A Flexible Learning System for Wrapping Tables and Lists in HTML Documents

Cohen, Hurst, Jensen

WWW2002
Outline

• Wrapping and learning wrappers for HTML pages.
• Appropriate document representation for automated wrapper learning.
• A learning system for wrapper induction.
• Extracting from tables.
• Experiments.
• Conclusions.
Summary

• Repeated relation instances have similar or identical layout (efficient communication), though not necessarily identical encoding.
• The syntax of the encoding may not describe the syntax of the document element (better views of the document required).
• Instructing a learning system by examples requires capture of the users intentions (more examples reduce ambiguity, shared document idiom reduces examples).

WheezeBong.com:
Contact Info

Currently we have offices in three locations (original locations in bold):
• **Pittsburgh**, PA
• **Provo** UT
• Honolulu, HI

</b><i>Pittsburgh</i></b></i>

</i><b>Provo</i></b>
Wrapping and Learning Wrappers for HTML Pages.

- Aim – to be able to access structured (and relational) information on web pages programmatically, as if accessing a database.
- Input – some HTML and a query expression, Output – a set of relations.
- Simple solution – hand crafted scripts developed for each page; the query is implicit in the script.
- Ideal solution – machine generated programs produced with a minimum of human instruction; the query is elicited from the trainer.
Design Challenges

• Be future proof (tune using bias).
• Know when applicable and when broken.
• HTML –
  – There are many ways to express a particular layout (e.g. tabular layout can be achieved in a variety of ways).
  – Consistency in encoding is not required for visual consistency (e.g. ordering of text modifying tags).
• Inform the learning process in an intuitive and consistent manner.

WheezeBong.com:
Contact Info

Currently we have offices in two locations:
• Pittsburgh, PA
• Provo, UT
• Honolulu, HI
Appropriate Document Representation

• The DOM is an obvious starting point.
• Other appropriate views of the document
  – Token sequence.
  – The result of certain normalizations of the DOM.
  – A representation of the layout produced by the HTML when rendered in a browser.
  – …
• Each document view is valid for learning certain types of concepts/document regularities.
• A learning system should be able to make use of multiple views and be extensible as new views are made available/demanded by new problems.
A Learning System for Wrapper Induction

• Premise: A wrapper learning system needs careful engineering
  – 6 hand-crafted languages in WIEN (Kushmeric AIJ2000)
  – 13 ordering heuristics in STALKER (Muslea et al AA1999)

• Approach: Architecture that facilitates hand-tuning the “bias” of the learner.
  – Bias is an ordered set of “builders”
  – Builders are simple “micor-learners”
  – A single master algorithm co-ordinates learning.
A Learning System for Wrapper Induction

• A span is a subset of the document defined by a start and end point in the DOM.
A Learning System for Wrapper Induction

- A predicate is a binary relation on spans: \( p(s_1, s_2) \) means that \( s_2 \) is extracted from \( s_1 \).
- Membership can be tested:
  - Given \((s_1, s_2)\) is \( p(s_1, s_2) \) true?
- Predicates can be executed:
  - EXECUTE\((p, s_1)\) is the set of \( s_2 \) for which \( p(s_1, s_2) \) is true.
- Example:
  - \( p(s_1, s_2) \) iff \( s_2 \) are the tokens below an \(<_<\) in \( s_1 \)
  - EXECUTE\((p, s_1)\) extracts:
    - Pittsburgh, PA
    - Provo, UT
    - Honolulu, HI
  - \( p(s_1, s_2) \) iff \( s_2 \) starts with ‘P’…

WheezeBong.com:
Contact Info
Currently we have offices in three locations:
- Pittsburgh, PA
- Provo, UT
- Honolulu, HI
A Learning System for Wrapper Induction

- Predicates are implemented by simple languages.
- \( L_{\text{bracket}} \): \( p \) is defined by a pair of strings \([l, r]\) and \( p_{[l, r]}(s_1, s_2) \), is true iff \( s_2 \) is preceded by \( l \) and followed by \( r \).
- \( \text{EXECUTE}(p_{\text{for, Induction}}, s_1) = \{ \text{“Wrapper”} \} \)
- \( L_{\text{tagpath}} \): \( p \) is defined by a sequence of tags \( \{t_0, t_1, \ldots, t_k\} \) and \( p_{\{t_0, t_1, \ldots, t_k\}}(s_1, s_2) \) is true iff \( s_1 \) and \( s_2 \) correspond to DOM nodes and \( s_2 \) is reached from \( s_1 \) by following a path ending in \( t_0, t_1, \ldots, t_k \).
A Learning System for Wrapper Induction

• For each language \( L \) there is a builder \( B_L \) which implements a few simple operations:
  – \( \text{LGG(positive examples of } p(s1, s2)) \): least general \( p \) in \( L \) that covers all the positive examples.
    • For \( L_{\text{bracket}} \), longest common prefix and suffix of the examples.
  – \( \text{REFINE}(p, \text{examples}) \): a set of \( p \)'s that cover some but not all of the examples.
    • For \( L_{\text{tagpath}} \), extend the path with one additional tag that appears in the examples.
The Learning Algorithm

• Inputs
  – An ordered set of builders (order defines bias).
  – Positive examples of the predicates to be learned (negative examples can be inferred).

• Algorithm
  – Compute the LGG of positive examples for each builder.
  – If any LGG is consistent with the implicit negative data, then return it (break ties in favour of earlier builders).
  – Otherwise, execute the best LGG to get explicit negative examples, then apply a FOIL-like learning algorithm using LGG and REFINE to create “features”.

### Extraction from Tables

<table>
<thead>
<tr>
<th>Actresses</th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>Lucy</td>
<td>Lawless</td>
<td>Images</td>
<td>Links</td>
</tr>
<tr>
<td>Angelina</td>
<td>Jolie</td>
<td>Images</td>
<td>Links</td>
</tr>
<tr>
<td>...</td>
<td>...</td>
<td>...</td>
<td>...</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Singers</th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>Madonna</td>
<td>Images</td>
<td>Links</td>
<td></td>
</tr>
<tr>
<td>Brittany</td>
<td>Shakespeare</td>
<td>Images</td>
<td>Links</td>
</tr>
<tr>
<td></td>
<td>...</td>
<td>...</td>
<td>...</td>
</tr>
</tbody>
</table>
Extraction from Tables

- Classify HTML TABLE nodes as true tables or non tables.
- Render each table into a logical representation using an extended version of the HTML table rendering algorithm.
- Record the true logical table position of each cell (not limited to TD/TH nodes).
- Determine the geometric role of each cell – {cut-in, normal}.
Extraction from Tables: Deriving an Abstract Model

- Start with the standard table rendering algorithm.
- Attempt to “inline” complex geometric blocks:
  - Nested tables
  - Lists
  - BR separated lines
  - Plain text

<table>
<thead>
<tr>
<th>Tel:</th>
<th>123 456</th>
</tr>
</thead>
<tbody>
<tr>
<td>Fax:</td>
<td>123 567</td>
</tr>
</tbody>
</table>

```html
<table>
<tr><td>Tel:</td><td>123 456</td></tr>
<tr><td>Fax:</td><td>123 567</td></tr>
</table>
```
**Extraction from Tables**

<table>
<thead>
<tr>
<th>Actresses [cut-in cell] {0,0 – 3, 0}</th>
</tr>
</thead>
<tbody>
<tr>
<td>Lucy</td>
</tr>
<tr>
<td>Angelina</td>
</tr>
<tr>
<td>…</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Singers [cut-in cell] {0,4 – 3,4}</th>
</tr>
</thead>
<tbody>
<tr>
<td>Madonna</td>
</tr>
<tr>
<td>Brittany</td>
</tr>
<tr>
<td>…</td>
</tr>
</tbody>
</table>

- Element name, words in last cut-in. (e.g. table cells where the last cut-in was “singers”)
- Tagpath builder with awareness of co-ordinates. (e.g. table cells with y = 5).
Experiments: real world wrapping problems

<table>
<thead>
<tr>
<th>Problem</th>
<th>WL2</th>
<th>Problem</th>
<th>WL2</th>
</tr>
</thead>
<tbody>
<tr>
<td>JOB1</td>
<td>3</td>
<td>CLASS1</td>
<td>1</td>
</tr>
<tr>
<td>JOB2</td>
<td>1</td>
<td>CLASS2</td>
<td>3</td>
</tr>
<tr>
<td>JOB3</td>
<td>1</td>
<td>CLASS3</td>
<td>3</td>
</tr>
<tr>
<td>JOB4</td>
<td>2</td>
<td>CLASS4</td>
<td>3</td>
</tr>
<tr>
<td>JOB5</td>
<td>2</td>
<td>CLASS5</td>
<td>6</td>
</tr>
<tr>
<td>JOB6</td>
<td>9</td>
<td>CLASS6</td>
<td>3</td>
</tr>
<tr>
<td>JOB7</td>
<td>4</td>
<td>Median</td>
<td>3</td>
</tr>
<tr>
<td>Median</td>
<td>2</td>
<td>Median</td>
<td>3</td>
</tr>
</tbody>
</table>
Experiments: real world wrapping problems

Average accuracy versus number of training examples

Baseline
No Tables
No Format
Conclusions

• Wrapper learners need tuning. Structuring the bias space provides a principled approach to tuning.

• Builders let one mix generalization strategies based on different views of the document:
  – DOM
  – Token sequence
  – Table model
  – …

• Multiple views of the document improve performance by allowing builders to express concepts in appropriate language.

• (The TABLE element is a good example of a specification that failed – cf Keynote talk, TBL).