Learning Definitions of Online Sources for Information Integration

Craig Knoblock
University of Southern California

This is joint work with Mark Carman,
Kristina Lerman, and Anon Plangprasongsok
Learning Models of the Sources: Source Modeling vs. Schema Matching

- **Schema Matching**
  - Align schemas between data sources
  - Assumes static sources and complete access to data

- **Source modeling**
  - Incrementally build models from partial data (e.g., web services, html forms, programs)
  - Model not just the fields but the source types and even the function of a source
  - Support richer source models (a la Semantic Web)
Mediators Require Source Definitions

- New service => no source definition!
- Can we discover a definition automatically?

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

```
SELECT MIN(price)
FROM flight
WHERE depart="LAX"
AND arrive="MXP"
```
Inducing Source Definitions by Example

- Step 1: classify input & output semantic types

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)
Inducing Source Definitions - Step 2

- **Step 1**: classify input & output semantic types
- **Step 2**: generate plausible definitions

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

source2($lat1, $long1, $lat2, $long2, dist) :-
  greatCircleDist(lat1, long1, lat2, long2, dist2),
  source3(dist2, 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).
```

```python
source4($zip1, $zip2, dist) :-
  centroid(zip1, lat1, long1),
  centroid(zip2, lat2, long2),
  greatCircleDist(lat1, long1, lat2, long2, dist2),
  convertKm2Mi(dist1, dist2).
```
Inducing Source Definitions – Step 3

- **Step 1:** classify input & output semantic types
- **Step 2:** generate plausible definitions
- **Step 3:** invoke service & compare output

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

```
match
```

```
source4($zip1, $zip2, dist):-
centroid(zip1, lat1, long1),
centroid(zip2, lat2, long2),
greatCircleDist(lat1, long1, lat2, long2, dist2),
convertKm2Mi(dist1, dist2).
```

<table>
<thead>
<tr>
<th>$zip1</th>
<th>$zip2</th>
<th>dist (actual)</th>
<th>dist (predicted)</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>

2 August 2006

University of Southern California
Our Approach to Semantic Labeling

Leverage existing knowledge to learn semantics of data used by Web services

Domain model
... Place
   Street
   Zipcode
   Latitude
   Longitude
... Distance
... Weather
   Temperature
   Humidity
...

80+ types with examples

Motivation      Approach     Classify      Search      Scoring      Related Work      Conclusions

Metadata based classifier
invoke
src
output data
Content-based classifier
model

wsdl

-<complexType=ZipCodeCoordinates>
  <element=LatDegrees type="s:float"/>
  <element=LonDegrees type="s:float"/>
-<message=GetZipCodeCoordinatesSoapIn>
  <part=zip type="s:string"/>

2 August 2006
University of Southern California
Metadata-based Classification

- **Observation 1**
  Similar data types tend to be named with similar words, and/or belong to operations that have similar name
  - 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**
  - Used to classify parameters used by Web services (Hess & Kushmerick, 2004)
    - Each input/output parameter represented by a term vector $t$
  - Based on independence assumption
    - Terms are independent from each other given the class label $D$ (semantic type)
    - $P(D|t) \equiv \Pi_i P(t_i|D)$
  - Independence assumption unrealistic for Web services
    - e.g., “TempFahrenheit”: “Temp” and “Fahrenheit” often co-occur in the Temperature semantic type

- **Logistic regression avoids the independence assumption**
  - Estimates probabilities from the data
    - $P(D|t) = \text{logreg}(wt)$
Metadata-based Classification Evaluation

- Data collection
  - Data extracted from 313 WSDL files from Web service portals (bindingpoint and webservicex)

- Data processing
  - Names were extracted from operation, message, datatype and facet (predefined option)
  - Names tokenized into individual terms
  - 10,000+ data types extracted
    - Each one assigned to one of 80 classes in geospatial and weather domains (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

Developed a domain-independent language to represent the 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 types
  - Phone numbers
    - Examples
      - 310 448–8714
      - 310 448–8775
      - 212 555–1212
    - Patterns
      - [(310) 448 – 4DIGIT]
      - [(3DIGIT) 3DIGIT – 4DIGIT]
- Algorithm to learn patterns from examples
- Patterns for all semantic types in the domain model
Patterns for Semantic Labeling

- Use learned patterns to map new data to types in the domain model
  - 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
Evaluations Results
Using all semantic types in classification

Restricting semantic types to domain of the source
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>
Searching for Definitions

- Search space of *conjunctive queries*:
  \[ \text{target}(X) :- \text{source1}(X_1), \text{source2}(X_2), \ldots \]
- For scalability don’t allow negation or union
- Perform Top-Down Best-First Search

1. First sample the New Source

2. Then perform best-first search through space of candidate definitions

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

\[
\text{while (queue not empty)} \quad \\
\quad v := \text{best definition from queue}; \\
\quad \text{forall } (v’ \text{ in } \text{Expand}(v)) \quad \\
\quad \quad \text{if } (\text{Eval}(v’) > \text{Eval}(v)) \quad \\
\quad \quad \quad \text{insert } v’ \text{ into queue;} \\
\]

Expressive Language
Sufficient for modeling most online sources

Motivation      Approach     Classify      Search      Scoring      Related Work      Conclusions

2 August 2006 University of Southern California
Invoking the Target

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

Invoke

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

Invoke source with representative values

- Try randomly generating input tuples:
  - Combine examples of each type
  - Use distribution if available

<table>
<thead>
<tr>
<th>Input</th>
<th>Output</th>
</tr>
</thead>
<tbody>
<tr>
<td>(&lt;\text{zip1, dist1}&gt;)</td>
<td>(&lt;\text{zip2, dist2}&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>
</tr>
<tr>
<td>(&lt;60632, 10874.2&gt;)</td>
<td>({})</td>
</tr>
</tbody>
</table>

Randomly Combined Example Values

Non-empty Result

Empty Result

2 August 2006
University of Southern California
Invoking the Target

Generate Input Tuples: 
<zip1, dist1>

Invoke

source5( $zip1, $dist1, zip2, dist2)

Invoke source with *representative* values

- Try randomly generating input tuples:
  - Combine examples of each type
  - Use distribution if available
- If *only empty invocations* result
  - Try invoking other sources to generate input
- Continue until sufficient non-empty invocations result
Top-down Generation of Candidates

Start with empty clause & generate specialisations by

- Adding one predicate at a time from set of sources
- Checking that each definition is:
  - Not logically redundant
  - Executable (binding constraints satisfied)

source5(_,_,_,_).

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

source5( $zip1,$dist1,zip2,dist2)

New Source 5
Best-first Enumeration of Candidates

- Evaluate each clause produced
- Then expand best one found so far
- Expand high-arity predicates incrementally

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

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

New Source 5
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 ILP techniques
Non-standard technique
Evaluating Candidates

- Compare output of clause with that of target.
- Average the results across different input tuples.
Evaluating Candidates II

Candidates may return multiple tuples per input
- Need measure that compares sets of tuples!

<table>
<thead>
<tr>
<th>Input</th>
<th>Target Output</th>
<th>Clause Output</th>
</tr>
</thead>
<tbody>
<tr>
<td>&lt;60632, 874.2&gt;</td>
<td>{}</td>
<td>{}</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 III

PROBLEM: All sources assumed incomplete
- Even *optimal definition* may only produce overlap
- Want definition that *best predicts* the target’s output
- Use Jaccard similarity to score candidates

\[
\text{return average}\left( \text{fitness} \right) \quad \text{forall (tuple in InputTuples)}
\]

\[
T_{\text{target}} = \text{invoke}(\text{target, tuple})
\]

\[
T_{\text{clause}} = \text{execute}(\text{clause, tuple})
\]

if not (\mid T_{\text{target}}\mid = 0 \text{ and } \mid T_{\text{clause}}\mid = 0)

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

return average(\text{fitness})
Missing Output Attributes

Some candidates produce less output attributes:
- Makes comparing them difficult

1. `source5(zip1,_,_,_) :- source4(zip1,zip1,_)`.
2. `source5(zip1,_,zip2,dist2) :- source4(zip2,zip1,dist2)`.

Penalize candidate by number of “negative examples”

`source5($zipcode, $distance, $zipcode, $distance)`

First candidate doesn’t produce either outputs, thus:
- Penalty = `|{zipcode}| x |{distance}|`
- For numeric types use accuracy to approximate cardinality
Different Input Attributes

- Some clauses take different inputs from target:
  
  ```prolog
  source5($zip1,$dist1,zip2,_) :- source4($zip1,$zip2,dist1).
  ```

  **Target Input**

  **Clause Input**

- `zip2` is an input parameter for clause but not target
- Should invoke operation with *every possible zip code!*

- Problem: algorithm should return & not get banned!
- Solution: sample to estimate score for clause:
  - record the scaling factor $= \frac{|\{\text{zipcode}\}|}{\# \text{invocations}}$
  - bias search: choose at least half of tuples to be positive

> 40,000 zip codes in US

2 August 2006
Approximating Equality

Allow flexibility in values from different sources

- **Numeric Types like** *distance*
  
  10.6 km $\approx$ 10.54 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.
Experiments – Setup

Problems:
- 25 target predicates
- *same* domain model
  (70 Semantic Types and
  37 Predicates)
- 35 known sources

System Settings:
- Each target source invoked at least 20 times
- Time limit of 20 minutes imposed

Inductive search bias:
- Maximum clause length 7
- Predicate repetition limit 2
- Maximum variable level 5
- Candidate must be executable
- Only 1 variable occurrence per literal

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*.
Actual Learned Examples

1. `GetDistanceBetweenZipCodes($zip0, $zip1, dis2):-
   GetCentroid(zip0, lat1, lon2), GetCentroid(zip1, lat4, lon5),
   GetDistance(lat1, lon2, lat4, lon5, dis10), ConvertKm2Mi(dis10, dis2).`

2. `USGSElevation($lat0, $lon1, dis2):-
   ConvertFt2M(dis2, dis1), Altitude(lat0, lon1, dis1).`

3. `YahooWeather($zip0, cit1, sta2, , lat4, lon5, day6, dat7, tem8, tem9, sky10) :-
   WeatherForecast(cit1,sta2,,lat4,lon5,,day6,dat7,tem9,tem8,,sky10,,),
   GetCityState(zip0, cit1, sta2).`

4. `GetQuote($tic0,pri1,dat2,tim3,pri4,pri5,pri6,pri7,cou8,,pri10,,,pri13,,com15) :-
   YahooFinance(tic0, pri1, dat2, tim3, pri4, pri5, pri6, pri7, cou8),
   GetCompanyName(tic0,com15,,),Add(pri5,pri13,pri10),Add(pri4,pri10,pri1).`

5. `YahooAutos($zip0, $mak1, dat2, yea3, mod4, , , pri7, ) :-
   GoogleBaseCars(zip0, mak1, , mod4, pri7, , , yea3),
   ConvertTime(dat2, , dat10, , ), GetCurrentTime( , , dat10, ).`

Distinguished forecast from current conditions

Motivation      Approach     Classify      Search      Scoring      Related Work      Conclusions
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 (sec)</th>
<th>Attributes Learnt</th>
</tr>
</thead>
<tbody>
<tr>
<td>geospatial</td>
<td>9</td>
<td>136</td>
<td>303</td>
<td>84%</td>
</tr>
<tr>
<td>financial</td>
<td>2</td>
<td>1606</td>
<td>335</td>
<td>59%</td>
</tr>
<tr>
<td>weather</td>
<td>7</td>
<td>368</td>
<td>693</td>
<td>69%</td>
</tr>
<tr>
<td>hotels</td>
<td>4</td>
<td>43</td>
<td>374</td>
<td>60%</td>
</tr>
<tr>
<td>cars</td>
<td>2</td>
<td>68</td>
<td>940</td>
<td>50%</td>
</tr>
</tbody>
</table>
Related Work

ILA & Category Translation (Perkowitz & Etzioni 1995)
Learn functions describing operations on internet

- Our system learns *more complicated* definitions
  - Multiple attributes, Multiple output tuples, etc.

iMAP (Dhamanka et. al. 2004)
Discovers complex (many-to-1) mappings between DB schemas

- Our system learns *many-to-many* mappings
- Our approach is more general (single search algorithm)
- We deal with problem of invoking sources
Related Work

- 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
Discussion

- 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