are not in Match
    - Add Ai ↔ Bj to Match
Outline

✓ Basic matchers
✓ Combining match predictions
✓ Applying constraints and domain knowledge to candidate schema matches
✓ Match selector
➢ Applying machine learning techniques to enable the schema matcher to learn
  ▪ From matches to mappings

Applying machine learning techniques to enable the schema matcher to learn
[SIGMOD01]

■ Schema matching tasks often repetitive
  ▪ Creating schema matches from sources to mediator in the same domain
■ Same concepts tend to re-occur for a given domain
  ▪ So the designer can create schema matches more quickly over time
■ Questions:
  ▪ Can the schema matching also improve over time?
  ▪ Or: can a schema matcher learn from previous experience?
➢ Machine learning techniques can be applied to schema matching, thus enabling the matcher to improve over time
Learning to match

- Suppose $n$ data sources $s_1, \ldots, s_n$ whose schemas must be matched into the mediated schema $G$

- Goal:
  - To train the system by manually providing it with schema matches on a small nb of data sources (e.g., $s_1, \ldots, s_m$, where $m$ is much smaller than $n$)
  - The system generalizes from the training examples so that it is able to predict matches for sources $s_{m+1}, \ldots, s_n$

Training the base learners

- Learning classifiers for elements in the mediated schema
  - The classifier for an element $e$ in the mediated schema examines an element in a source schema and predicts whether it matches $e$

- To create classifiers, employ a machine learning algorithm
  - Each machine learning algorithm typically considers only one aspect of the schema and has advantages/inconveniences
  - So, use a multi-strategy learning technique
Multi-strategy learning

Training phase:
- Employ a set of learners $l_1, \ldots, l_k$
  - Each base learner creates a classifier for each element $e$ of the mediated schema from its training examples
  - Training examples are derived using the semantic matches between the mediated schema and the training data sources
- Use a meta-learner to learn weights for the different base learners
  - For each element $e$ of the mediated schema and base learner $l$, the meta-learner computes a weight $w_{e,l}$
    - It knows how to do that, because we are working with training examples

Matching phase:
- When presented with a schema $S$ (of a data source) whose elements are $e_1', \ldots, e_t'$. Let $p_{e_1}(e')$ be the prediction of learner $l$ on whether $e'$ matches $e$
- Combine the learners:
  $$ p_e(e') = \sum_{j=1}^{k} w_{e,j} \cdot p_{e,j}(e') $$

Components of the LSD system
(Learning Source Descriptions)

![Diagram of LSD system components]
Training phase

1. Manually specify mappings for some sources
2. Extract source data
3. Create training data for each base learner
4. Train the base learners
5. Train the meta-learner

Base Learners

Input
- schema information: name, proximity, structure, ...
- data information: value, format, ...

Output
- prediction weighted by confidence score

Examples
- Rule-based learner
  - day-phone ≈ (agent-phone, 0.9), (description, 0.1)
- Naive Bayes learner
  - “Kent, WA” ≈ (address, 0.8), (name, 0.2)
  - “Great location” ≈ (description, 0.9), (address, 0.1)
Rule-based learner

- Examines a set of training examples and computes a set of rules, based on features of the training examples, that can be applied to test instances
  - Rules can be represented as logical formulae or as decision trees
  - Works well in domains where the set of rules can accurately characterize instances of the class (e.g., identifying elements that adhere to certain formats)
  - When the learned rules are applied to an instance, return 1 if the instance satisfies the rules and 0 otherwise

Example: rule-based learner for identifying phone numbers (1)

- Positive and negative examples of phone numbers:

<table>
<thead>
<tr>
<th>Example</th>
<th>instance?</th>
<th># of digits</th>
<th>position of (</th>
<th>position of )</th>
<th>position of –</th>
</tr>
</thead>
<tbody>
<tr>
<td>(608)435-2322</td>
<td>yes</td>
<td>10</td>
<td>1</td>
<td>5</td>
<td>9</td>
</tr>
<tr>
<td>(60)445-284</td>
<td>no</td>
<td>9</td>
<td>1</td>
<td>4</td>
<td>8</td>
</tr>
<tr>
<td>849-7394</td>
<td>yes</td>
<td>7</td>
<td>-</td>
<td>-</td>
<td>4</td>
</tr>
<tr>
<td>(1343) 429-441</td>
<td>no</td>
<td>10</td>
<td>1</td>
<td>6</td>
<td>10</td>
</tr>
<tr>
<td>43 43 (12 1285)</td>
<td>no</td>
<td>10</td>
<td>5</td>
<td>12</td>
<td>-</td>
</tr>
<tr>
<td>5549902</td>
<td>no</td>
<td>7</td>
<td>-</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>(212) 433 8842</td>
<td>yes</td>
<td>10</td>
<td>1</td>
<td>5</td>
<td>-</td>
</tr>
</tbody>
</table>
Example: rule-based learner for identifying phone numbers (2)

- Common method to learn rules is to create a decision tree

![Decision Tree Example]

- Encodes rules such as:
  - If i has 10 digits, a '(' in position 1 and ')’ in position 5, then yes
  - If i has 7 digits, but no ‘-’ in position 4, then no

Naive Bayes Learner

- Examines the tokens of a testing instance and assigns to the instance the most likely class given the occurrences of tokens in the training set
  - Effective for recognizing text fields
  - Example instances are: values of a column, names of attributes, or descriptions of schema elements
  - Given a test instance, the learner converts it into a bag of tokens
    - Tokens generated by parsing and stemming words and symbols in the instance

  Ex: “RE/MAX Greater Atlanta Affiliates of Roswell” becomes “re/max greater atlanta affli of roswell”
Naive Bayes learner at work

Given that \( c_1, \ldots, c_n \) are elements of the mediated schema, the learner is given a test instance \( d = \{w_1, \ldots, w_k\} \), where \( w_i \) are tokens, to classify.

**Goal:** assign \( d \) to the element \( c_d \) with the highest posterior probability given \( d \):

\[
C_d = \arg \max_{c_i} P(C_i | d)
\]

Using the Bayes rule:

\[
P(C_i | d) = \frac{P(d | C_i) P(C_i)}{P(d)}
\]

\[
=> C_d = \arg \max_{c_i} \left[ P(d | C_i) P(C_i) / P(d) \right]
\]

\[
= \arg \max_{c_i} \left[ P(d | C_i) P(C_i) \right]
\]

\( P(d | C_i) \) and \( P(C_i) \) must be estimated from the training data.

Estimation of \( P(d | c_i) \) and \( P(c_i) \)

- \( P(c_i) \) is approximated by the portion of the training instances with label \( c_i \).
- To compute \( P(d | c_i) \) assume that the tokens \( w_j \) appear in \( d \) independently of each other given \( c_i \).

\[
P(d | c_i) = P(w_1 | c_i) P(w_2 | c_i) \ldots P(w_k | c_i)
\]

\[
P(w_j | c_i) = \frac{n(w_j, c_i)}{n(c_i)}, \text{ where}
\]

- \( n(w_j, c_i) \): number of times token \( w_j \) appears in all training instances with label \( c_i \).
- \( n(c_i) \): total number of tokens in the training instances with label \( c_i \).
Conclusion of Naive Bayes learner

- Naive Bayes performs well in many domains in spite of the fact the independence assumption is not always valid.
- Works best when:
  - There are tokens strongly indicative of the correct label, because they appear in one element and not in the others.
    - Ex: “beautiful”, “fantastic” to describe houses
  - There are only weakly suggestive tokens, but many of them
- Doesn’t work well
  - Short or numeric fields (such as color, zip code, number of bathrooms)

Components of the LSD system (Learning Source Descriptions)
Training the meta-learner (1)

- Learns the weights to attach to each of the base learners, from the training examples
  - Can be different for every mediated-schema element
- How does it work?
  - Asks the base learners for predictions on training examples
  - Judges how well each learner performed in providing the prediction for each mediated-schema element
  - Assigns to each combination (mediated schema element $c_i$, base learner $L_j$) a weight indicating how much it trusts that learner predictions regarding $c_i$

Training the meta-learner (2)

- Given an element $e_s$ of a source schema $s$
- Training example describes properties of $e_s$ and predictions of learners
- Training data for the meta-learner have the form:
  - $(f_1, \ldots, f_m, p_1, \ldots, p_n, p^\ast)$, where $f_1, \ldots, f_m$ are features of $e_s$ (average length, distinct values in the field), $p_i$ is the prediction that base learner $L_i$ made for the pair $(e_s, c_i)$, and $p^\ast$ is the correct prediction for $e_s$
- Meta-learner can use a learning algorithm (ex: linear regression) to compute the appropriate weights
  - Ex: it can learn a rule that gives the Naive-Bayes more weight when the attribute has long texts
Charlie comes to town

Find houses with 2 bedrooms priced under 300K

Data Integration

Find houses with 2 bedrooms priced under 300K
Example

**Mediated schema**

- address
- price
- agent-phone
- description

**Schema of realestate.com**

- location
- listed-price
- phone
- comments

**Learned hypotheses**

**Rule-based learner**

- If “phone” occurs in the name => agent-phone

**Naive-bayes learner**

**Training the Learners**

**Mediated schema**

- address
- price
- agent-phone
- description

**Schema of realestate.com**

- location
- listed-price
- phone
- comments

**realestate.com**

- <location> Miami, FL </>  
- <listed-price> $250,000 </>  
- <phone> (305) 729 0831 </>  
- <comments> Fantastic house </>

- <location> Boston, MA </>  
- <listed-price> $110,000 </>  
- <phone> (617) 253 1429 </>  
- <comments> Great location </>

**Rule-based Learner**

- (location, address)  
- (listed-price, price)  
- (phone, agent-phone)  
- (comments, description)  

**Naive Bayes Learner**

- (“Miami, FL”, address)  
- (“$ 250,000”, price)  
- (“(305) 729 0831”, agent-phone)  
- (“Fantastic house”, description)
Applying the Learners

<table>
<thead>
<tr>
<th>Schema of homes.com</th>
<th>Mediated schema</th>
</tr>
</thead>
<tbody>
<tr>
<td>area</td>
<td>address</td>
</tr>
<tr>
<td>day-phone</td>
<td>price</td>
</tr>
<tr>
<td>extra-info</td>
<td>agent-phone</td>
</tr>
<tr>
<td>extra-info</td>
<td>description</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th></th>
<th>Rule Learner</th>
<th>Meta-Learner</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>&lt;area&gt;Seattle, WA&lt;/area&gt;</td>
<td>Naive Bayes</td>
<td>(address,0.8), (description,0.2)</td>
<td></td>
</tr>
<tr>
<td>&lt;area&gt;Kent, WA&lt;/area&gt;</td>
<td>......</td>
<td>(address,0.6), (description,0.4)</td>
<td></td>
</tr>
<tr>
<td>&lt;area&gt;Austin, TX&lt;/area&gt;</td>
<td>Meta-Learner</td>
<td>(address,0.7), (description,0.3)</td>
<td></td>
</tr>
<tr>
<td>&lt;day-phone&gt;(278) 345 7215&lt;/day-phone&gt;</td>
<td>Naive Bayes</td>
<td>(address,0.7), (description,0.3)</td>
<td></td>
</tr>
<tr>
<td>&lt;day-phone&gt;(617) 335 2315&lt;/day-phone&gt;</td>
<td>......</td>
<td>(agent-phone,0.9), (description,0.1)</td>
<td></td>
</tr>
<tr>
<td>&lt;day-phone&gt;(512) 427 1115&lt;/day-phone&gt;</td>
<td>......</td>
<td>(address,0.6), (description,0.4)</td>
<td></td>
</tr>
</tbody>
</table>

<extra-info>Beautiful yard</extra-info> |  |
<extra-info>Great beach</extra-info> |  |
<extra-info>Close to Seattle</extra-info> |  |

Recap. Multi-strategy learning

- **Training phase:**
  - Employ a set of learners $l_1, \ldots, l_k$
    - Each base learner creates a classifier for each element $e$ of the mediated schema from its training examples.
  - Use a meta-learner to learn weights for the different base learners.
    - For each element $e$ of the mediated schema and base learner $l$, the meta-learner computes a weight $w_{e,l}$.

- **Matching phase:**
  - When presented with a schema $S$ whose elements are $e_1', \ldots, e_s'$.
  - Apply the base learners to $e_1', \ldots, e_s'$. Let $p_{e,l}(e')$ be the prediction of learner $l$ on whether $e'$ matches $e$.
  - Combine the learners:
    $$p_e(e') = \sum_{j=1}^k w_{e,lj} * p_{e,lj}(e')$$
Observations

- LSD provides attribute correspondences (schema matches), but not schema mappings
- Many similar systems: COMA, COMA++, Falcon-AO, ...

Another example

Consider the Naive-Bayes learner and the following mediator and source schemas:

Mediator: LibraryItems(name, description)

S1: Books(title, text)

S2: Magazines(nameM, descM)

S3: Items(N, T)

The following correspondences (between the mediator and S1, S2) were performed by hand:

Books.title = LibraryItems.name
Books.text = LibraryItems.description
Magazines.nameM = LibraryItems.name
Magazines.descM = LibraryItems.description
Consider that the data values shown are the result of a pre-processing step where parsing, stemming, and deletion of stop-words were performed.

Data values of source S1 and S2 (training data):
Books (‘jane eyre’, ‘beautiful interesting love story’)
Books (‘peter pan’, ‘children adventure’)
Books (‘star war’, ‘success science fiction history’)
Magazines (‘hola’, ‘famous spanish magazine picture people’)
Magazines (‘national geograph’, ‘beautiful picture nature’)
Magazines (‘computer world’, ‘interesting technical article computer’)

Data values of source S3 (testing data):
Items (‘national geo’, ‘interesting pictures articles famous’)

Compute the correspondences returned by the Naive Bayes learner for Source S3 taking into account the corresponding data values. Show all intermediate steps and results.

We can consider that we apply stemming to the words: geography, geograph, pictures, articles.

Training examples:
Words/tokens with label “name”: {jane, eyre, peter, pan, star, war, hola, national, geo, computer, world} -> 11
Words/tokens with label “description: {beautiful (2), interesting (2), love, story, children, adventure, success, science, fiction, history, famous, spanish, magazine, picture (2), people, nature, technical, article, computer} -> 22

Test examples:
N: {national, geo}
T: {interesting, picture, article, famous}

\[
P(N, \text{name}) = P(N|\text{name}).P(\text{name})
\]

\[
P(N, \text{description}) = P(N|\text{description}).P(\text{description})
\]

\[
P(T, \text{name}) = P(T|\text{name}).P(\text{name})
\]

\[
P(T, \text{description}) = P(T, \text{description}).P(\text{description})
\]
\[ P(\text{name}) = \text{fraction of training examples with label "name"} = \frac{3}{6} = 0.5 \]
\[ P(\text{description}) = \text{fraction of training examples with label "description"} = \frac{3}{6} = 0.5 \]
\[ P(\text{national}|\text{name}),P(\text{geo}|\text{name}) = \]
\[ = \frac{n(\text{national}, \text{name})}{n(\text{name})} \times \frac{n(\text{geo}, \text{name})}{n(\text{name})} = \frac{1}{11} \times \frac{1}{11} \]
\[ P(N, \text{name}) = \frac{1}{11} \times \frac{1}{11} \times 0.5 = 0.0041 \]
\[ P(\text{national}|\text{description}),P(\text{geo}|\text{description}) = 0 \]
\[ P(N|\text{description}) = 0 \]
\[ P(\text{interesting}|\text{name}),P(\text{picture}|\text{name}),P(\text{article}|\text{name}),P(\text{famous}|\text{name}) = \]
\[ = \frac{n(\text{interesting}, \text{name})}{n(\text{name})} \times \frac{n(\text{picture}, \text{name})}{n(\text{name})} \times \frac{n(\text{article}, \text{name})}{n(\text{name})} \times \frac{n(\text{famous}, \text{name})}{n(\text{name})} = 0 \]
\[ P(T, \text{name}) = 0 \]

Conclusion:
The correspondences are: N ~ name and T ~ description

\[ P(\text{interesting}|\text{description}),P(\text{picture}|\text{description}),P(\text{article}|\text{description}),P(\text{famous}|\text{description}) = \]
\[ = \frac{n(\text{interesting}, \text{description})}{n(\text{description})} \times \frac{n(\text{picture}, \text{description})}{n(\text{description})} \times \frac{n(\text{article}, \text{description})}{n(\text{description})} \times \frac{n(\text{famous}, \text{description})}{n(\text{description})} = \]
\[ = \frac{2}{22} \times \frac{2}{22} \times \frac{1}{22} \times \frac{1}{22} = 0.000017 \]
\[ P(T, \text{description}) = \frac{4}{(22 \times 22 \times 22 \times 22)} \times 0.5 = 0.00000854 \]

Conclusion:
The correspondences are: N ~ name and T ~ description
Outline

- Base matchers
- Combining match predictions
- Applying constraints and domain knowledge to candidate schema matches
- Match selector
- Applying machine learning techniques to enable the schema matcher to learn

> From matches to mappings

From matches to mappings

- Until now: Schema matches are correspondences between the source and the target schemas
- Now: specifying the operations to be performed on the source data so that they can be transformed into the target data
  - Use DBMS as transformation engines for relational data

- Creating mappings becomes a process of query discovery
  - Find the queries, using joins, unions, filtering, aggregates, that correctly transform the data into the desired schema
  - Algorithm that explores the space of possible schema mappings consistent with the matches
  - Used in the CLIO system
User interaction

- Creating mappings is a complex process
- System generates the mapping expressions automatically
  - The possible mappings are automatically produced using the semantics conveyed by constraints such as foreign keys.
- System shows the designer example data instances so that she can verify which are the right mappings

Motivating example

- Question:
  - Union professor salaries with employee salaries, or
  - Join salaries computed from the two correspondences?
Possible mappings (1)

Not clear which join path to choose for mapping f1!

- If attribute ProjRank is a foreign key of the relation PayRate, then the mapping would be:
  \[
  \text{SELECT } P.HrRate * W.Hrs \\
  \text{FROM PayRate } P, \text{ WorksOn } W \\
  \text{WHERE } P.Rank = W.ProjRank
  \]
- If attribute ProjRank is not a foreign key of the relation PayRate. Instead, the name attribute of WorksOn is a foreign key of Student and the Yr attribute of Student is a foreign key of PayRate (the salary depends on the year of the student). Then, the following query should be chosen:
  \[
  \text{SELECT } P.HrRate * W.Hrs \\
  \text{FROM PayRate } P, \text{ WorksOn } W, \text{ Student } S \\
  \text{WHERE } W.Name = S.Name \text{ AND } S.Yr = P.Rank
  \]

Possible mappings (2)

- One interpretation of \(f_2\) is that values produced from \(f_1\) should be joined with those produced by \(f_2\)
  - Then, most of the values in the source DB would not be mapped to the target
- Another interpretation: there are two ways of computing the salary of employees: one applying to professors and another to other employees. The corresponding mapping is:
  \[
  \text{SELECT } P.HrRate * W.Hrs \\
  \text{FROM PayRate } P, \text{ WorksOn } W, \text{ Student } S \\
  \text{WHERE } W.Name=S.Name \text{ AND } S.Yr = P.Rank
  \text{UNION ALL}
  \text{SELECT } \text{Salary} \\
  \text{FROM Professor}
  \]
Principles to guide the mapping construction

- If possible, all values in the source appear in the target
  - Choose a union rather than a join
- If possible, a value from the source should only contribute once to the target
  - Associations between values that exist between source items should not be lost
  - Use a join rather than a cartesian product to compute the salary value using $f_1$

Possible mappings (3)

- Consider a few more correspondences:
  - $f_3$: Professor(Id) $\approx$ Personnel(Id)
  - $f_4$: Professor(Name) $\approx$ Personnel(Name)
  - $f_5$: Address(Addr) $\approx$ Personnel(Addr)
  - $f_6$: Student(Name) $\approx$ Personnel(Name)
- They fall into two candidate sets of correspondences:
  - $f_2, f_3, f_4$ and $f_5$: map from Professor to Personnel
  - $f_1, f_6$: map from other employees to Personnel
- The algorithm explores the possible joins within every candidate set and considers how to union the transformations corresponding to each candidate set.
Possible mappings (4)

f3: Professor(Id) $\approx$ Personnel(Id)
f4: Professor(Name) $\approx$ Personnel(Name)
f5: Address(Addr) $\approx$ Personnel(Addr)
f6: Student(Name) $\approx$ Personnel(Name)

Most reasonable mapping is:

```sql
SELECT P.Id, P.Name, P.Sal, A.Addr
FROM Professor P, Address A
WHERE A.Id = P.Id
UNION ALL
SELECT NULL as ID, S.Name, P.HrRate*W.Hrs, Null as Addr
FROM Student S, PayRate P, WorksOn W
WHERE S.name=W.name AND S.Yr = P.Rank
```

Possible mappings (5)

f3: Professor(Id) $\approx$ Personnel(Id)
f4: Professor(Name) $\approx$ Personnel(Name)
f5: Address(Addr) $\approx$ Personnel(Addr)
f6: Student(Name) $\approx$ Personnel(Name)

But this one is also possible:

```sql
SELECT NULL as ID, NULL as Name, NULL as Sal, Addr
FROM Address A
UNION ALL
SELECT P.Id, P.Name, P.Sal, NULL as Addr
FROM Professor P
UNION ALL
SELECT NULL as ID, S.name as Name, NULL as Sal, NULL as Addr FROM Student S
```
Query discovery algorithm - Goal

Eliminates unlikely mappings from the large search space of candidate mappings and identifies correct mappings a user might not otherwise have considered.

Query discovery algorithm - Characteristics

- **Is interactive**
  - Explores the space of possible mappings and proposes the most likely ones to the user
- **Accepts user feedback**
  - To guide it in the right direction
- **Uses heuristics**
  - Can be replaced by better ones if available
Query discovery algorithm – Input

- Set of correspondences $M = \{f_i: (A_i \approx B_i)\}$
  - $A_i$: set of attributes in source $S$
  - $B_i$: an attribute of the target $T$
- Possible filters on source attributes
  - Range restriction on an attribute, aggregate of an attribute, etc

Query discovery algorithm – 1st phase

- Create all possible candidate sets $\nu$ (subsets of $M$), that contain at most one correspondence per attribute of $T$
  - Each candidate set represents one way of computing the attributes of $T$
  - If a set covers all attributes of $T$, it is called complete set
  - Elements of $\nu$ do not need to be disjoint
    - The same correspondence can be used in multiple ways to compute $T$
Example

- Given the correspondences:
  \[ f_1 : S1.A \approx T.C \]
  \[ f_2 : S2.A \approx T.D \]
  \[ f_3 : S2.B \approx T.C \]
- Then the complete candidate sets are:
  \[ \{ \{ f_1, f_2 \}, \{ f_2, f_3 \} \} \]
- The singleton sets \{f_1\}, \{f_2\} and \{f_3\} are also candidate sets.

Query discovery algorithm – 2nd phase

- Consider the candidate sets in \( \nu \) and search for the best set of joins within each candidate set
  - Considering a candidate set \( v \) in \( \nu \) and suppose all correspondences \( (A_i \approx B_i) \in v \)
  - Then, search for join paths connecting distinct relations mentioned in \( A_i \) for all correspondences in \( v \), using the following:

  **Heuristic:**
  - A join path can be either:
    - A path through foreign keys
    - A path proposed by inspecting previous queries on \( S \), or
    - A path discovered by mining the data for joinable columns in \( S \)
Query discovery algorithm – 2nd phase

- The set of candidate sets in $\mathbf{\mathbf{\nu}}$ for which we find join paths is denoted by $\mathbf{\mathbf{\zeta}}$. When there are multiple join paths, use the following for selecting join paths:

  **Heuristic:**
  - Prefer paths through foreign keys.
  - If there are multiple such paths, choose one that involves an attribute on which there is a filter in a correspondence, if it exists.
  - To further rank paths, favor the join path where the estimated difference between the outer join and inner join is the smallest
  - Favors joins with the least number of dangling tuples

Query discovery algorithm – 3rd phase

- Examine the candidate sets in $\mathbf{\mathbf{\zeta}}$, and tries to combine them by union so they cover all the correspondences in $\mathbf{\mathbf{M}}$.
- Search for covers of the correspondences
  - A subset $\mathbf{\mathbf{T}}$ of $\mathbf{\mathbf{\zeta}}$ is a **cover** if it includes all the correspondences in $\mathbf{\mathbf{M}}$ and it is minimal (cannot remove a candidate set from $\mathbf{\mathbf{T}}$ and still obtain a cover).

Example:

- $\mathbf{\zeta} = \{\{f_1, f_2\}, \{f_2, f_3\}, \{f_1\}, \{f_2\}, \{f_3\}\}$
- Possible covers include
  - $\mathbf{\mathbf{T}}_1 = \{\{f_1\}, \{f_2, f_3\}\}$
  - $\mathbf{\mathbf{T}}_2 = \{\{f_1, f_2\}, \{f_2, f_3\}\}$. 
Query discovery algorithm – 3rd phase

- If there are multiple possible covers, use the following:

**Heuristic:**
- Choose the cover with the smallest nb of candidate sets (a simpler mapping should be more appropriate)
- If there is more than one with the same nb of candidate sets, choose the one that includes more attributes of T (to cover more of that schema)

Query discovery algorithm – 4th phase

Creates a schema mapping expression as an SQL query
- First creates an **SQL query for each candidate set** in the selected cover and then **unions** them

**Example:** Suppose \( \mathbf{v} \) is a candidate set:
- Attributes of T and correspondence functions mentioned in \( \mathbf{v} \) are put in the **SELECT** clause
- Each of the relations in the join paths found for \( \mathbf{v} \) are put in the **FROM** clause
- The corresponding join predicates are put in the **WHERE** clause
- Any filters associated with the correspondences in \( \mathbf{v} \) are also added to the **WHERE** clause
- Finally takes the union of the queries for each candidate set in the cover
Example

Source schema:
Hired(ID, Name, Salary)
Temporary(Name, Category), Category: FK(CategoryRate)
CategoryRate(Category, HrRate)
Project(Acronym, Category), Category: FK(CategoryRate)
Works(Name, Acronym, Hours), Name: FK(Temporary), Acronym: FK(Project)

Target schema:
People(ID, Name, Salary)

Correspondences:
C1 : Hired.Salary ≈ People.Salary
C2 : Works.Hours, CategoryRate.HrRate ≈ People.Salary
C3 : Hired.ID ≈ People.ID
C4 : Hired.Name ≈ People.Name
C5 : Temporary.Name ≈ People.Name

Example – phase 1

Phase 1:

Create all possible candidate sets \( v \) (subsets of \( M \)), that contain at most one correspondence per attribute of the target schema

\[
V = \{
\{C1, C3, C4\}, \{C1, C3, C5\}, \{C2, C3, C4\}, \{C2, C3, C5\},
\{C1, C3\}, \{C1, C4\}, \{C1, C5\}, \{C2, C3\}, \{C2, C4\}, \{C2, C5\}, \{C3, C5\}, \{C3, C4\},
\{C1\}, \{C2\}, \{C3\}, \{C4\}, \{C5\}
\}
\]

Complete set if it covers all attributes of the target schema:

\[
\{C1, C3, C4\}, \{C1, C3, C5\}, \{C2, C3, C4\}, \{C2, C3, C5\}
\]
Example – phase 2

Phase 2:

From [Miller00]: “… prunes from the set of potential candidate sets those that cannot be mapped into a good query. In particular, if the value correspondences in the potential candidate set map values from several source relations, we need to find a way of joining the tuples between these relations.”

The only correspondence that satisfies the condition in the second sentence is C2.

There are two possible join paths:

J1) Works |X| Temporary |X| CategoryRate

J2) Works |X| Project |X| CategoryRate

Heuristic 2 does not help to decide between the two, because we do not have any additional information.

So, the algorithm shows to the user both results of Phase 2: the one obtained with J1 and the one obtained with J2. It is up to him/her to choose the best one.

G = {
    {C1, C3, C4},
    {C1, C3}, {C1, C4}, {C2, C5}, {C3, C4},
    {C1}, {C2}, {C3}, {C4}, {C5}
}

Example – phase 3

Phase 3:

Search for covers (cover all correspondences and are minimal).

Cov1: { {C1, C3, C4}, {C2, C5} }
Cov2: { {C1, C3, C4}, {C2}, {C5} }
Cov3: { {C1}, {C2}, {C3}, {C4}, {C5} }
Cov4: { {C1, C3}, {C1, C4}, {C2, C5} }
Cov5: { {C1, C3}, {C2}, {C1, C4} }
Cov6: { {C1, C3}, {C2}, {C4}, {C5} }

…

Using the Heuristic, we choose Cov1: { {C1, C3, C4}, {C2, C5} } because it has the smallest number of candidate sets.
Example – phase 4

Assuming join path J1, the resulting query is:

```sql
SELECT ID, Name, Salary
FROM Hired h
UNION
SELECT Null, w.Name, r.HrRate * w.getHours Salary
FROM Works w, CategoryRate r, Temporary t
WHERE w.Name = t.Name
AND t.Category = r.Category
```

Assuming join path J2, the resulting query is:

```sql
SELECT ID, Name, Salary
FROM Hired
UNION
SELECT null ID, w.Name, w.getHours*cr.HrRate
FROM CategoryRate cr, Works w, Project p
WHERE w.Acronym = p.Acronym
AND p.Category = cr.Category
```

Next lecture

- Introduction to Data Cleaning