Automatic Wrapper Generation

Kristina Lerman
University of Southern California

Manual Wrapper Generation

- Manual wrapper generation requires user to:
  - Specify the schema of the information source
    - Single tuple
    - Nested type
    - List of tuples or nested types
  - Label data on several example HTML pages
  - Tedious, especially for lists
- Goal of Automatic Wrapper Generation is to eliminate user intervention

Overview

- Methods for automatic wrapper creation and data extraction
  - Grammar Induction approach:
    - Towards Automatic Data Extraction from Large Web Sites
  - Website structure-based approach:
    - AutoFeed: An Unsupervised Learning System for Generating Webfeeds
    - Using the Structure of Web Sites for Automatic Segmentation of Tables
  - DOM-based:
    - Simile/PiggyBank

Grammar Induction Approach

- Pages automatically generated by scripts that encode results of db query into HTML
- Given a set of pages generated by the same script
- Learn the grammar of the pages
- Use the grammar to parse the pages
- Data extraction step

Limitations

- Learnable grammars
  - Variety of schema structure: tuples (with optional attributes) and lists of (nested) tuples
  - Does not efficiently handle disjunctions – pages with alternate presentations of the same attribute
  - Context-free Grammars (Hong&Clark paper)
  - Limited learning ability
- User needs to provide a set of pages of the same type

Website Structure-based Approach

- Websites attempt to simplify user navigation and interaction with data by organizing how data is presented across the site
  - Hierarchical organization
  - List of results
  - Detail pages...
- Machine learning methods take advantage of structure to extract data
Limitations

- Performance depends on the choice of the learning algorithm
  - Noise can affect performance (Lerman et al paper)
  - Can combine different learning algorithms to improve performance (Autofeed)

DOM-based Approach

- Use Document Object Model tree of HTML pages to extract data
- Works well, but on fairly simple and well-structured pages

Towards Automatic Data Extraction from Large Web Sites by Crescenzi, Mecca, & Merialdo

RoadRunner Overview

- Automatically generates a wrapper from large structured web pages
- Supports nested structures and lists
- Efficient approach to large, complex pages with regular structure

Example Pages

- Extracts the fields and hierarchical structure
- Depends on well-structured HTML
- Only extracts at the entire field level

Extracted Result
Approach

- Given a set of example pages
- Generates a Union-free Regular Expression (UFRE)
  - RE without any disjunctions
  - List of tuples (possibly nested): (a, b, c)+
  - Optional attributes: (a)?
  - Strong assumption that usually holds
- Find the least upper bounds on the RE lattice to generate a wrapper in linear time
- Reduces to finding the least upper bound on two UFREs

Matching/Mismatches

- Given a set of pages of the same type
- Take the first page to be the wrapper (UFRE)
- Match each successive sample page against the wrapper
- Mismatches result in generalizations of the regular expression
- Types of mismatches:
  - String mismatches
  - Tag mismatches

Example Matching

String Mismatches: Discovering Fields

- String mismatches are used to discover fields of the document
- Wrapper is generalized by replacing "John Smith" with #PCDATA
- \(<HTML> Books of: \langle B \rangle John Smith \rangle \rightarrow \langle HTML \rangle Books of: \langle B \rangle \#PCDATA

Example Matching

Tag Mismatches: Discovering Optionals

- First check to see if mismatch is caused by an iterator (described next)
- If not, could be an optional field in wrapper or sample
- Cross search used to determine possible optionals
- Image field determined to be optional: `( <img src=.../>)?`
Example Matching

Tag Mismatches: Discovering Iterators

- Assume mismatch is caused by repeated elements in a list
  - End of the list corresponds to last matching token: 
    - Beginning of list corresponds to one of the mismatched tokens: </UL>
  - These create possible "squares"
- Match possible squares against earlier squares
- Generalize the wrapper by finding all contiguous repeated occurrences:
  - $(<LI><I>Title:</I>\#PCDATA</LI> \+)$

Internal Mismatches

- Generate internal mismatch while trying to match square against earlier squares on the same page
- Solving internal mismatches yield further refinements in the wrapper
- List of book editions
- $<I>Special!</I>$

Recursive Example

Discussion

- Assumptions:
  - Pages are well-structured
  - Want to extract at the level of entire fields
  - Structure can be modeled without disjunctions
  - Search space for explaining mismatches is huge
  - Uses a number of heuristics to prune space
  - Limited backtracking
  - Limit on number of choices to explore
  - Patterns cannot be delimited by optionals
  - Will result in pruning possible wrappers
AutoFeed: An Unsupervised Learning System for Generating Webfeeds by Gazen and Minton

Relational Model of a Web Site
- Sites are well structured to improve user experience
- Generation: Given relational data, scripts generate web site, e.g., weather site
- Extraction is opposite task: Given web site, find underlying relational data

Overview of Approach
- Sites aim to be easy to understand and navigate
  - Many types of structure
    - Graph structure of site’s links
    - URL naming scheme
    - Content of pages
    - HTML structure within page types,…
- Experts focus on individual structures and output discoveries as hints
  - Experts are heterogeneous
  - Probabilistically combine experts (don’t have to be correct all the time)

Hints
- Hints describe local structural similarities within pages or within data
- Hints help find relational structure of the site
  - Used to cluster pages and data

Overview

Page-level Experts
- URL patterns give clues about site structure
  - Similar pages have similar URLs, e.g.:
    - http://www.bookpool.com/sml/0321349806
    - http://www.bookpool.com/sml/0131118269
    - http://www.bookpool.com/ss/L?pu=MN
- Page templates
  - Similar pages contain common sequences of substrings
Data-level Experts

- Page layout gives clues about relational structure
  - Similar items aligned vertically or horizontally, e.g.
  - Coincidental alignment results in bad hints
- List structure
  - List rows are represented as repeating HTML structures

Page and Data Similarity

- Surface structure is often not helpful
  - E.g., page with an empty list and one with a long list will be found not similar
- Instead, use local structural similarities
  - Experts output hints
  - Structure represented as hints
  - Clusters evaluated probabilistically using hints
  - Probabilistic representation gives flexible framework for combining possibly conflicting hints

Probabilistic Evaluation

- Find clustering that maximizes probability of observing hints:
  \[ P(\text{clustering}|\text{hints}) = P(\text{hints}|\text{clustering}) \times P(\text{clustering}) / P(\text{hints}) \]
- Generative process for \( P(\text{hints}|\text{clustering}) \)
  \[
  P(i) = \prod_{m=1}^{\text{tokens}} \begin{cases} \frac{1}{2} & \text{if } m \text{ is a matched pair of tokens} \\ \frac{1}{2} \times \frac{1}{\text{tokens of pair matched}} & \text{if } m \text{ is an unmatched token} \end{cases}
  \]
- Page Hints
  \[
  \begin{align*}
  \text{ProbPage} &= \frac{1}{\text{tokens of pair matched}} \\
  \text{ProbPage} &= \frac{1}{\text{tokens of pair matched}} \\
  \text{ProbPage} &= \frac{1}{\text{tokens of pair matched}}
  \end{align*}
  \]

Clustering Algorithm

- Leader-follower algorithm
  - Process items one at a time
    - If distance to nearest cluster < threshold, add item to it
    - Else create new cluster
  - "distance" between page and page-cluster is determined by change in \( P(\text{clustering}|\text{hints}) \)
  - Prevents one large cluster, b/c smaller probabilities assigned to hints in larger clusters

Clustering

- Cluster pages and data
- Page-clusters are parents of data-clusters
- For example:

Clusters to Tables

- Base tables are built from clusters that contain a single tuple per page.
  - Each cluster becomes a column.
  - Each row represents a page.
- Lists constructed from clusters not used for base table
Evaluation

- AutoFeed evaluated against manually built wrappers
  - But can only evaluate part of output
- For each target column in the wrapper
  - Find matching AutoFeed column
  - Count relevant retrieved values
  - Calculate precision and recall

<table>
<thead>
<tr>
<th>Target Column from Supervised Wrapper</th>
<th>Column Extracted by AutoFeed</th>
</tr>
</thead>
<tbody>
<tr>
<td>Los Angeles</td>
<td>Mostly Cloudy</td>
</tr>
<tr>
<td>San Francisco</td>
<td>Hi</td>
</tr>
<tr>
<td>San Diego</td>
<td>Retrieved &amp; Relevant = 3</td>
</tr>
<tr>
<td>Pittsburgh</td>
<td>Retrieved = 4</td>
</tr>
<tr>
<td>Philadelphia</td>
<td>Relevant = 5</td>
</tr>
<tr>
<td>Retrieved &amp; Relevant = 3</td>
<td>Retrieved = 4</td>
</tr>
<tr>
<td>Relevant = 5</td>
<td>Precision = 3/4</td>
</tr>
<tr>
<td>Recall = 3/5</td>
<td></td>
</tr>
</tbody>
</table>

Experiments and Results

- **Domains**
  - Extract product name, manufacturer, price, etc. from online retail sites (Buy.com, CompUSA, etc.)
  - Extract titles, URLs from journals (DMTCS, JAIR, etc.)
  - Extract job id, position, location from job listings (50 Forbes 500 companies)
- **Good results**
  - Data extracted correctly from many of the sites
- **Some problems**
  - Extracting larger fields, e.g. “Price: $19.95”
  - Over-general & over-specific

<table>
<thead>
<tr>
<th>Field</th>
<th>RR</th>
<th>Rel</th>
<th>Prec</th>
<th>Retr</th>
</tr>
</thead>
<tbody>
<tr>
<td>_door</td>
<td>95</td>
<td>95</td>
<td>100</td>
<td>95</td>
</tr>
<tr>
<td>_job</td>
<td>85</td>
<td>85</td>
<td>100</td>
<td>85</td>
</tr>
<tr>
<td>_model</td>
<td>90</td>
<td>90</td>
<td>90</td>
<td>80</td>
</tr>
<tr>
<td>_name</td>
<td>95</td>
<td>95</td>
<td>100</td>
<td>95</td>
</tr>
<tr>
<td>_price</td>
<td>80</td>
<td>80</td>
<td>100</td>
<td>80</td>
</tr>
<tr>
<td>_title</td>
<td>75</td>
<td>75</td>
<td>100</td>
<td>75</td>
</tr>
<tr>
<td>url</td>
<td>100</td>
<td>100</td>
<td>100</td>
<td>100</td>
</tr>
</tbody>
</table>

AutoFeed Conclusions

- **Promising approach**:
  - Multiple experts for multiple structures
  - Common language for collecting and combining evidence
- **Principle applicable to beyond web extraction**
  - Interpreting complex environments
- **Need to improve prototype system**:
  - More experts
  - Better ways to combine evidence
  - Confidence scores on hints
  - Cannot-link hints

Using the Structure of Web Sites for Automatic Segmentation of Tables by Lerman et al.

Exploiting Structure of Web Sites

- Most Web sites that allow user to access databases present results in dynamically generated pages
- Web tables
- Web sites are very structured in terms of
  - Organization of the site
  - Layout of pages
  - Content of data
- Exploit this structure for automatic information extraction
Structure of Pages and Data

- Pages are generated from a template
- Data in the same "column" is of the same type
  - E.g., each listing starts with NAME, followed by ADDRESS, CITY, STATE, etc.

Underlying Structure is not Always Clear

- Variability of real-world data may obscure the underlying structure
- Missing columns
  - "List Price" and "You save"
- Formatting
- Content

Problem Overview

Automatically, efficiently extract records from Web tables

Given a set of list and detail pages...
- Segment list data using information from detail pages
- Logic based approach
  - Based on Constraint Satisfaction Problems (CSP)
  - Encode relations between data on list and detail pages as logical constraints and solve them
- Probabilistic inference approach
  - Learns a model from data
  - Record segmentation is an assignment that maximizes the likelihood of data given the model

Identify Table and Extract Data

Web sources generate list pages from a template and fill them with query results
- Deduce page template
  - Given two or more example pages, derive the template used to generate them
- Table data and formatting tags are not part of the template
- Heuristic: largest slot contains the table
- Extract table data
  - Extract contiguous sequences of tokens from the largest page slot

Record Segmentation Basics (1)

- List and detail pages present two views of the same record
- Some overlapping fields
- Each detail page is a distinct record
- Assumption: Web tables are laid out horizontally
  - Each record is in a separate row
  - Order in which extracts appear in the text stream of list page is the same order they appear in the table

Record Segmentation Basics (2)

For each extract $E_i$, record all detail pages on which it appears

- John Smith
  - Findlay, OH 45840
    - r1, /2
    - (419) 423-1212
  - (740) 335-5555
  - (740) 335-5555
- George W. Smith
  - Findlay, OH 45840
    - r3
    - (419) 423-1212
  - (740) 335-5555
Observations of extracts on detail pages add valuable information for record segmentation.

Second record can be

- \( E_{i} \) belongs to exactly one record \( r_{j} \)
- Only contiguous blocks of extracts can be assigned to the same record

Uniqueness constraint

Consecutiveness constraint

Position constraint

No two extracts assigned to same record can appear in the same position on the detail page

Constraints are expressed mathematically and solved using integer optimization
Probabilistic Model for Record Extraction: Variables

- **Observed variables**
  - \( T = \{ T_1, \ldots, T_n \} \) token types of extract \( E_i \)
  - \( D = \{ D_1, \ldots, D_n \} \) detail pages on which \( E_i \) was observed

- **Unobserved variables**
  - \( R = \{ R_1, \ldots, R_n \} \) record id
  - \( C = \{ C_1, \ldots, C_n \} \) column label
  - \( S = \{ S_1, \ldots, S_n \} \) \( S_i = \text{true} \) if \( E_i \) is the start of a new record; \( \text{false} \) otherwise

- **Dependencies**
  - Given by arrows, eg, \( P(C_i | C_{i-1}) \)

- **Segmentation**
  - Find values for \( R \) and \( C \) given \( T \) and \( D \) variables:
    - \( \text{argmax} \ P(R, C | T, D) \)

Learning the Model

- **Constrain the problem further**
  - **Bootstrap**
    - Detail pages provide initial guesses for parameters
    - \( P(R_i = r_i) \)
    - Evidence about where records start: \( P(S_i = \text{true}) = 1 \)
    - Token types of columns \( P(T_j | C_i) \)
  - **Structure**
    - Table has \( \pi \) columns specified by the underlying database schema
    - However, not every record will have an attribute for every field, i.e., not every record has \( \pi \) fields
    - Number of fields in a record estimated from data

Learning Algorithm

- **Use EM to implement the inference algorithm**
  1. Initial guess for \( \pi_k \) for each record \( j \)
  2. For each potential record, update \( P(C_i | T, C_{i-1}) \)
  3. Update \( P(S_j | C_i) \)
  4. Update \( P(R_j | R_{i-1}, D_i, S_i) \)

  Result is the most likely assignment of data to \( R \) and \( C = \text{record segmentation} \)

Validation

- **Input data**
  - list and detail pages from 12 sites in domains: book sellers, property tax, white pages, corrections

- **Metrics**
  - \( P = \text{Cor} / (\text{Cor} + \text{InCor} + \text{NonRecords}) \)
  - \( R = \text{Cor} / (\text{Cor} + \text{UnsegRecords}) \)
  - \( F = 2PR/(P + R) \)

- **Results**
  - CSP approach: \( P = 0.85, R = 0.84, F = 0.84 \)
  - Probabilistic approach: \( P = 0.74, R = 0.99, F = 0.85 \)
  - Good performance for an automatic algorithm!
Discussion of Results

- CSP approach is very reliable on clean data, but sensitive to errors in data source
  - Attribute has one value on list page and another on detail page
- Probabilistic approach tolerates inconsistencies and is more expressive
- Combination of two techniques may be more robust

Comparison with RoadRunner

- RoadRunner System (Crescenzi et al, 2001)
  - Automatically learns the page and table template by exploiting similarities in page layout (HTML tags)
  - Uses the template to automatically extract data
  - Does not allow for disjunctions
    - Disjunctions are necessary to represent alternative layout instructions for the same field

Conclusion

- Domain-independent approach for automatically extracting and segmenting data from Web tables
- Approach leverages additional information provided by Web site structure
  - Logic based approach
    - Information provided by detail pages encoded as constraints and solved to obtain record segmentation
  - Probabilistic inference approach
    - Information provided by detail pages and table structure represented as a probabilistic model
    - Use inference to learn proper segmentation
- Validated approach on 12 Web sites from diverse information domains
  - Efficient, accurate performance, F=0.85 and F=0.84