Learning Definitions of Online Sources for Information Integration

Craig Knoblock
University of Southern California

This is joint work with Mark Carman, Kristina Lerman, and Atena Pasupatapopthong.

Abundance of Information Sources

Motivation
Approach
Classify
Search
Scoring
Related Work
Conclusions

Orbitz
Travel Deals
Cheap Flights
Travelocity
Airfares
Used Cars for Sale!

Yahoo Classifieds
GoogleBase
Hotels
Tsunami Warnings!

Exchange Rates
Weather Forecasts
Realtime Stock Quote
Weather Conditions
Package Deals
Last Minute Flights

New Cars for Sale!
Classified Listings
Hotel Deals
Earthquake Data
Currency Rates
Stock Quotes
Flight Status

Mediators resolve Heterogeneity

Mediators Require Source Definitions

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

Inducing Source Definitions by Example

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

Step 1: classify input & output semantic types

zip code
new source
distance

source1(zip, lat, long) :- centroid(zip, lat, long).
source2($zip1, $zip2, dist) :-
  greatCircleDist($zip1, $zip2, dist).
source3($dist1, dist2) :-
  convertKm2Mi($dist1, $dist2).

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

New service => no source definition!
Can we discover a definition automatically?
**Inducing Source Definitions - Step 2**

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

**Our Approach to Semantic Labeling**

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

![Diagram of semantic labeling process](image)

**Motivation**

- Domain model
- Source 1
- Source 2
- Source 3

**Approach**

- Naive Bayes classifier
- Used to classify parameters used by Web services (Hess & Kushmerick, 2004)
- Each input/output parameter represented by a term vector
- Based on independence assumption
- Terms and independence from data others given class label of semantic type

**Classify**

- Independence assumption unrealistic for Web services
- E.g., "temperature" and "humidity" often co-occur in the data

**Search**

- Logistic regression avoids the independence assumption
- Estimates probabilities from the data

**Scoring**

- Classify & output semantic types

**Rank classification results**

- Instance can belong to more than one class
- Data extracted from 313 WSDL files from Web service portals (binding point and webservices)

**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

**Evaluation**

- Data collection
- Data extracted from 313 WSDL files from Web service portals (binding point and webservices)

- 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>0.85</td>
<td>0.81</td>
<td>0.88</td>
<td>0.90</td>
</tr>
<tr>
<td>Logistic Regression</td>
<td>0.93</td>
<td>0.91</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: 119-448-1212, 310-448-8714, 212-555-2121
  - Patterns: [119] [448] [1212], [310] [448] [8714], [212] [555] [2121]
- 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

![Graph showing evaluation results]
2. Then perform best-first search until sufficient non-empty invocations result. If only empty invocations result, try randomly generating input tuples:

```
Input Tuples:
<zip1, dist1>
```

For scalability, don’t allow negation or union. 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>Input parameters</th>
<th>Output accuracy</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

### Invoking the Target
- Try invoking other sources to generate input.
- Use distribution if available.

### Evaluations Results 2

Using all semantic types in classification. Restricting semantic types to domain of the source.

### Searching for Definitions
- Search space of conjunctive queries.
- target(X1): source1(X1), source2(X1), ...
- For scalability, don’t allow negation or union.
- Perform Top-Down Best-First Search:
  - Start with empty clause & generate specialisations by adding one predicate at a time from set of sources.
  - If only empty invocations result, try invoking other sources to generate input.
  - Continue until sufficient non-empty invocations result.

### Expressive Language
- Sufficient for modeling most online sources.

### 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:
    - Executable (binding constraints satisfied).

### Invoking the Target
- New Source

### Top-down Generation of Candidates
- New Source
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).
```

```
source3(zip1,zip2,dist2) :- source4(zip1,zip2,dist2).
source6(zip1,zip2,dist2) :- source5(zip1,zip2,dist2).
```

Evaluating Candidates

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

```
Input Tuples:
- <07310, 1.09>, ...
- <28041, 240.46>, ...
- <60632, 874.2>, ...
```

```
Target Output:
- <zip2, dist2>
```

```
Clause Output:
- <zip2, dist2>
- ...
```

Evaluating Candidates II

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

```
Overlap!
No
```

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

```
forall (tuple in input tuples)
    if not (T_target=0 and T_clause=0) then
        F target/T target = 1 - |T_target| / |T_target| + |T_clause| - |T_target| 
        F clause/T clause = 1 - |T_clause| / |T_clause| + |T_target| - |T_clause|
```

```
Jaccard similarity between the sets
```

Limiting the Search

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

```
source5(zip1,_,zip2,dist2).
source4(zip1,_,_,_).
```

Missing Output Attributes

- Some candidates produce less output attributes
- Makes comparing them difficult
- 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
**Motivation | Approach | Classify | Search | Scoring | Related Work | Conclusions**

**Different Input Attributes**
- Some clauses take different inputs from target:
  - `source5(zip1, dat1, zip2) :- source4(zip1, zip2, dis1)`
  - `Target Input`  
  - `Clause Input`
- `zip2` is an input parameter for clause but not target.

**Problem:** algorithm should return & not get banned!  
**Solution:** sample to estimate score for clause:
- record the scaling factor = |{record the scaling factor = |{zipcode}|/ #invocations}|

**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>1600</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>44</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>

**Approximating Equality**
- Allow flexibility in values from different sources
- **Numeric Types like distance**
  - `10.5 km = 10.54 km`
- **Error Bounds** (e.g., `+/- 1%`)
- **Complex Types like date**
  - `Mon, 31 July 2006 = 7/31/06`

**Experimental Results**

**Experiments - Setup**
- 25 target predicates  
- same domain model  
- 35 known sources
- Each target source invoked at least 20 times

**Equality Approximations:**
- 1% for distance, speed, temperature & price
- 0.002 degrees for latitude & longitude
- JaroWinkler > 0.85 for company, hotel & airport

**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** (Ohmanna et al. 2004)
- Discovers complex (many-to-1) mappings between DB schemata
  - 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)
- Naive 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