Source Modeling:
Learning Definitions of Online Sources for Information Integration

Mark Carman
Information Sciences Institute
University of Southern California

I will present work by Craig Knoblock, Kristina Lerman, Anon Plangprasongschok, and myself.
Abundance of Information Sources

Motivation
Approach
Labeling
Modeling
Related Work
Conclusions

- Orbitz
- Travel Deals
- Cheap Flights
- Travelocity
- Airfares
- Used Cars for Sale!
- Yaho Classifieds
- GoogleBase
- Hotel
- Airfare
- Tsunami
- Warnings!
- Exchange Rates
- Weather
- Recasts
- Realtime Stock Quote
- Package Deals
- Stock Quotes
- Warnings!
- Earthquake Data
- Cheap Flights
- New Cars for Sale!
- Last Minute Flights
- Classified Listings
- Weather Recasts
Mediators resolve Heterogeneity
Mediators Require Source Definitions

- New service => no definition!
- Can we model it automatically?

**Source Definitions:**
- Orbitz Flight Search
- United Airlines
- Qantas Specials

**Reformulated Query:**

```
SELECT MIN(price)
FROM flight
WHERE depart="LAX"
AND arrive="MXP"
```

**New Service:**
- Alitalia

**Reformulated Query:**

- `lowestFare("LAX","MXP")`
- `calcPrice("LAX","MXP","economy")`

**Generate Model of Service**
Modeling Sources: an Example

Step 1: classify input & output
semantic types, using:
- Metadata (labels)
- Data (content)

source1($zip, lat, long) :-
centroid(zip, lat, long).

source2($lat1, $long1, $lat2, $long2, dist) :-
greatCircleDist(lat1, long1, lat2, long2, dist).

source3($dist1, dist2) :-
convertKm2Mi(dist1, dist2).

source4( $startZip, $endZip, separation)
Modeling Sources: Step 2

Step 2:
model functionality of service by:
- generating plausible definitions

source1($zip, lat, long) :-
  centroid(zip, lat, long).

source2($lat1, $long1, $lat2, $long2, dist) :-
  greatCircleDist(lat1, long1, lat2, long2, dist).

source3($dist1, dist2) :-
  convertKm2Mi(dist1, dist2).

source4( $zip1, $zip2, dist) :-
  source1(zip1, lat1, long1),
  source1(zip2, lat2, long2),
  source2(lat1, long1, lat2, long2, dist2),
  source3(dist2, dist).

centroid(zip1, lat1, long1),
centroid(zip2, lat2, long2),
greatCircleDist(lat1, long1, lat2, long2, dist2),
convertKm2Mi(dist1, dist2).
Modeling Sources: Step 2

Step 2:
model functionality of service by:

- generating plausible definitions
- comparing the output they produce

```
source4($zip1, $zip2, dist) :-
    source1($zip1, lat1, long1),
    source1($zip2, lat2, long2),
    source2(lat1, long1, lat2, long2, dist2),
    source3(dist2, dist).
```

<table>
<thead>
<tr>
<th>..</th>
<th>..</th>
<th>..</th>
<th>..</th>
</tr>
</thead>
<tbody>
<tr>
<td>80210</td>
<td>90266</td>
<td>842.37</td>
<td>843.65</td>
</tr>
<tr>
<td>60601</td>
<td>15201</td>
<td>410.31</td>
<td>410.83</td>
</tr>
<tr>
<td>10005</td>
<td>35555</td>
<td>899.50</td>
<td>899.21</td>
</tr>
</tbody>
</table>

match
Summary - Modeling Sources

Step 1: Semantic Labeling
Classify input & output *semantic types*, using:
- Labels: metadata
- Content: output data

Step 2: Functional Modeling
Model the *functionality* of service by:
- Search: generating plausible definitions
- Scoring: compare the output they produce
Approach to Semantic Labeling

Leverage existing knowledge to label inputs & outputs:

**Semantic Types with Examples:**
- Zipcode: "90066", ...
- Latitude: "34.12", ...
- Temperature: "30°F", ...
- Humidity: "35%", ...

**Labeled Example WSDL Files:**
- Operation = "GetZipCodeCoordinates"
  - Input = "zip" <Zipcode>
  - Output1 = "LatDegrees" <Latitude>
  - Output2 = "LonDegrees" <Longitude>

**Metadata based classifier**

**Content-based classifier**

**New Service**

Motivation      Approach     Labeling Modeling      Related Work      Conclusions
Observation 1:
Similar types tend to be named similarly, and/or belong to operations with similar names

- Treat as (ungrammatical) text classification problem
- Approach taken by previous works

Observation 2:
The classifier must be a soft classifier

- Instance can belong to more than one class
- Rank classification results
**Independence Assumption**

Naïve Bayes classifier

- Input/output parameter represented by term vector $t$ (Hess & Kushmerick, 2004)
- Based on independence assumption
  Terms are independent given class label $D$:
  \[
  P(D|t) \leftarrow \Pi_i P(t_i|D)
  \]
- Assumption unrealistic as terms often co-occur:
  e.g., “Temp” and “Fahrenheit” in “getTempInFahrenheit”

Logistic regression

- Avoids independence assumption by estimating probabilities from data
  \[
  P(D|t) = \text{logreg}(wt)
  \]
Metadata-based Classification: Evaluation

Data collection
- Data extracted from 313 WSDL files

Data processing
- Labels extracted from operation, message, datatype and facet (predefined option)
- Labels tokenized into individual terms

10,000+ parameters extracted
- Each was assigned to one of 80 semantic types e.g. latitude, city, humidity.
- Other classes treated as “Unknown” class
Evaluation Results

- Both Naïve bayes and Logistic regression were tested using 10-fold cross validation

<table>
<thead>
<tr>
<th>Classifier</th>
<th>Top 1</th>
<th>Top 2</th>
<th>Top 3</th>
<th>Top 4</th>
</tr>
</thead>
<tbody>
<tr>
<td>Naïve Bayes</td>
<td>0.65</td>
<td>0.84</td>
<td>0.88</td>
<td>0.90</td>
</tr>
<tr>
<td>Logistic Regression</td>
<td>0.93</td>
<td>0.98</td>
<td>0.99</td>
<td>0.99</td>
</tr>
</tbody>
</table>
Content-based Classification

- Idea: Learn a model of the content of data and use it to recognize new examples

Use domain-independent language to represent structure of data

- Token-level
  - Specific tokens
  - General token types based on syntactic categories of token’s characters

- Hierarchy of types
  - allows for multi-level generalization
Patterns for Describing Data

- Pattern is a sequence of tokens and general token types:
  - Phone numbers

<table>
<thead>
<tr>
<th>Examples</th>
<th>Patterns</th>
</tr>
</thead>
<tbody>
<tr>
<td>(310) 448–8714</td>
<td>[(310) 448 – 4DIGIT]</td>
</tr>
<tr>
<td>(310) 448–8775</td>
<td>[(3DIGIT) 3DIGIT – 4DIGIT]</td>
</tr>
<tr>
<td>(212) 555–1212</td>
<td></td>
</tr>
</tbody>
</table>

- Algorithm learns patterns from examples
- Patterns for each semantic type in domain model
Patterns for Semantic Labeling

Use patterns to map new data to semantic types

- Score how well patterns associated with a semantic type describe a set of examples
- Heuristics include:
  - Number of matching patterns
  - How specific the matching patterns are
  - How many tokens of the example are left unmatched
- Output four top-scoring types
Semantic Labeling Evaluation

Information domains and semantic types

- Weather Services
  - Temperature, SkyConditions, WindSpeed, WindDir, Visibility
- Directory Services
  - Name, Phone, Address
- Electronics equipment purchasing
  - ModelName, Manufacturer, DisplaySize, ImageBrightness, …
- UsedCars
  - Model, Make, Year, BodyStyle, Engine, …
- Geospatial Services
  - Address, City, State, Zipcode, Latitude, Longitude
- Airline Flights
  - Airline, flight number, flight status, gate, date, time
Evaluation Results
Empirical Validation

- Automatically model the inputs and outputs used by Geospatial and Weather Web Services
  - Given the WSDL file of a new service
  - 8 services (13 operations)

- Results

<table>
<thead>
<tr>
<th>classifier</th>
<th>total</th>
<th>correct</th>
<th>accuracy</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>input parameters</strong></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>metadata-based</td>
<td>47</td>
<td>43</td>
<td>0.91</td>
</tr>
<tr>
<td><strong>output parameters</strong></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>metadata-based</td>
<td>213</td>
<td>145</td>
<td>0.68</td>
</tr>
<tr>
<td>content-based</td>
<td>213</td>
<td>107</td>
<td>0.50</td>
</tr>
<tr>
<td>combined</td>
<td>213</td>
<td>171</td>
<td>0.80</td>
</tr>
</tbody>
</table>
Functional Modeling

Model the *functionality* of service by:

- Searching through the space of plausible definitions
- Score definitions by comparing the output they produce with that of the source being modeled
Searching for Definitions

- Search space of *conjunctive queries*:
  \[
  \text{target}(X) :\text{source1}(X_1), \text{source2}(X_2), \ldots
  \]

1. Sample the new source

2. Best-first search through space of candidate definitions

Invoke \textit{target} with set of random inputs;
Add empty clause to \textit{queue};

while (\textit{queue} not empty)
  \[
  \begin{align*}
  v & := \text{best definition from } \textit{queue}; \\
  \forall (v') \in \text{Expand}(v) \\
  \text{if } (\text{Eval}(v') > \text{Eval}(v)) \\
  \text{insert } v' \text{ into } \textit{queue};
  \end{align*}
  \]
Invoking the Target

Generate Input Tuples: \(<zip1, dist1>\)

Invoke

\[ source5(\ \$zip1, \$dist1, \text{zip2}, \text{dist2}) \]

Invoke source with \textit{representative} values

- Randomly generate input tuples
  Use distribution if available
- If no output is produced:
  Try invoking other sources to generate input

<table>
<thead>
<tr>
<th>Input</th>
<th>Output</th>
</tr>
</thead>
<tbody>
<tr>
<td>(&lt;07307, 50.94&gt;)</td>
<td>{(&lt;07097, 0.26&gt;, &lt;07030, 0.83&gt;, &lt;07310, 1.09&gt;, \ldots}}</td>
</tr>
<tr>
<td>(&lt;60632, 10874.2&gt;)</td>
<td>{}</td>
</tr>
</tbody>
</table>

Motivation      Approach
Labeling      Modeling
Related Work      Conclusions
Top-down Generation of Candidates

Start with empty clause & specialize it by:

- Adding a predicate from set of sources
- Checking that each definition is executable & not redundant

New Source 5

```
source5(_,_,_,_).
source5(zip1,_,_,_) :- source4(zip1,zip1,_).
source5(zip1,_,zip2,dist2) :- source4(zip2,zip1,dist2).
source5(_,dist1,_,dist2)    :- <(dist2,dist1).
...```

source5( $zip1,$dist1,zip2,dist2)
Best-first Enumeration of Candidates

Evaluate the clauses produced and expand the best one found

source5(_,_,_,_).

source5(zip1,_,_,_) :- source4(zip1,zip1,_).
source5(zip1,_,zip2,dist2) :- source4(zip2,zip1,dist2).
source5(_,dist1,_,dist2) <(dist2,dist1).

source5(zip1,dist1,zip2,dist2) :- source4(zip2,zip1,dist2), source4(zip1,zip2,dist1).
source5(zip1,dist1,zip2,dist2) :- source4(zip2,zip1,dist2), <(dist2,dist1).

…
Limiting the Search

- Extremely Large Search space!
- Constrained by use of Semantic Types
- Limit search by:
  - Maximum Clause length
  - Maximum Predicate Repetition
  - Maximum Number of Existential Variables
  - Definition must be Executable
  - Maximum Variable Repetition within Literal

Standard techniques
Non-standard technique
Evaluating Candidates

- Compare output of each candidate with that of target.
- Average results across different input tuples.

<table>
<thead>
<tr>
<th>Input</th>
<th>Target Output</th>
<th>Clause Output</th>
</tr>
</thead>
<tbody>
<tr>
<td>&lt;$zip1, $dist1&gt;</td>
<td>{ }</td>
<td>{&lt;60629, 2.15&gt;, &lt;60682, 2.27&gt;, &lt;60623, 2.64&gt;, ...}</td>
</tr>
<tr>
<td>&lt;07307, 50.94&gt;</td>
<td>{&lt;07097, 0.26&gt;, &lt;07030, 0.83&gt;, &lt;07310, 1.09&gt;, ...}</td>
<td>{ }</td>
</tr>
<tr>
<td>&lt;28041, 240.46&gt;</td>
<td>{&lt;28072, 1.74&gt;, &lt;28146, 3.41&gt;, &lt;28138, 3.97&gt;, ...}</td>
<td>{&lt;28072, 1.74&gt;, &lt;28146, 3.41&gt;}</td>
</tr>
</tbody>
</table>

No Overlap
No Overlap
Overlap!
Evaluating Candidates II

- Candidates may return multiple tuples per input
  Need measure that compares sets of tuples!
- Sources may not be complete
  Want definition that best predicts the target’s output
- Use Jaccard similarity

\[
\text{forall (tuple in InputTuples)} \quad \begin{align*}
T_{\text{target}} &= \text{invoke}(\text{target, tuple)} \\
T_{\text{clause}} &= \text{execute}(\text{clause, tuple)} \\
\text{if not (} | T_{\text{target}}| = 0 \text{ and } | T_{\text{clause}}| = 0) \end{align*}
\]

\[
\text{fitness} = \frac{|T_{\text{target}} \cap T_{\text{clause}}|}{|T_{\text{target}} \cup T_{\text{clause}}|}
\]

return average(\text{fitness})

At least half of input tuples are non-empty invocations of target

Similarity metric is Jaccard similarity between the sets

Average results only when output is returned
Approximating Equality

Allow flexibility in values from different sources

- **Numeric Types like** *distance*
  
  \[10.6 \text{ km} \approx 10.54 \text{ km}\]
  
  Error Bounds (e.g. +/- 1%)

- **Nominal Types like** *company*
  
  Google Inc. \(\approx\) Google Incorporated
  
  String Distance Metrics (e.g. JaroWinkler Score \(> 0.9\))

- **Complex Types like** *date*
  
  Mon, 31. July 2006 \(\approx\) 7/31/06
  
  Hand-written equality checking procedures.
Experimental Setup

- 25 problems (target predicates) involving real services
- 35 known sources
- Time limit of 20 minutes imposed

Equality Approximations:
- 1% for distance, speed, temperature & price
- 0.002 degrees for latitude & longitude
- JaroWinkler > 0.85 for company, hotel & airport
- hand-written procedure for date.

Inductive search bias:
- Max clause length: 7
- Predicate repetition: 2
- Max variable level: 5
- Executable candidates
- No variable repetition
Actual Learned Examples

1. \textbf{GetDistanceBetweenZipCodes}($\text{zip0}, \text{zip1}, \text{dis2})$:
   \begin{align*}
   &\text{GetCentroid}(\text{zip0, lat1, lon2}), \text{GetCentroid}(\text{zip1, lat4, lon5}), \\
   &\text{GetDistance}(\text{lat1, lon2, lat4, lon5, dis10}), \text{ConvertKm2Mi}(\text{dis10, dis2}).
   \end{align*}

2. \textbf{USGSElevation}($\text{lat0}, \text{lon1}, \text{dis2}$):
   \begin{align*}
   &\text{ConvertFt2M}(\text{dis2, dis1}), \text{Altitude}(\text{lat0, lon1, dis1}).
   \end{align*}

3. \textbf{YahooWeather}($\text{zip0, cit1, sta2, lat4, lon5, day6, dat7, tem8, tem9, sky10}$) :-
   \begin{align*}
   &\text{WeatherForecast}(\text{cit1, sta2, lat4, lon5, day6, dat7, tem9, tem8, sky10}), \\
   &\text{GetCityState}(\text{zip0, cit1, sta2}).
   \end{align*}

4. \textbf{GetQuote}($\text{tic0, pri1, dat2, tim3, pri4, pri5, pri6, pri7, cou8, pri10, pri13, com15}$) :-
   \begin{align*}
   &\text{YahooFinance}(\text{tic0, pri1, dat2, tim3, pri4, pri5, pri6, pri7, cou8}), \\
   &\text{GetCompanyName}(\text{tic0, com15}), \text{Add}(\text{pri5, pri13, pri10}), \text{Add}(\text{pri4, pri10, pri1}).
   \end{align*}

5. \textbf{YahooAutos}($\text{zip0, mak1, dat2, yea3, mod4, pri7, pri7}$) :-
   \begin{align*}
   &\text{GoogleBaseCars}(\text{zip0, mak1, mod4, pri7, yea3}), \\
   &\text{ConvertTime}(\text{dat2, dat10}), \text{GetCurrentTime}(\text{dat10}).
   \end{align*}

Distinguished forecast from current conditions

current price = yesterday’s close + change
Experimental Results

Results for different domains:

<table>
<thead>
<tr>
<th>Problem Domain</th>
<th># of Problems</th>
<th>Avg. # of Candidates</th>
<th>Avg. Time (s)</th>
<th>Avg. Precision</th>
<th>Avg. Recall</th>
</tr>
</thead>
<tbody>
<tr>
<td>Geospatial</td>
<td>9</td>
<td>136</td>
<td>303</td>
<td>100%</td>
<td>84%</td>
</tr>
<tr>
<td>Financial</td>
<td>2</td>
<td>1606</td>
<td>335</td>
<td>56%</td>
<td>63%</td>
</tr>
<tr>
<td>Weather</td>
<td>8</td>
<td>368</td>
<td>693</td>
<td>91%</td>
<td>62%</td>
</tr>
<tr>
<td>Hotels</td>
<td>4</td>
<td>43</td>
<td>374</td>
<td>90%</td>
<td>60%</td>
</tr>
<tr>
<td>Cars</td>
<td>2</td>
<td>68</td>
<td>940</td>
<td>50%</td>
<td>50%</td>
</tr>
</tbody>
</table>
Related Work: Semantic Labeling

Metadata-based classification of data types used by Web services and HTML forms
(Hess & Kushmerick, 2003)
- Naïve Bayes classifier
- No invocation of services

Woogle: Metadata-based clustering of data and operations used by Web services
(Dong et al, 2004)
- Groups similar types together: Zipcode, City, State
- Cannot invoke services with this information
Related Work: Functional Modeling

Category Translation Problem: Learn functions describing operations on internet
(Perkowitz & Etzioni 1995)
- System learns less complicated definitions
  Single input attribute, single output tuples.

iMAP: Complex (many-to-1) schema matcher
(Dhamanka et. al. 2004)
- Discovers many-to-1 not many-to-many mappings
- Built on top of multiple type-specific search algorithms
  (our approach is more general)
- Doesn’t deal with problem of invoking sources
Conclusions

- Assumption: overlap between new & known sources
- Nonetheless, the technique is widely applicable:
  - Redundancy
  - Scope or Completeness
  - Binding Constraints
  - Composed Functionality
  - Access Time
Conclusions

- Integrated approach to learning:
  - *How to invoke a web service*
  - *The semantic types of the output*
  - *A definition of what the service does*

- Provides an approach to generate source descriptions for the Semantic Web
  - Little motivation for providers to annotate services
  - Instead we generate metadata automatically

- Also provides an approach to automatically discover new sources of data