Mapping Maintenance for Data Integration Systems

R. McCann, B. AlShebli, Q. Le, H. Nguyen, L. Vu, A. Doan
University of Illinois at Urbana-Champaign
VLDB 2005

Laurent Charlin

October 26, 2005

Outline

- Problem Definition
- Previous Work in the domain & Background information
- Maveric, automatic mapping verification system
  - Sensor Ensemble
  - Perturber
  - Multi-Source Trainer
  - Filter
- Results from the paper
- Critique & Discussion

Overall

- They Assume a typical schema integration instance

![Diagram showing a mediated schema with source schemas and wrappers connected to websites like homeseekers.com, yahoo.com, and windermere.com]
Problem Definition

Statement

- They want to solve the Semantic mapping verification problem (i.e., answer the question, Is a given mapping broken?)
  - Assume that the Semantic mapping has been done
- Motivation: They have found that the dominating cost is often the mapping maintenance (detect and repair).

Previous Work

Background

- The authors have a strong AI (machine learning) background
- They are very active in this domain
- From Doan’s Ph.D. thesis "First, it introduced machine learning as an indispensable component of matching solutions. Second, it articulated a multi-component, highly extensible architecture for schema matching. Third, it showed how to learn from past matching efforts (to improve accuracy of subsequent matching tasks)."

Some Perspective

- First, Regression tester, relies on formatting regularities
- Kushmerick (2000), RAPTURE system, syntactic only
  - Could be 1 sensor in the current system
- Lerman et al. (2003), syntactic measure as well (results in the experiments section).
  - Learned structural information
  - Positive Data only
Previous Work

Some Perspective

- Activity monitoring (for example, Fawcett et al., '99)
- It might be formulated as a stream problem
  - Data is continuously arriving from querying the sources
  - You even have some control on the stream since you’re controlling the queries.

Typical Machine Learning - Online learning algorithm

- Given a \( m \) experts (sensors)
- At each time step (iteration)
  - the sensors predict \( \text{score}_j \in [0, 1] \)
  - the learner, based on all the \( \text{score}_j \) predicts \( \text{score}_{\text{cum}} \in [0, 1] \)
  - compare \( \text{score}_{\text{cum}} \) with the actual label of the example (update the sensor weights based on this).
- In verification (testing): You simply predict with the learned weights
- (this is taken from Robert Schapire's lectures)

Back to the current problem
1 - The Sensors

- Computational modules which capture specific characteristics of a source $S$
- Idea
  1. Train them on data from $S$
  2. Deploy them to monitor the data returned by queries

1- The Sensors

- Generate examples from querying "valid" semantic mappings
- Two types of parameters to learn
  - The parameters of each sensor (Gaussian mean and variance)
  - The weight of the sensors (in the Winnow algorithm)

1 - Winnow Algorithm

Train the Sensor Combiner

Input: examples $R_0,...,R_l$ labeled with + or - alarm threshold sensors $S_0,...,S_l$ (already trained on $R_0,...,R_l$)
Output: sensor weights $w_0,...,w_l$

1. Initialize each weight $w_i$ to 1
2. Repeat for each example $R_i$
   - for each sensor $S_j$ score $i$ the score of $S_j$ when applied to $R_i$
   - $\text{score}_i = \text{the combined score of all sensors using } w_0,...,w_l$
   - if (score $>_0$ and $R_i$ label = +) // false alarm
     - $w_i = w_i * 2$ for each score $< 0$
   - else if (score $<_0$ and $R_i$ label = +) // raised alarm
     - $w_i = w_i / 2$ for each score $> 0$
   - until a stopping criterion is reached
3. Return $w_0,...,w_l$

- Final classifier is given by

$$\text{vote}_{\text{valid}} = \sum_{i=0}^{m} w_i \times \text{score}_i$$

$$\text{vote}_{\text{invalid}} = \sum_{i=0}^{m} w_i \times (1 - \text{score}_i)$$
1 - The Sensor Types

- Value Sensors
  - Monitor features of attributes
  - Data modeled according to a Gaussian distribution
  - Density Scoring $score_v = 1 - P(v)$
  - Normalized Density scoring $score_v = Pr[P(v') \geq P(v)]$

- Trend Sensors
  - Work much like the Value sensors

- Layout Sensors
  - Monitors the HTML layouts

- Constraint Sensors
  - Monitors pre-defined attribute constraints

2 - Perturbation

- Problem: No Negative Example
- Solution: Generate negative examples from current source (S) data
- Corrolary: They are also trying to generate more diversified examples
- They Simulate the following situations
  - Change in the Source Query Interface
  - Change Source Data
  - Change the Presentation Format
- In training they incorporate this new data (both positive and negative) into the examples $R$.

- The addition of positive (invalid data) changed the way the score is calculated
- It’s due to the fact that they use Gaussian modeling of the data (they cannot incorporate both positive and negative examples in their distributions).
- They have two Gaussians
- $score_{cum} = score_- / (score_- + score_+)$
3 - Multi-Source training

- Usually you have to train on a single source $S$
- What about using other sources $S'$ from the same domain which have equivalent attributes
- Example (two attributes which are tied by the semantic schema):
  - $S$: price $185,000$
  - $S'$: amount $185,000USD$

4 - Filtering

- Motivation: Have to find a balance between false positive and false negative (it cannot be attained by changing the value of the threshold $\theta$).
- 3 filters
  - Each as the ability to silence attribute
  - If some are not silenced after passing the filters then you raise an alarm

1. Syntaxic Recognizer (much in the style of previous papers on the subject)
2. Exploiting External Sources
   - Trains a new sensor on newly acquired data from a different source
   - Lets you learn from other previously broken mappings
3. Learning from the web (Google is your friend!)
   - Is “185,000 USD” a cost?
   - Search for:
     3.1 “185,000 USD”
     3.2 “cost 185,000 USD”
   - If the ratio is high enough, then it’s valid (ie: silence the attribute)
   - This is the most semantic it gets
Results

<table>
<thead>
<tr>
<th>Domain</th>
<th>Lerman System</th>
<th>Sensor Ensemble (D)</th>
<th>Sensor Ensemble (ND)</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>P / R</td>
<td>F-I</td>
<td>P / R</td>
</tr>
<tr>
<td>Flights</td>
<td>0.81 / 1.00</td>
<td>0.85</td>
<td>0.93 / 0.98</td>
</tr>
<tr>
<td>Books</td>
<td>0.83 / 1.00</td>
<td>0.89</td>
<td>0.90 / 0.99</td>
</tr>
<tr>
<td>Researchers</td>
<td>0.77 / 0.99</td>
<td>0.84</td>
<td>0.90 / 0.99</td>
</tr>
<tr>
<td>Real Estate</td>
<td>0.45 / 0.90</td>
<td>0.63</td>
<td>0.80 / 0.82</td>
</tr>
<tr>
<td>Inventory</td>
<td>0.52 / 0.89</td>
<td>0.67</td>
<td>0.75 / 0.90</td>
</tr>
<tr>
<td>Courses</td>
<td>0.49 / 0.94</td>
<td>0.66</td>
<td>0.92 / 0.88</td>
</tr>
</tbody>
</table>

Conclusion

- They improve previous work by
  - Broader collection of evidences
  - The ensemble of sensors allow to use these evidences
  - Weighted combining of sensors
- They still have to improve
  - Unrecognized formats (not seen in training and not on the web)
  - Mixed same type attributes when they were switched
Final remarks

- Why not use a more powerful meta-learning algorithm (i.e., AdaBoost (Schapire '90s))? 
- How far can we push the stream analogy? 
- Why does (D) perform as well as (ND)? 

- Previous work do mostly syntaxic analysis 
- This paper does it a little better 
  - They have the same fundamental problems as other papers 
  - Very adhoc sensors 
  - Needs to train on every source independently 
  - Filtering is a good idea but it needs to be pushed further 
- Next step, try to understand semantically the query returns 
- AI techniques would become even more important