AUTOMATED INFORMATION EXTRACTION FROM WEB PAGES
USING AN INTERACTIVE LEARNING AGENT

JUGAL K. KALITA AND PARITOSH ROHILLA
Dept. of Computer Science, University of Colorado, Colorado Springs CO 80933, USA

Due to the dynamic nature of the Web the layout of information on a Web page can change often. If a comparison-shopping agent relies on the programmer to detect changes in the layout and change the information extraction algorithms accordingly, the agent's efficiency and accuracy are compromised. The process of manually changing code is cumbersome. In addition, an agent built with hard-coded logic specific to a Web site works only for that domain. We have built a GUI based system, which enables the agent to learn to extract product information from a Web page. The algorithms use machine learning to help make the agent generic and easily adaptable to various product domains. We avoid any hard coding. In addition, the system is able to learn the desired information based upon just few training samples. Such a capability enables adding new sites for a product category relatively easy.

1 System Overview

The following steps are involved in semi-automatically extracting relevant information from Web pages for comparison-shopping:

1. A structure for the relevant information on Web pages needs to be specified.
2. The learning engine needs samples that fit the structure defined.
3. From the training samples, the learning engine produces extraction rules.
4. The extraction rules are applied to Web pages to extract relevant information. These results also determine if more training samples are necessary.
5. The rules learnt can be fine-tuned if the Learner cannot capture all details.

Our system has two programs, namely the Learner and the Extractor. Both programs interface with a common database. The Learner has modules for the first three tasks. The Extractor handles the last two.

Our approach uses the inherent structure of tags and syntactic properties of plain text to infer rules. Our approach differs from published approaches based on wrappers or other techniques [1,2,5]. The entire page rather than being broken up into tokens is converted into a document tree. The tree is made up of tags and plain text nodes. The Learner tries to identify a node of interest by exploiting the properties of this tree and the plain text nodes.

2 The Learner

The rules learnt by the Learner for a particular page are stored in the database. The Extractor uses these rules to extract records from target Web pages.
2.1 Structure specification

Most Web pages that provide information about products have an inherent structure. This segment of the page can be thought of as consisting of several records. A record is a group consisting of coherent pieces of related information [3]. Thus each record has several fields. Out of the fields a record has, we may be interested only in a few selected ones. For example, the relevant fields of a record (say, book) may be like the ones shown in figure 1. While defining the structure of the records, we can mark some fields as mandatory. For example, in constructing a comparison-shopping agent for books, the mandatory fields for each record can be Title and Price. Besides providing names for various fields of the record structure, it may be helpful to also provide additional information such as the data type of the field. The extraction algorithm uses the data type information to reject nodes that do not match.

2.2 Providing training samples

Training samples shown to the Learner are records contained in Web pages. Several Web pages representative of the same record structure are obtained. The Learner has a GUI interface, which facilitates the learning process. A screen shot of the GUI is shown in figure 2. A user loads sample pages one at a time. Once loaded, a sample page looks like a text file without tags. The entire Web page is first converted into a document tree. The plain text nodes, which appear in the display area, are indented according to their depth in the tree. The indentation gives a feel of rendering. This can help the human trainer recognize record boundaries. Every node in the tree is
2.3 Generation of extraction rules

Extraction rules are learnt for every element of the record. Key properties of the document tree are utilized to formulate rules. Figure 3 shows a document tree corresponding to parts of a Web page. Our goal is to develop rules to extract fields that comprise records. The document tree for any Web document containing several records shows a number of interesting features:

1. Most records in the tree have a similar pattern. The fields show common properties across records, e.g., they are at the same depth, and have parent nodes with the same tag.
2. Every node in a document tree has a unique node number. This uniqueness property helps avoid conflicts while grouping identical looking fields into their respective records.
3. All plain text nodes show up as leaf nodes. Thus, the problem of identifying a field now narrows down to identifying the appropriate leaf node.

The following information is gathered for all the fields of various sample records that are shown by the trainer:

- The depth of a node in the document tree is recorded. Since all records have an identical pattern, it is very likely that this field is always at the same depth across all records.
- For each field, we find the sequence of tags, starting from the root. For example, in Figure 3 the node containing “Artificial Intelligence” has the tag sequence “html;body;table;tr;td;b”.

![Figure 3. A document tree](image-url)
• The relative position of a field, the difference between its node number and the node number of the first field in the record, is recorded.
• Any number of word(s) or character(s) that stay constant across all records of a field are keywords for that field. Keywords can help in resolving ambiguity.
• Any number of word(s) or character(s) that should not be part of the plain text of the field are classified as omitwords for that field. Any plain text nodes matching the omitwords are ignored at extraction time.
• The entire text associated with the field is also stored. We attempt to infer characteristics of the field by examining the text of a field across all records. For example, we can find the average size of the text in the field.

The rule generation algorithm uses all of the information gathered above to formulate rules for each field of the record.

3 The Extractor

The Extractor extracts and displays the records from the loaded document. The trainer specifies the rule set to be applied to the document. Having two GUIs, one for the Learner and the other for the Extractor helps the trainer to immediately view results of the samples that he provides to the Learner. Based upon the results he can either stop the learning process or continue to provide more samples.

3.1 Applying Extraction Rules

The extraction module is a rule-based deduction system [4]. We have established the following general antecedent-consequent rules for each field of the record structure:
• if depth of node = learned depth ∧ tag sequence of node = learned tag sequence then node is a candidate node.
• if node is a candidate node ∧ node text has the specified data type ∧ node has learned keywords ∧ node doesn’t have learned omitwords ∧ text length is between min and max values then node belongs to the field.

The extraction process follows a bottom up approach to form records. This approach helps deal with records that do not have all fields. Every node that qualifies as a field is extracted from the page, irrespective of the record it belongs to. The extracted fields are then grouped together into records.

3.2 Rule refinement

The Extractor GUI provides a facility to look at the rules and make manual changes. This facility is handy if the algorithms are unable to calculate proper values for the rules. It is advisable that the trainer does not use this feature often.
4 Experimental Results

We selected 11 Web sites that sell products belonging to different product categories. We filled product search forms on each of these Web sites and obtained pages showing product listings. We then ran our Learner and Extractor programs. Table 1 shows the results from some of our experiments.

<table>
<thead>
<tr>
<th>Web Sites</th>
<th>Time (Min)</th>
<th>Total Samples</th>
<th>Records Expected</th>
<th>Records Extracted</th>
<th>Wrong Records</th>
<th>Inconsistent Records</th>
<th>Fine Tune</th>
<th>Recall %</th>
<th>Precision %</th>
</tr>
</thead>
<tbody>
<tr>
<td>Barnes &amp; Noble</td>
<td>6</td>
<td>5</td>
<td>55</td>
<td>45</td>
<td>0</td>
<td>0</td>
<td>Y</td>
<td>100</td>
<td>100</td>
</tr>
<tr>
<td>Book Coop</td>
<td>10</td>
<td>4</td>
<td>157</td>
<td>157</td>
<td>0</td>
<td>0</td>
<td>Y</td>
<td>100</td>
<td>100</td>
</tr>
</tbody>
</table>

All online stores considered have simple record structures. The time that a trainer can spend trying to get our system to learn to extract the records can vary between 15 – 40 minutes. These also include the time it took to count and determine if the extracted records were incomplete or wrong. The experiments indicate that given sufficient amount of time our system can achieve a recall rate of 100% for all stores. The precision is 100%. On almost all Web sites, the rule refinement involved changing only the minimum and maximum values for the length of the text that can appear in a field. We think this overhead could have been reduced by more careful selection of sample records. For some Web documents we were able to achieve a recall rate of over 75% without fine-tuning. The final rules that were learnt show a very impressive recall and precision rate.

References