RULE LEARNING FROM SEMI-STRUCTURED DOCUMENTS BY INDUCTIVE LOGIC PROGRAMMING

Toshiharu Takeuchi

Department of Computer Science, Tokyo Institute of Technology
2-12-1 Ookayama, Meguro-ku, Tokyo, 152-8552 Japan
E-mail:takeuchi@mi.cs.titech.ac.jp

ABSTRACT

One of the hot research areas is knowledge discovery on structured documents like HTML and XML documents. In the case of XML documents, most popular approach to mining a knowledge is structural approach which find some kind of similar pattern(often tree structure or XPath) in interested XML documents. On the other hand, there is relational data mining approach such as ILP(Inductive Logic Programming). In this article relational approach for mining data from XML documents is proposed. To achieve relational data mining from XML documents, relations (or Background knowledge) about element or attribute in a XML documents is dynamically generated from schema language of XML such as DTD (Data Type Definition) or "XML Schema". In this article, case of DTD is given. The objective of this paper is to introduce general-purpose and domain independent relational data mining tool for valid XML document by inductive logic programming.

1. MOTIVATION AND INTRODUCTION

XML(eXtensive Markup Language)[1] is markup language for describing data. An XML document has similar structure that an HTML document has, but as name indicates, XML is extensible with tag name and attribute it has where HTML is fixed. Extensibility of XML allow one to create a XML document which incorporate data and structure of data to one entity. For this flexibility, XML becoming not only a standard data format in web and but also format of large-scale knowledge resource. Since XML is extensive, it is quite natural that structure of a XML document is reflection of concept structure of data stored in the document.

All XML documents must be **well-formed** document: a start tag < . . . > must have a corresponding end tag </ . . . > and every tag must be appropriately nested. Structure of a XML documents is validated by schema language such like *DTD(Data Type Definition)* or *XML Schema*[2]. If an XML document is validated by schema, the document is called **valid**. Since every tag in XML document is nested, the well-formed XML document can be seen as tree with its nodes are labeled by tag name and leaf node is string data or tag elements with no child subelement. Fig.1 shows a simple valid name list XML document and its DTD. In this XML document a tag element `member` indicate each person and its subelemnent `name`,`PR`,`URL` and `email` is the data what these name indicate. A tag of `name` and `PR` has its data as subelemnt of string. A tag of `URL` and `email` has its data in attribute. Attribute id in tag `member` is identifier of element and attribute friend is reference of other element specified by id value.

In Fig.1, the structure of XML document is validated by DTD also in Fig.1. In the case of DTD, there are (1).declarations about sort and ordinal of subelement of XML tag (please read the DTD declaration in Fig.1 `(name, PR,(email|URL)*)` as same manner of regular expression). (2).declarations about name and data type of attribute of tag (`<!ATTLIST member id ID #REQUIRED>` declares tag member has attribute `id` and type of attribute is ID value - value of attribute must not duplicate).

Research of data mining from a XML document mostly focus on extracting structural pattern of the document. *XPath*[3] or *XQuery*[4] is commonly used to represent structual pattern of interested XML documents. As the name XPath indicates, this approach extract common structure of interested XML documents. An XML document has tree structure, clustering based on tree structure is also done[1][2].

On the other hand, ILP(Inductive Logic Programming) is research area of machine learning that represent learned concept by logic program(e.g. Prolog). The typical goal of ILP system is given a set of background knowledge(predicate) BK and set of positive and negative instances of target predicate to be learned (P and N), find hypothesis H such like satisfy follow.

$$BK \wedge H \models P$$
$$BK \wedge H \not\models N$$

Since ILP employed formulation of logic program, mining relational concept is possible. To conbine data mining and ILP, there is a relational data mining[3].

Combine ILP with data mining from XML or HTML documents, [1] introduced logic like deduction and induction on structure of the document called "Hedge logic programs." Transformation from hedge logic program to regular logic program is also introduced. Term of hedge logic program is part of an XML document and unification and anti-unification over these terms are done in the process of deduction and induction. Since this approach is based on unification, it can be seen as one of a structural approaches.

On the other hand, [4] used ILP to learn information extraction rules from natural language data. It is quite challenging, but

This work is supported in part by 21st Century COE Program "Frame workd for Systematization and Application of Large-sclale Knowledge Resources"

[1]http://www.w3c.org/XML
[2]http://www.w3c.org/XML/Schema
[3]http://www.w3.org/TR/xpath
[4]http://www.w3.org/TR/xquery/

```
<!ELEMENT namelist (member)+>
<!ELEMENT member (name, PR,(email|URL)*)>
<!ELEMENT name (#PCDATA)>
<!ELEMENT PR (#PCDATA)>
<!ELEMENT email EMPTY>
<!ELEMENT friend EMPTY>
<!ELEMENT URL EMPTY>
<!ATTLIST URL url CDATA #REQUIRED>
<!ATTLIST member id ID #REQUIRED>
<!ATTLIST email add CDATA #REQUIRED>
<!ATTLIST member friend IDREFS #IMPLIED>

<?xml version="1.0"?>
<!DOCTYPE namelist
          SYSTEM "simplenamelist.dtd">
<namelist>
  <member id="tanaka" friend="suenaga">
    <name>Tanaka</name>
    <PR>I have no page, and no web site!</PR>
  </member>
  <member id="suenaga" friend="tanaka ohta">
    <name>suenaga</name>
    <PR>I have web page and e-mail</PR>
    <URL url="www.foo.ne.jp/~hehe/" />
    <email add="suenaga@foo.ne.jp" />
  </member>
  ....
```

Fig. 1. Example of Simple DTD and XML doucment.

markup is needed and ontology must be supply to the system and ontology building and markup is done by human.

The contribution of this article is to propose general-purpose, domain independent method for mining concept from XML documents. To achieve this, ILP system *Quantified Decision Tree*[5] and dynamic background knowledge generation from XML schema language is introduced. Since ILP employs logical formlation, proposed method can be used for general-purpose: one can give any predicate of concept such as information retrieval, searching data or categorization to be learned. To avoid domain dependency, dynamic background generation is used.

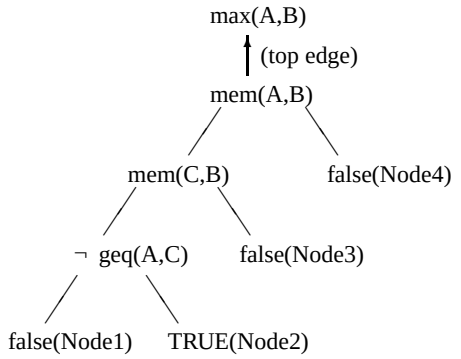
This article is organized as follow. Section 2, QDT-learner employed in experiment and its data format Quantified Decision Tree is introduced. In section 3, background knowledge generation from DTD used by QDT is introduced. Section 4 shows fusible relation in XML document not being learnable by existing method. Summary and future works is given in section 5.

2. QUANTIFIED DESISION TREE

A *Quantified Decision Tree*(short for QDT) is a first order binary decision tree with its test nodes are a **first order positive and negative literal** and variables in these test literals are existentially quantified. A QDT for n-array predicate $target(X_1, \ldots, X_n)$ [5] is a binary tree attached with a most general atom [6] of the predicate $target$ which represents clause head. The binary tree, which represents a clause body, is connected as shown in Fig.2 by the top arrow to clause head.

[5]Respecting the tradition of Prolog, string starts with capital letters are variable.

[6]For n-array predicate $target$, a most general atom is a form $target(X_1, \ldots, X_n)$ and each X_i is a distinct variable.



First order clause of each larf node.

Node1: $\exists C\,(mem(A,B) \wedge mem(C,B) \wedge \neg geq(A,C))$

Node2: $\exists C\,(mem(A,B) \wedge mem(C,B)) \wedge \forall C\,(mem(A,B) \wedge mem(C,B) \rightarrow geq(A,C))$

Node3: $\exists C\,(mem(A,B)) \wedge \forall C\,(mem(A,B) \rightarrow \neg mem(A,B))$

Node4: $\neg mem(A,B)$

Meaning of predicates.

max(A,B)	A is largest element of list B.
geq(A,B)	Same as $A \geq B$
mem(A,B)	Argument A is element of list B.

Fig. 2. Example of Quantified Decision Tree for binary predicate max

As usual decision tree, a positive and negative instance of variables in clause head predicate, which is in the form of tuple, is distributed in a manner as if the term in tuple satisfies (does not satisfy) test literal, the tuple is distributed in left(right) subtree and this process continued until tuple reaches a leaf node. In the process of distributing a tuple T via a binary tree, there may be new variables in test node literal and new tuple T' yields to keep track with such new variables. This new tuple T' is defined here as *extended tuple* and the tuple T is defined as *base tuple.* If at least one extended tuple T' from base tuple T satisfy the test node, the base tuple is distributed in left subtree and if no extended tuple satisfies the test node, then the base tuple is distributed in right subtree. Fig.2 shows an example of QDT representation of binary predicate max.

Each leaf can be compiled to sentence of first order logic. In a case of base tuple reached to "Node 1" in Fig.2, instances of variable A and B in the tuple satisfy the predicate $mem(A,B)$ and there are some C satisfying $mem(C,B)$, and, at last, there are extended tuple(s) that satisfy $\neg geq(A,C)$. So, the first order description of Node1 is like in Fig:1. In a case of base tuple reached to "Node2", it is same as "Node1" until test node $mem(C,B)$ but there are no extended tuple that satisfy $\neg geq(A,C)$. So, there are extended tuples satisfy $mem(A,B) \wedge mem(C,B)$, and if extended tuples satisfy $mem(A,B) \wedge mem(C,B)$, then all of these extended tuples **do not** satisfy $\neg geq(A,C)$. First order sentence for every node is also shown in Fig.2.

The QDT-learner system, employed here, is inductive logic programming(ILP) system which learns a concept in the form of

QDT. Since QDT formulation can represent concept with $\forall$ quantifier, QDT-learner can learn the concept with positive and negative. Also, both $\exists$ and $\forall$ quantifier are appropriately treated (not negation-as-failure of negative literal as in Prolog). In addition, output of QDT-learner can be compiled to prolog program by *First order compiler*. In general, computation of $\forall$ will be infinite, but the domain(universe) of argument being quantified by $\forall$ is finite, computation will be possible. Saying intuitively, if the prolog execution about the premise in clause body is finite, compiled prolog program is executable.

The current version of QDT-learner search space of quantified decision tree in the manner of *Simulated Annealing*(SA) to avoid local minimum and criteria to be miminize is based on MDLP (Minimum Description Length Principle). Since the system uses SA for its top level algorithm, a concept is learnable which must use non-discriminating (or functional) literal to describe(e.g. `max/2` in Fig.2).

To avoid redundant computation, sort information and input/output information about background knowledge is also have to be given to the system. QDT of Fig:2 is also simple sample of the system outputs.

3. BACKGROUND GENERATION FROM XML SCHEMA LANGUAGE

QDT-learner must be given background knowledge and instance of concept to be learn. To provide background knowledge to QDT-learner, dynamic background knowledge generation from XML schema language, DTD or XML Schema, is done. By this dynamic background knowledge generation, prolog program parsing the input XML document validated by the schema language is generated. The benefit of generating background knowledge from schema language is

- An instance of XML schema language can be seen as definition of structure of data stored in an XML document: by parsing the XML schema instance, meaning of attribute of each tag is clear (e.g. ID, IDREF or IDREFS in a DTD) and subelement model of each tag is clear.
- In the case of an XML Schema, type can be attached to each subelement of an XML document tag and type specific predicate can be used.
- Sort information of dynamically generated predicate in background knowledge is available (e.g. If a tag element has no attribute value, no need to execute predicate about attribute value for this tag element).

Dynamic background knowledge generation is done by parsing XML schema language and instantiate meta rule from declaration in XML schema language. In this process, sort and input/output information for input to QDT-learner is also generated. To simplify, case of a DTD is explained here. The DTD and XML document instance in Fig.1 is used.

Clause head of Meta rule is imperfect atom which contains variables in predicate name(not in argument) and form is like follow.

```
X_child_Y_is(PNode,ChildNode).
```

`X` and `Y` are instantiated according to declaration in schema.

In declaration of DTD in Fig.1, the element `member` has subelement `name`.So, in the process of BK generation, X is instantiated to `member` and Y is to `name` and clause head

```
  %--- Predicates for structure of document
get_common_ancestor(CNode1,CNode2,PNode).
namelist_child_member_is(PNode,ChildNode).
member_child_name_is(PNode,ChildNode).
member_child_PR_is(PNode,ChildNode).
member_child_email_is(PNode,ChildNode).
member_child_URL_is(PNode,ChildNode).
  %--- Predicates for string processing
get_pcdata(PNode,PCData).
is_substring(OuterString,InnerString).
has_common_substring(Str1,Str2).
  %--- Predicates for attribute
get_idrefs_friend_from_member(Node,RefedNode).
get_element_URL_attr_url(Node,Val).
get_element_email_attr_add(Node,Val).
```

Fig. 3. Heads of Predicate generated from DTD in 1

```
member_child_name_is(PNode, ChildNode)
```

is generated. Clause body of meta rule is prolog program that calls Java method to parse XML document. The string `member` and `name` is given to the argument of Java program to specify of name of tag element to do with.

In the case of DTD, meta rules are categorized as follow

- meta rule for element and its subelement.
- meta rule for element and its attribute.
- meta rule for ID, IDREF and IDREFS.

If subelement `PCDATA` or attribute `CDATA` (which indicate subelement or value of attribute is charactor string) is declareed, then program `is_substring/2` and `has_common_substring/2` is also added to background knowldge to provide basic character processing. Fig.3 shows example of the background knowldge generated from DTD in Fig.1.

Term to be bound in an argument is unique constant identifire to the tag element (not like in [6] in which a functional term is possible). An identifire of tag element is as follows

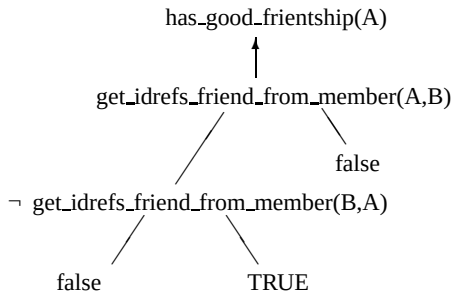
```
"File Name"__"Tag Name"__"No. of occurrence"
```

Since an XML document is finite and learning generative models are out of scope, number of tag elements are finite.

4. EXAMPLE OF SYSTEM OUTPUT

The example shown here is artifical but feasible one being not learnable by structural aproach and also not learnable by other ILP system since the concept must use $\forall$ quantifier to be described. The background knowldge is generated from DTD in Fig.1 and the input XML document is also in Fig.1. Generated background knowldge is also shown in Fig.3.

The targe concept `has_good_friendship/1` in the namelist XML document of Fig.1 is intuitively that member given in argument has good friendship if the member regard other member(s) as friend, then the other member(s) always regard member in argument as friend. A member regarded as friend is given in `member` attribute `friend`. This attribute is declared as `IDREFS` type. So, if a tag element satisfy `has_good_friendship/1`, the element has attribute of `IDREFS` and all element identified



First order clause of each larf node.

$$has\_good\_friendship(A) \leftarrow$$
$$\exists B\, get\_idrefs\_friend\_from\_member(A,B) \wedge$$
$$\forall B \left(get\_idrefs\_friend\_from\_member(A,B) \rightarrow get\_idrefs\_friend\_from\_member(B,A) \right)$$

Fig. 4. Learned QDT from the name list XML doucment and its equivalent first order sentense.

by the `IDREFS` has also `IDREFS` that spesify the element satisfy `has_good_friendship/1`.

Fig.4 shows the output of the QDT-learner system and the output QDT naturally describe this concept.

5. SUMMARY AND FUTURE WORKS

The method proposed here is general-purpose and domain independen relational data mining method for valid XML documents. To achieve domain independency, background knowledge is generated from a XML schema language by instanciating pre-defined meta-rule. In example given here, background knowldge generation from a DTD is done. To learn concept in example, $\forall$ quantifire must be apropriately treated and system output of example is shown.

But XML documents has no necessity to be validated with the schema, and using a XML also is without schema for validation. There are resarch to generate a DTD from XML doucment[7], combine DTD generation and QDT-learner with dynamic background generation will be interesting aproach to tackle with data mining from XML document with no schema.

XML Schema is also schema language to validate an XML document. An XML Schema can make more minute description than DTD, so it is expected that generated background knowldge will be minute and also apropriate for the interested XML document. Dynamic background generation for XML Schema is in progress.

6. REFERENCES

[1] F. De Francesca, G. Gordano, R. Ortale, and A. Tagarelli, "Distance-based clustering of xml documents," September 2003, pp. 75–78, ECML/PKDD'03 workshop proceedings.

[2] R. Ortale A. Tagarelli G. Costa, G. Manco, "A Tree-based Approach to Clustering XML Documents by Structure," *RT-ICAR-CS-04-04*, aprile 2004.

[3] Saso Dzeroski and Nada Lavrac, Eds., *Relational Data Mining*, Springer, 2001.

[4] James Stuart Aitken, "Learning information extraction rules: An inductive logic programming approach.," 2002, pp. 355–359, Proceedings 15th European Conference on Artificial Intelligence.

[5] T.Sato, "Program extraction from quanti ed decision trees(Extended abstract)," *Proc. of Machine Intelligence, Bury St Edmunds.*, vol. 17, pp. 78–80, 2000.

[6] Akira Ishino Akihiro Yamamoto, Kimihito Ito and Hiroki Arimura, "Modelling Semi-structured Documents with Hedges for Deduction and Induction," *Proc. of Inductive Logic Programming : 11th International Conference, ILP 2001, Strasbourg, France*, pp. 490–498, 2001.

[7] Minos Garofalakis, Aristides Gionis, Rajeev Rastogi, S. Seshadri, and Kyuseok Shim, "Xtract: a system for extracting document type descriptors from xml documents," in *SIGMOD '00: Proceedings of the 2000 ACM SIGMOD international conference on Management of data*. 2000, pp. 165–176, ACM Press.