

Learning Logic Wrappers for Information Extraction from the Web

Costin Bădică and Elvira Popescu
University of Craiova
Software Engineering Department
Bvd.Decebal 107, Craiova, 200440, Romania
badica_costin@software.ucv.ro

Amelia Bădică
University of Craiova
Business Information Systems Department
A.I.Cuza 13, Craiova, RO-1100, Romania
ameliabd@yahoo.com

Abstract

This paper discusses a methodology for applying general-purpose first-order inductive learning to extract information from Web documents structured as unranked ordered trees. The methodology is applied to information extraction from real-world HTML page sets that represent product information sheets, an important task in product data integration. The methodology addresses the problems of defining information extraction rules in the form of logic wrappers and mapping the task of learning these rules to general purpose first-order inductive learning.

1 Introduction

The Web is a continuously growing information repository with a rich semantic structure that spans many application areas. The Web, however, has been designed primarily for human consumption rather than automated processing. This is a major obstacle for automating tasks like information searching, filtering and extraction. Therefore the problem of Web content mining has received a lot of attention in the last years ([2, 3, 7, 12, 5, 15]).

This paper deals with automatizing information extraction (IE hereafter) from Web documents using general purpose first order inductive learning (FOIL hereafter). IE was originally concerned with locating specific pieces of information in text documents written in natural language ([10]) and then expanded rapidly to cover extraction tasks from networked documents including HTML and XML.

In our work we consider (semi-)structured information sources modeled as unranked ordered trees ([13]), rather than linear strings. This exploits the structural information conveyed by the document tree and has the advantage that the extracted item can depend on its structural context in a document, while this information is lost if the document is treated strictly linear. IE approaches that adopt the tree representation of documents are reported in [2, 7, 15, 3].

The paper is a follow-up of our previous work reported in [2]. There we presented a methodology and experimental results on applying FOIL to IE from HTML documents in the area of product data integration ([4, 1]). Here we discuss some problems and enhancements to the approach originally presented in [2] including: enriching the target document representation with predicates *first* and *last*, extending the approach to tuple extraction and using an explicit set of negative examples rather than using the closed world assumption (CWA hereafter).

The rest of the paper is structured as follows. Section 2 contains some background on IE, document representations using logic programming and inductive learning. 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 an overview: i) of the IE process, ii) of our relational representation of HTML documents and the corresponding logic wrappers and iii) on the use of FOIL to learn IE rules. More details can be found in [2].

2.1 The IE Process

We propose a generic IE process structured as a linear sequence of stages: i) Browse or crawl the Web and download HTML pages of interest; 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 an appropriate representation as input for the learning program; v) Apply the learning algorithm and obtain the wrapper; vi) Apply the wrapper to extract information.

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 (stage vi).

2.2 L-Wrappers for IE from HTML Documents

The idea of representing documents and wrappers using logic programming is not entirely new ([5, 6, 9, 16, 17]). All of these approaches assume that the target document is represented as a linear list of tokens with features.

For example, in the SRV system described in [5], the relations from the rule bodies check various token features like: length, position in the text fragment, if they are numeric or capitalized, a.o. SRV was adapted to learn IE rules from HTML. For this purpose new token features were added to check the HTML context in which a token occurs.

In our approach we treat from the very beginning an XHTML document as an unranked ordered tree ([13]).

The *structure* of an XHTML document consists of a set of document nodes nested in a tree like structure. Each node has assigned a specific tag from a finite set of tags Σ . 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 tree can be represented using three relations:

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

To represent the content component of an XHTML document tree, we introduce two sets: 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. With these notations, the content part of an XHTML document tree can be represented using two relations:

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

Note that if X is an unbound variable and Y is a bound variable then $\neg next(X, Y) = true$ means that Y is instantiated with the first child of its parent node. Therefore we can introduce the relation $first$ defined as $(first(X) = true) \Leftrightarrow (X \text{ is the first child of its parent node})$ and replace $\neg next(X, Y)$ with $first(Y)$. Similarly, if X is a bound variable and Y is an unbound variable then $\neg next(X, Y) = true$ means that Y is instantiated with the last child of its parent node. Therefore we can introduce the relation $last$ defined as $(last(X) = true) \Leftrightarrow (X \text{ is the last child of its parent node})$ and replace $\neg next(X, Y)$ with $last(X)$.

Based on our previous results in [2] we found useful to enhance the representation of an XHTML document with predicates $first$ and $last$. This representation allows the definition of logic wrappers for IE extraction. Let us call them *L-wrappers* (i.e. *logic wrappers*).

Definition 1 (*L-wrapper*) An L-wrapper is a logic program defining a relation $extract(N_1, \dots, N_k) \subseteq \mathcal{N} \times \dots \times \mathcal{N}$ as a set of clauses. The head of a clause is $extract(N_1, \dots, N_k)$ and the body of a clause 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 $content$ can be used to find the actual content of the extracted tuples and $attribute_value$ could also be used in the clause bodies to put additional constraints on nodes attributes. Note also that in our case $\mathcal{R} = \{child, next, tag, first, last\}$.

For example, if we want to extract all the pairs of text nodes of an XHTML document such that the parent of the first node is the left sibling of the parent of the second node, we can use an L-wrapper with the following IE rule:

$$extract(A, B) \leftarrow tag(A, text) \wedge tag(B, text) \wedge \\ child(C, A) \wedge child(D, B) \wedge next(C, D)$$

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 FOIL

FOIL – First Order Inductive Learner is a general purpose first order inductive learning program developed by J. R. Quinlan and his team at the beginning of the '90s ([14]).

The goal of FOIL is to construct a logic program that represents an appropriate intensional definition of a target relation in terms of a given set of background relations. The *target relation* (the relation that is learnt) is defined extensionally as a set of tuples that is usually partitioned into a set of positive tuples and a set of negative tuples. The learnt definition must cover all the positive tuples (or a large fraction of them) and no negative tuples (or a small fraction of them). The *background relations* are also defined extensionally as sets of tuples. See [14] for more details.

An important step in our IE experiments was to prepare the input for FOIL. We assume that during the training stage the user selects a single XHTML document – the training document, and performs a few extraction tasks on it. The problem is to map the training data into input for FOIL.

In our experiments, FOIL's input has three components:

- 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.

Speed/monthly volume	
Print speed, black (pages per minute)	Up to 50 ppm
First page out, black	8 secs
Processor speed	300 MHz

Figure 1. An XHTML document fragment and its graphic view

ii) The extensional definition of the target relation *extract*. The positive examples are the tuples manually extracted by the user in the training stage. If negative examples are not provided explicitly then they are automatically generated by FOIL using CWA.

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_σ , for all $\sigma \in \Sigma$, defined as $r_\sigma = \{N | tag(N, \sigma) = true\}$.

Paper [2] contains a discussion of the FOIL’s command-line parameters that we found useful in our experiments.

3 Experiments and Discussion

We performed experiments of learning L-wrappers for extracting printer information from Hewlett Packard’s Web site. The printer information is represented in two column HTML tables (see figure 1 and [2] for more details).

In what follows we consider three problems: i) explicit enumeration of negative examples; this approach is compared with our results from [2]; ii) combining the use of relations *first* and *last* in document representation with no negative literals in the body of the learnt clause; iii) extraction of pairs (feature name, feature value).

3.1 Explicit Enumeration of Negative Examples

The use of CWA has the advantage that the negative examples do not have to be explicitly enumerated. But this has also a serious drawback: if only a part of the relevant items are annotated and used as positive examples (i.e. the user has manually extracted some, but not all the useful items) then, by applying CWA, the set of negative examples will also include the rest of the items to be further extracted.

One approach to avoid this is to let the user explicitly indicate one or more fragments of the training document where the relevant information is located and then explicitly generate the negative examples by excluding all the items

Table 1. Experiment with *first* and *last* and no negated literals

Ex.no.	No.pos.ex	Cl.acc.	Prec.	Rec.
1	1	0 %	0.436	1
2	3	0 %	0.448	1
3	4	0 %, 10 %	1	1
4	12	20 %, 40 %	1	0.800
5	24	20 %, 60 %	1	1
6	28	80 %	1	1

inside these fragments. For example, for the HP printer information sheets, the relevant information is located in the table with the list of printer features.

We performed experiments with this approach and compared the results with the CWA approach from [2], using the same training document and the same sets of items manually annotated. In all the experiments with 6 or more positive examples (allowing the use of negated literals, as in [2]), the values of precision and recall were 1.

3.2 Using *first* and *last*

Analyzing the wrappers learnt in the experiments from [2] we noticed that all the occurrences of $\neg next$ could be replaced with *last* and *first*. Taking also into account that the computation of the best literal during the top-down learning process is very expensive, we thought to improve a bit the learning time by combining *first* and *last* in the document representation with no negated literals in the learnt clauses. The results are reported in table 1.

3.3 Extending the Approach to Tuple Extraction

Learning IE rules for tuple extraction proceeds similarly with the single item case. The difference is that the *extract* relation is now described as a list of positive and negative tuples rather than a list of positive and negative items.

We performed an experiment of learning to extract the tuples containing the feature name and feature value from HP printer information sheets. In this experiment we have used the *first* and *last* relations in the document representation, the CWA approach to generate negative examples and no negated literals in the wrappers. The experimental results are reported in table 2.

Note that the number of negative examples grows exponentially with the tuple arity. Because this number exceeds the size of FOIL’s default tuple space, we have explicitly set this size to a higher value and the fraction of the negative examples actually used in the learning process to 20%.

An L-wrapper example learnt in this experiments is:

$$extract(A, B) \leftarrow tag(A, text) \wedge tag(B, text) \wedge child(C, A) \wedge child(D, B) \wedge child(E, C) \wedge child(F, E) \wedge child(G, D) \wedge child(H, G) \wedge child(I, F) \wedge child(J, I) \wedge$$

Table 2. Experiment with tuples extraction

Ex.no.	Neg.lit.	No.pos.ex	Cl.acc.	Prec.	Rec.	No.lit.
1	Yes	24, 28	80 %	0.959	1	14

$next(J, K) \wedge first(J) \wedge child(K, L) \wedge child(L, H).$

This rule extracts all the pairs of text nodes such that the grand-grand-grand-grandparent of the first node (node J) is the first child of its parent node and the left sibling of the grand-grand-grand-grandparent of the second node (node K). By analyzing the XHTML printer information sheets using an appropriate XML tool like Oxygen¹, it can be noticed that this rule correctly matches all the pairs (feature name, feature value).

4 Conclusions

In this paper we have shown that FOIL is useful for inducing L-wrappers for tuple extraction from HTML information sources and that learning can be improved by enhancing the document representation or letting the user to specify document fragments where the useful information is located. As future work we have in mind the investigation of the possibility to translate our L-wrappers to an XML query language and the development of tool support to enable experiments on significantly larger page sets.

References

- [1] Bădică, C., Bădică, A., Lițoiu, V.: Enhancing WWW E-Commerce by Acquiring and Managing Product Knowledge. In: *Proc. TAINN'03*, vol.E-7, Çanakkale, Turkey (2003) 684–692.
- [2] 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.
- [3] Chidlovskii, B.: Information Extraction from Tree Documents by Learning Subtree Delimiters. In: *Proc. IJCAI-03 Workshop on Information Integration on the Web (IIWeb-03)*, Acapulco, Mexico (2003), 3–8.
- [4] Fensel, D., Ding, Y., Omelayenko, B., Schulten, E., Botquin, G., Brown, M., Flet, A.: Product Data Integration. *IEEE Intelligent Systems*, Vol. 16, No.4, IEEE Computer Society (2001) 54–59.
- [5] Freitag, D.: Information extraction from HTML: application of a general machine learning approach. In: *Proc. AAAI'98*, (1998), 517–523.
- [6] 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).
- [7] 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, 6th European Conf.*, Helsinki, Finland, LNAI 2431, Springer-Verlag (2002), 299–310.
- [8] Kushmerick, N.: Wrapper induction: Efficiency and expressiveness. *Artificial Intelligence*, No.118, Elsevier Science (2000), 15–68.
- [9] Kushmerick, N., Thomas, B.: Adaptive Information Extraction: Core Technologies for Information Agents, In: *Intelligent Information Agents R&D in Europe: An AgentLink perspective* (Klusck, Bergamaschi, Edwards & Petta, eds.). LNCS 2586, Springer-Verlag (2003).
- [10] Lenhart, W., Sundheim, B.: A Performance Evaluation of Text-Analysis Technologies. In: *AI Magazine*, 12(3) (1991), 81–94.
- [11] Mitchell, T.M.: *Machine Learning*, McGraw-Hill, (1997).
- [12] Muşlea, I.: Extraction Patterns for Information Extraction Tasks: A Survey. In: *Proc. AAAI-99 Workshop on Machine Learning for Information Extraction*, (1999).
- [13] Neven, F.: Automata Theory for XML Researchers. *SIGMOD Record*, 31(3) (2002), 39–46.
- [14] Quinlan, J. R., Cameron-Jones, R. M.: Induction of Logic Programs: FOIL and Related Systems, *New Generation Computing*, 13, (1995), 287–312.
- [15] 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.
- [16] Thomas, B.: Anti-Unification Based Learning of T-Wrappers for Information Extraction. In: *Proc. AAAI-99 Workshop on Machine Learning for Information Extraction*, (1999).
- [17] 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.

¹<http://www.oxygenxml.com>