

# Tuples Extraction from HTML Using Logic Wrappers and Inductive Logic Programming

Costin Bădică<sup>1</sup>, Amelia Bădică<sup>2</sup>, and Elvira Popescu<sup>1</sup>

<sup>1</sup> University of Craiova, Software Engineering Department  
Bvd.Decebal 107, Craiova, RO-200440, Romania  
{badica.costin, elvira.popescu}@software.ucv.ro

<sup>2</sup> University of Craiova, Business Information Systems Department  
A.I.Cuza 13, Craiova, RO-200585, Romania  
ameliabd@yahoo.com

**Abstract.** This paper presents an approach for applying inductive logic programming to information extraction from HTML documents structured as unranked ordered trees. We consider information extraction from Web resources that are abstracted as providing sets of tuples. Our approach is based on defining a new class of wrappers as a special class of logic programs – *logic wrappers*. The approach is demonstrated with examples and experimental results in the area of collecting product information, highlighting the advantages and the limitations of the method.

## 1 Introduction

The Web was designed for human consumption rather than machine processing. This hinders the task of extracting useful information from it. Therefore the problem of automating information extraction (IE hereafter) from the Web has received a lot of attention in the last years ([1, 3, 6, 4, 11, 9]).

This paper deals with automating IE from HTML documents using inductive logic programming (ILP hereafter). IE is concerned with locating specific pieces of information in text documents including HTML and XML. A program that is actually used for performing the IE task is called a *wrapper* ([7]). ILP studies learning from examples within the framework provided by clausal logic.

IE from HTML uses two document models: a linear or string-based model and a tree-based model. It is appreciated that tree-based wrappers are more expressive than string-based wrappers ([11, 9]). In our work we consider HTML documents modeled as unranked ordered trees. IE approaches that adopted tree-based models were also reported in [6, 11, 3]. Recently, [9] proposed the combination of string and tree wrappers to yield more powerful wrappers.

The paper is a follow-up of our previous work reported in [1, 2]. In [1] we presented a methodology and experimental results on applying ILP to IE from HTML documents that represent printer information. The task was to extract single fields. In [2] we enhanced the original approach by enriching the target document representation and using an explicit set of negative examples, rather than treating all non-positive examples as negative examples.

In this paper we adopt a relational data model of the information resources and we are focusing on the task of tuples extraction. Many Web resources can be abstracted in this way, including: search engines result pages, product catalogues, news sites, product information sheets, a.o. Their content is presented to the human user as HTML documents formatted using HTML tables or lists. We consider two problems: extracting tuples from a rectangular table and extracting tuples from a nested table with the information organized hierarchically.

Our paper is structured as follows. Section 2 contains some background on IE, tree-based document representations using logic programming and ILP. Section 3 contains a discussion of the problems addressed in our experiments and some experimental results. Section 4 concludes and points to future work.

## 2 Background

This section contains background on: IE, relational representation of XHTML documents, logic wrappers and the use of ILP for IE. More details are in [1].

### 2.1 The IE Process

We propose a generic IE process structured into the following sequence of stages: i) Collect HTML pages of interest using Web browsing and crawling and download them to a local repository; ii) Preprocess and convert the documents from HTML to XHTML; iii) Manually extract a few items and annotate accordingly the XHTML documents; iv) Parse the annotated XHTML documents and generate the input for the learner; v) Apply the learning algorithm and obtain the wrapper; vii) Compile the wrapper to an XML query language; viii) Apply the wrapper to extract new information from other documents.

Note that this IE process is parameterized according to the following dimensions: number of documents used for manual IE (stage iii), the learning technique (stages iv and v) and the IE engine (stages vii and viii).

### 2.2 L-Wrappers for IE from HTML

The idea of representing documents and wrappers using logic programming is not entirely new ([4, 5, 8, 12]). All of these approaches assume that the target document is represented as a list of tokens with features. We propose a relational representation of an XHTML document seen as an unranked ordered tree.

The *structure* of an XHTML document consists of a set of document nodes that are nested in a tree like structure. Each node has assigned a specific tag from a given finite set of tags  $\Sigma$ . There is a special tag  $text \in \Sigma$  that designates a text node.

The *content* of an XHTML document consists of the actual text in the text elements and the attribute-value pairs attached to the other document elements.

We assign a unique identifier (an integer) to each node of the document tree. Let  $\mathcal{N}$  be the set of all node identifiers.

The structure of an XHTML document is represented using three relations:

- i)  $child \subseteq \mathcal{N} \times \mathcal{N}$ ,  $(child(P, C) = true) \Leftrightarrow (P \text{ is the parent of } C)$ .
- ii)  $next \subseteq \mathcal{N} \times \mathcal{N}$ ,  $(next(L, N) = true) \Leftrightarrow (L \text{ is the left sibling of } N)$ .
- iii)  $tag \subseteq \mathcal{N} \times \Sigma$ ,  $(tag(N, T) = true) \Leftrightarrow (T \text{ is the tag of node } N)$ .

We introduce two sets to represent the content component of an XHTML document tree: the set  $\mathcal{S}$  of content elements, which are the strings attached to text nodes and the values assigned to attributes, and the set  $\mathcal{A}$  of attributes. The content of an XHTML document is then represented using two relations:

- i)  $content \subseteq \mathcal{N} \times \mathcal{S}$ ,  $(content(N, S) = true) \Leftrightarrow (S \text{ is the string contained by the text node } N)$ .
- ii)  $attribute\_value \subseteq \mathcal{N} \times \mathcal{A} \times \mathcal{S}$ ,  $(attribute\_value(N, A, S) = true) \Leftrightarrow (S \text{ is the string value of the attribute } A \text{ of the node } N)$ .

Based on our experimental results reported in [2], we found useful to enhance the representation of the document structure with two new relations:

- i)  $first \subseteq \mathcal{N}$ ,  $(first(X) = true) \Leftrightarrow (X \text{ is the first child of its parent node})$ .
- ii)  $last \subseteq \mathcal{N}$ ,  $(last(X) = true) \Leftrightarrow (X \text{ is the last child of its parent node})$ .

This representation allows the definition of *L-wrappers* (i.e. *logic wrappers*).

**Definition 1.** (*L-wrapper*) An L-wrapper is a logic program defining a relation  $extract(N_1, \dots, N_k) \subseteq \mathcal{N} \times \dots \times \mathcal{N}$ . For each clause, the head is  $extract(N_1, \dots, N_k)$  and the body is a conjunction of literals in a set  $\mathcal{R}$  of relations.

Every tuple of relation *extract* is extracted by the wrapper. Note that relation *content* is used to find the content of the extracted tuples. *attribute\_value* could also be used in the clause bodies to check additional constraints on attributes of extracted nodes. Note also that in our case  $\mathcal{R} = \{child, next, tag, first, last\}$ .

The manual writing of L-wrappers is a tedious and error-prone process requiring a careful analysis of the target document. Fortunately, it is possible to learn L-wrappers with a general purpose learning program.

### 2.3 Learning L-Wrappers Using ILP

ILP is a combination of inductive machine learning with representation of theories as logic programs. In our experiments we have used FOIL – First Order Inductive Learner, an ILP program developed by J. R. Quinlan at the beginning of the '90s ([10]). In what follows we assume that the reader has some familiarity with FOIL. For a detailed description see [10].

We mapped our training data as input for FOIL. We assumed that in the training stage the user selects a single training document and performs a few extraction tasks on it. FOIL's input is split in three parts:

- i) The definition of the relations arguments types. We used a single type that represents the set  $\mathcal{N}$  of all the nodes found in the training document.
- ii) The extensional definition of the target relation *extract*. The positive examples are the tuples manually extracted by the user. If negative examples are not provided explicitly then they are automatically generated by FOIL using the closed world assumption (i.e. all the other tuples that are not positive examples).

Speed/monthly volume	
Print speed, black (pages per minute)	Up to 15 ppm
Print speed, color (pages per minute)	Up to 11 ppm
Recommended monthly volume, maximum	12,000 pages
Print quality / technology	
Print technology	HP Thermal Inkjet
Print quality, black	up to 1200 x 600 dpi
Print quality, color	up to 1200 x 600 dpi on photo paper
Resolution technology	HP PhotoREt III
Paper handling / media	
Paper trays, std.	2
Paper trays, max.	2

**Fig. 1.** An XHTML document fragment and its graphic view

iii) The extensional definitions of the background relations *child*, *next*, *tag*, *first* and *last*. For convenience, we have replaced the relation *tag* with a set of relations  $r_\sigma$ , for all  $\sigma \in \Sigma$ , defined as  $r_\sigma = \{N | tag(N, \sigma) = true\}$ .

### 3 Experiments and Discussion

We performed experiments of learning L-wrappers for extracting printer information from Hewlett Packard's Web site. The information is represented in two column HTML tables (see figure 1). Each row contains a pair (feature name, feature value). Consecutive rows represent related features and are grouped in feature classes. For example there is a row with the feature name 'Print technology' and the feature value 'HP Thermal Inkjet'. This row has the feature class 'Print quality/technology'. So actually this table contains triples (feature class, feature name, feature value). Some triples may have identical feature classes.

In what follows we consider two experiments: i) tuples extraction from flat information resources, exemplifying with pairs (feature name, feature value); ii) coping with hierarchical information by extracting pairs (feature class, feature name). We have used the same test data as in [1], only the learning tasks were changed. We used examples from a single training document with 28 tuples.

#### 3.1 Tuples Extraction from Flat Information Resources

We performed an experiment of learning to extract the tuples containing the feature name and feature value from HP printer information sheets. The experimental results are reported in table 1, row 1.

Because the number of tuples exceeds the size of FOIL's default tuple space, we explicitly set this size to a higher value (300000) and we used a random sample representing a fraction of the negative examples in the learning process.

**Table 1.** Experiments with tuples extraction

Ex.no.	No.pos.ex	Frac.neg.ex	Prec.	Rec.	No.lit.
1	24, 28	20 %	0.959	1	14
2	24, 28	20 %	1	1	15

The L-wrapper learnt consisted of a single clause:

$$\begin{aligned} extract(A, B) \leftarrow & tag(A, text) \wedge tag(B, text) \wedge child(C, A) \wedge child(D, B) \wedge \\ & child(E, C) \wedge child(F, E) \wedge child(G, D) \wedge child(H, G) \wedge child(I, F) \wedge \\ & child(J, I) \wedge next(J, K) \wedge first(J) \wedge child(K, L) \wedge child(L, H). \end{aligned}$$

This rule extracts all the pairs of text nodes such that the grand-grand-grandparent of the first node ( $J$ ) is the first child of its parent node and the left sibling of the grand-grand-grandparent of the second node ( $K$ ).

### 3.2 Tuples Extraction from Hierarchical Information Resources

The idea of modeling nested documents using a relational approach and building wrappers accordingly to this model is not entirely new; see [7].

In this section we present an experiment of learning to extract pairs (feature class, feature name) from printer information sheets. Note that because we may have many features in the same class, the information is hierarchically structured. The experimental results are reported in table 1, row 2.

The L-wrapper learnt consisted of a single clause:

$$\begin{aligned} extract(A, B) \leftarrow & child(C, A) \wedge child(D, B) \wedge tag(C, span) \wedge child(E, C) \wedge \\ & child(F, E) \wedge next(F, G) \wedge child(H, G) \wedge last(E) \wedge child(I, D) \wedge child(J, I) \wedge \\ & child(K, J) \wedge child(L, K) \wedge next(L, M) \wedge child(N, M) \wedge child(H, N). \end{aligned}$$

There is one difference from the flat case – how examples are collected. In this case, some examples will share the feature class. Moreover, in the general case, some fields will need to be selected repeatedly during the manual extraction process (like the feature class in the printer example). This can be avoided by designing the graphical user interface that guides the example selection such that the tuple previously selected is always saved and thus its fields may be reused in the next selection.

### 3.3 Discussion

In our experiments we used the closed world assumption to generate the negative examples. This means that each tuple not given as positive example, automatically counts as negative example. Let  $d$  be the size of the training document (i.e. the number of nodes) and let  $k$  be the tuple arity. The number of negative examples is proportional with  $d^k$ , i.e. it is exponential in the tuple arity. For example, our documents had about 1000 nodes. This means that for tuples of arity 3, the total number of negative examples is about  $10^9$ .

For this reason we had problems with learning to extract tuples of arity greater than 2. Because the number of negative examples exceeded the memory available, we were forced to use for learning a random sample representing a very

small fraction (less than 0.1 %) of the total number of the negative examples. This had the effect of producing wrappers with a very low precision.

## 4 Conclusions

In this paper we have shown the advantages and limitations of applying ILP to learn L-wrappers for tuple extraction from HTML documents structured as trees. As future work we have in mind the investigation of the possibility to automatically translate our L-wrappers to an XML query language and to find clever methods to reduce the number of negative examples needed.

## References

1. Bădică, C., Bădică, A.: Rule Learning for Feature Values Extraction from HTML Product Information Sheets. In: Boley, H., Antoniou, G. (eds): *Proc. RuleML'04*, Hiroshima, Japan. LNCS 3323, Springer-Verlag (2004) 37–48
2. Bădică, C., Popescu, E., Bădică, A.: Learning Logic Wrappers for Information Extraction from the Web. In: Papazoglou M., Yamazaki, K. (eds.) *Proc. SAINT'2005 Workshops. Computer Intelligence for Exabyte Scale Data Explosion*, Trento, Italy. IEEE Computer Society Press, (2005) 336–339
3. Chidlovskii, B.: Information Extraction from Tree Documents by Learning Subtree Delimiters. In: *Proc. IIWeb'03*, Acapulco, Mexico (2003), 3–8
4. Freitag, D.: Information extraction from HTML: application of a general machine learning approach. In: *Proc. AAAI'98*, (1998), 517–523
5. Junker, M., Sintek, M., Rinck, M.: Learning for Text Categorization and Information Extraction with ILP, In: *Proc. Workshop on Learning Language in Logic*, Bled, Slovenia, (1999)
6. Kosala, R., Bussche, J. van den, Bruynooghe, M., Blockeel, H.: Information Extraction in Structured Documents Using Tree Automata Induction. In: *Principles of Data Mining and Knowledge Discovery, 6<sup>th</sup> European Conf.*, Helsinki, Finland, LNAI 2431, Springer-Verlag (2002), 299–310
7. Kushmerick, N.: Wrapper induction: Efficiency and expressiveness. *Artificial Intelligence*, No.118, Elsevier Science (2000), 15–68
8. Kushmerick, N., Thomas, B.: Adaptive Information Extraction: Core Technologies for Information Agents, In: *Intelligent Information Agents R&D in Europe: An AgentLink perspective* (Klusch, Bergamaschi, Edwards & Petta, eds.). LNAI 2586, Springer-Verlag (2003), 79–103
9. Ikeda, D., Yamada, Y., Hirokawa, S.: Expressive Power of Tree and String Based Wrappers. In: *Proc. IIWeb'03*, Acapulco, Mexico, (2003), 21–16
10. Quinlan, J. R., Cameron-Jones, R. M.: Induction of Logic Programs: FOIL and Related Systems, *New Generation Computing*, 13, (1995), 287–312
11. Sakamoto, H., Arimura, H., Arikawa, S.: Knowledge Discovery from Semistructured Texts. In: Arikawa, S., Shinohara, A. (eds.): *Progress in Discovery Science*. LNCS 2281, Springer-Verlag (2002) 586–599
12. Thomas, B.: Token-Templates and Logic Programs for Intelligent Web Search. *Intelligent Information Systems*. Special Issue: Methodologies for Intelligent Information Systems, 14(2/3) (2000), 241–261