Framework for XML Document Classification

N.J.Divya Sree[#], P.Kavya Lekha[*], M.Divya[#]

[#]*Dept of CSE, K.L. University*
Vaddeswaram,Guntur,A.P.India
[1]divya.nuti@gmail.com
[3]divyam.1293@gmail.com

[*] *Dept of CSE, K.L. University*
Vaddeswaram,Guntur,A.P.India
[2]kavyalekha25@gmail.com

Abstract- For a wide variety of information systems XML has turn into an Universal format. As huge amounts of XML documents present on the web, classification is an important and immense task. XML documents have both structures and contents for a semi-structured data. Thus, text learning techniques are not very suitable for XML document classification as structures are not considered. This paper presents a novel complete framework for XML document classification. We first present a knowledge representation method for XML documents which is based on a typed higher order logic formalism. With this representation method, an XML document is represented as a higher order logic term where both its contents and structures are captured. We then present a decision-tree learning algorithm driven by precision/recall breakeven point (PRDT) for the XML classification problem which can produce comprehensible theories. Finally, a semi-supervised learning algorithm is given which is based on the PRDT algorithm and the co-training framework. Experimental results demonstrate that our framework is able to achieve good performance in both supervised and semi-supervised learning with the bonus of producing comprehensible learning theories.

***Keywords*—XML document, machine learning, knowledge representation, semi-supervised learning.**

I. INTRODUCTION

XML, a simple, powerful, and extensible data format that can be used to represent hierarchical data, has become the standard for data representation and exchange on the web.

First, Knowledge representation (KR) is usually the first step toward intelligent reasoning. Generally speaking, KR involves using some formal symbols to represent the knowledge around us. Second, in many cases comprehensible learning theories are traded-off for high accurate but incomprehensible theories. The advantage of comprehensibility of the output learning theory has been recognized by machine learning researchers. Third, learning with minimal labeled training examples is desirable for web intelligence. Supervised learning often requires a large number of labeled examples in order to learn accurately.

The contributions of this paper can be summarized as follows:

- A knowledge representation approach for XML documents based on a typed higher order logic formalism that is suitable for representing structured data.
- A decision-tree learning algorithm driven by the precision/recall (PRDT) heuristic for XML document classification.
- A semi-supervised learning algorithm for XML document classification which combines the Co-training framework and the PRDT algorithm.

II. RELATED WORK

The related work is reviewed from three aspects: knowledge representation of XML documents, learning with XML documents, and semi-supervised learning.

A. *Representation of Individuals:*

Individuals refer to the object of a learning problem. For example, XML documents are the individuals in the context of the learning problem of this paper. The formal basis for the representation of individuals is provided by the concept of a basic term.

The basic principle is that an individual should be represented by a closed term. This formalism provides a rich catalogue of data types with which to model individuals. The types appearing in this paper and their symbols are listed as follows: Base types include integers (Int), natural numbers (Nat), floats (Float), characters (Char), strings (String), and booleans (Ω). Other types that will appear in this paper include Tuples, Sets, and Lists. A tuple type is denoted by $\alpha_1 \times \cdots \times \alpha_n$, and a term of this type is denoted by $(t_1, \ldots, t_n)$ where t1; . . . ; tn are type $\alpha_1, \cdots, \alpha_n$, respectively. A list type is denoted by List α or [α] , and a term of this type is denoted by [t1; . . . , tn] where t1, . . ., tn is of type α. A set type is denoted by $\{\alpha\}$, and a term of this type is denoted by {t1, . . . , tn} where t1, . . . , tn are type α.

B. *Representation of Features*

Features refer to boolean predicates on the type of individuals. All the candidate features form the feature space of the learning problem. The learning algorithms intend to find the target function in the feature space. We outline how this higher order logic formalism supports expressing and constructing features. Simply put, predicates are built up incrementally by composition of simpler functions called transformations.

A transformation f is a function having a signature of the form $f : (\varrho_1 \to \Omega) \to \cdots \to (\varrho_k \to \Omega) \to \mu \to \sigma,$ wh ere any parameters (type variables) in $\varrho_1, \cdots, \varrho_k$ and σ appear in μ $k > 0,$, and Ω is the type of the booleans. The type μ is distinguished and is called the source of the transformation, while the type σ is called the target of the transformation. The number k is called the rank of the transformation. The intuitive idea behind the definition of transformation is that, given predicates $p_i : \varrho_i \to \Omega,$ for $i = 1, \ldots, k,\ f\, p_1 \ldots p_k$ is a function that takes individuals of type μ to individuals of type σ. By composing several such functions, the last of which is a transformation with target type boolean, a predicate on individuals of the desired type is obtained.

III. KNOWLEDGE REPRESENTATION FOR XML DOCUMENT CLASSIFICATION

In this section, we describe how to represent XML documents using the higher order logic formalism. XML documents are a typical type of semistructured data.1 semi structured data are data that have some structures but these structures are not fixed. XML documents have nesting structures which could be complex. Structures add more information to the contents of an XML document. The nesting structures of an XML document provide a way of information aggression and correlation. Both structures and contents play important roles in XML document classification.

A. *Representation of XML Document*

Individuals Elements are the basic building blocks of an XML document. An XML document must contain at least one element. Each element is enclosed by a start tag and an end tag. Tags of elements usually indicate the semantic meaning of the structure. The content of an element could be another element or plain text. An XML document can be validated using a Document Type Definition (DTD) or an XML schema. DTDs and XML schemas basically fulfill the same purpose, i.e., to serve as the grammar of a set of XML documents. However, they have very different syntax. We focus on the XML document representation with DTDs in this paper, as the XML schema is itself an XML document while the DTD has its own syntax. Moreover, an XML schema may contain an DTD.

a) *Structure Representation*

Formally, an XML document is represented as a six-tuple term.

type XML = XMLDecl *Misclist * DTD * Misclist * Element *Misclist

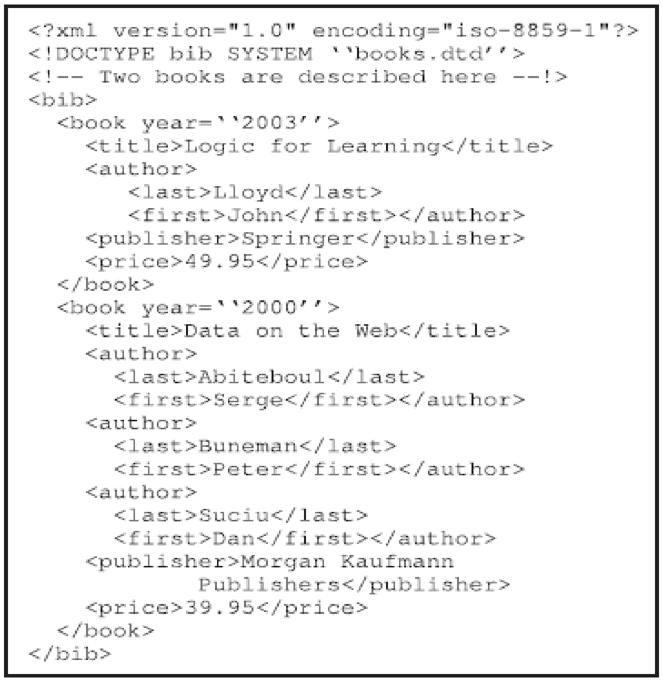
Here, XMLDecl represents the XML declaration; Misclist represents a list of miscellaneous items such as comments, processing instructions, and spaces; DTD represents the document type declaration; and Element represents the root element of the document. Though all these six components are non atomic type values, only the definition of type Element is recursive. The formal representation of an element is given as follows:

data Element = Elem TagName [Attribute] [Content]

Here, type Element is defined via a data constructor Elem which has three arguments representing the element name, the attributes and the element contents, respectively. TagName is a synonym of type String. [Attribute] and [Content] are the list of attributes and the list of content, respectively.

An attribute is composed of the attribute name and attribute value, and is represented using a tuple type Attribute = AttName * AttV alue, where AttNameandAttV alue are both synonym of strings.

It is the element content that makes an XML document nested and hierarchical. The content of an element can be another element, a piece of text, a reference to some entities, a CDATA section, a processing instruction or a comment,string.



```
<?xml version="1.0" encoding="iso-8859-1"?>
<!DOCTYPE bib SYSTEM ``books.dtd''>
<!-- Two books are described here --!>
<bib>
  <book year=``2003''>
    <title>Logic for Learning</title>
    <author>
       <last>Lloyd</last>
       <first>John</first></author>
    <publisher>Springer</publisher>
    <price>49.95</price>
  </book>
  <book year=``2000''>
    <title>Data on the Web</title>
    <author>
      <last>Abiteboul</last>
      <first>Serge</first></author>
    <author>
      <last>Buneman</last>
      <first>Peter</first></author>
    <author>
      <last>Suciu</last>
      <first>Dan</first></author>
    <publisher>Morgan Kaufmann
            Publishers</publisher>
    <price>39.95</price>
  </book>
</bib>
```

Fig. 1 .An example XML document

An element content is formally represented using data constructors.

data Content = El Element | Tx CharData | Ref Reference | . . .

Here, El, Tx, and so on, are data constructors of type Content. Data constructor El needs an argument of type Element to construct data of type Content, and Tx needs a type CharData to construct data of a type Content, and so on.

b) Content Representation

The text content in an XML document node is represented using the traditional vector-space model. We adopt the traditional text dimension reduction approach which includes stop words removal, stemming, and feature selection.

Structured feature selection is a method to conduct the feature selection procedure on the same part (element) of the document. In more detail, our method is to build n independent corpora of text by analyzing the DTD, where n is the number of elements in the DTD. Each corpus is identified by the path of the element which it corresponds to. The text features of the same element of all XML documents are collected and stored in this corpus. The feature selection will be conducted for each corpus independently.

B. Representation of Features for XML Learning

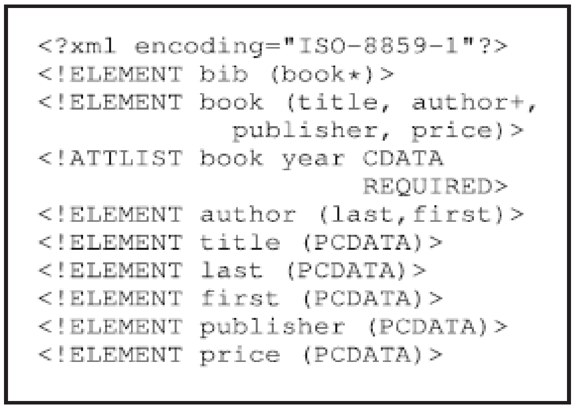
Transformations on XML are classified into two categories: generic transformations and data-specific transformations. Generic transformations come straight from the XML document representation and are applicable to all wellformed XML documents.

a) Generic Transformations

Generic transformations are determined by the individual representations and are fixed to all data sets having the same representation. We introduce some typical generic transformations for XML documents here.

The transformations corresponding to the document level of the XML document representations include projRootElement: XML→Element, which projects an XML document onto the root element.

Usually, most of the information of an XML document is stored in its root element and its nested structure. An important transformation on the element is *proj Contents : Element -> Contents*, which projects an element onto its nested contents. We use a setExists1 transformation to check whether the set contents has content satisfying a predicate.



```
<?xml encoding="ISO-8859-1"?>
<!ELEMENT bib (book*)>
<!ELEMENT book (title, author+,
            publisher, price)>
<!ATTLIST book year CDATA
                    REQUIRED>
<!ELEMENT author (last,first)>
<!ELEMENT title (PCDATA)>
<!ELEMENT last (PCDATA)>
<!ELEMENT first (PCDATA)>
<!ELEMENT publisher (PCDATA)>
<!ELEMENT price (PCDATA)>
```

Fig 2. The DTD file of the XML document in fig.1.

The transformation is setExists1 : $(Content \to \Omega) \to \{content\} \to \Omega$.. The following are conjunction transformations on type Element:

$$\wedge_3 : (Element \to \Omega) \to (Element \to \Omega) \to (Element \to \Omega) \to Element \to \Omega$$

b) Data-Specific Transformations

Data-specific transformations are associated with the specified DTD and XML documents in the application, which may include transformations on the element names, on the attribute names and values, and on features of the text content.

We can obtain a collection of useful data-specific transformations by analyzing the DTD. We use a Perl script to create data-specific transformations from a DTD by using the following rules:

- Rule1:Atransformation (=‘‘elementName’’): *TagName* →Ω is generated if a line starts with *<!ELEMENT*, where variable *elementName* is the string followings the *<!ELEMENT;*
- Rule2:Atransformation (=“attributeName’’): AttName → Ω is generated if a line starts ith *<!ATTLIST*, where variable *attributeName* is the string that comes second after the *<!ATTLIST*;
- Rule3:Ntransformations $(= val_{i}) : AttValue \to \Omega$are generated if the string following the attribute name is in the form of $(val_1|val_2| \ldots\ldots val_N)$, where i=1……,N;

c) Predicate Search Space

Predicate search space is constructed by using predicate rewrite systems. We now use an example to illustrate the predicate representation and generation on XML documents.

IV. A PRECISION/RECALL-DRIVEN DECISION TREE LEARNING ALGORITHM

This section describes a novel decision-tree learning algorithm based on the precision and recall criterion for XML document classification. To the best of our knowledge, this is the first work of using precision and recall as node splitting criteria in decision-tree algorithms.

A. The Precision/Recall Heuristic

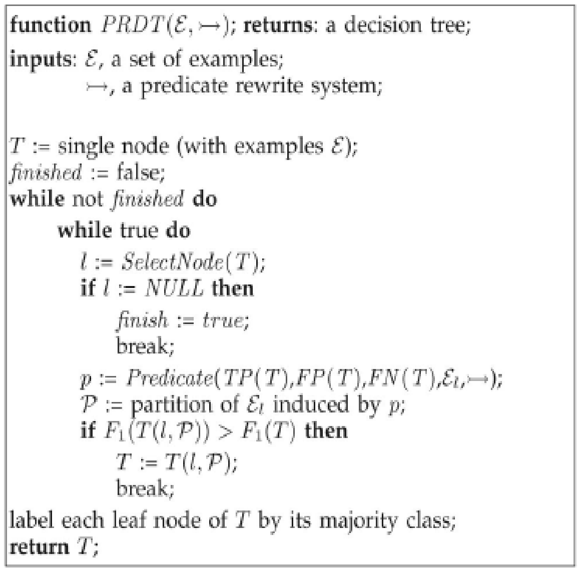
Precision and recall were originally two statistical measures widely used in information retrieval. Precision measures the “soundness” of the classifier, and recall measures the “completeness” of it.

The precision Pr and recall Rc can be defined using TP (True Positive), FP (False Positive), FN (False Negative), and TN (True Negative).

$$P_r = \frac{TP}{TP + FP} \qquad R_c = \frac{TP}{TP + FN'}$$

where TP is defined as the number of documents correctly assigned to the positive class; FP, the number of documents incorrectly assigned to the positive class; FN, the number of documents incorrectly assigned to the negative class; and TN, the number of documents correctly assigned to the negative class. When precision and recall are equal (or very close), this point is called the precision/recall-breakeven point (BEP) of the system.

Our motivation of choosing the precision and recall heuristic in XML document classification is mainly that XML documents have strong connections with text documents. Most XML documents contain text contents. Another reason is that for binary decision-trees, precision/recall is a more sensitive criteria than those single-criterion heuristics such as accuracy, especially when the number of examples in each class is very unbalanced, or the number of documents belonging to each category very small.



```
function $PRDT(\mathcal{E}, \rightarrowtail)$; returns: a decision tree;
inputs: $\mathcal{E}$, a set of examples;
        $\rightarrowtail$, a predicate rewrite system;

$T$ := single node (with examples $\mathcal{E}$);
$finished$ := false;
while not $finished$ do
    while true do
        $l := SelectNode(T)$;
        if $l := NULL$ then
            $finish := true$;
            break;
        $p := Predicate(TP(T), FP(T), FN(T), \mathcal{E}_l, \rightarrowtail)$;
        $\mathcal{P}$ := partition of $\mathcal{E}_l$ induced by $p$;
        if $F_1(T(l,\mathcal{P})) > F_1(T)$ then
            $T := T(l,\mathcal{P})$;
            break;
label each leaf node of $T$ by its majority class;
return $T$;
```

Fig. 3.Decision-tree algorithm based on the precision/recall heuristic.

B. The Precision/Recall-Driven Decision-Tree Algorithm

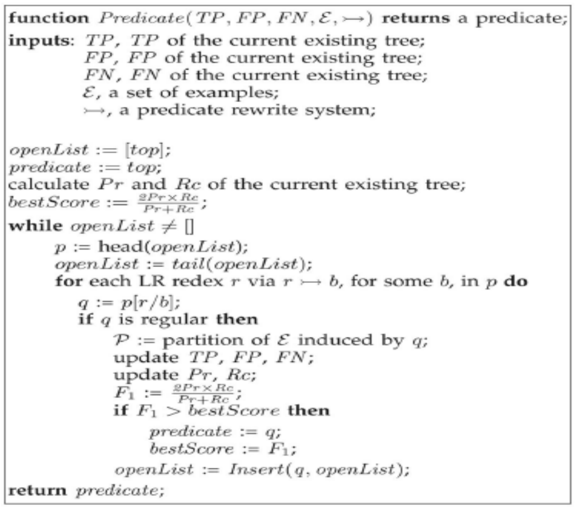
We first describe the main decision-tree algorithm, followed by a predicate selection algorithm and a node selection algorithm.

a) The Decision Tree Algorithm

The goal of this algorithm is to find a tree that can produce the best BEP value. Starting from a single node which is composed of the training data, the algorithm works toward two goals at the same time: looking for the point where the global precision recall are equal, and improving the F1 measure. The first goal is achieved by selecting the node which can most balance the precision and recall, which is done by a novel node selection algorithm. The second goal is achieved by finding a predicate to split this node which can best improve the F1 value of the tree, which is done by a predicate selection algorithm.

The PRDT algorithm requires two inputs: a set of training examples and a predicate rewrite system. The tree T is initialized as a single node containing all the training examples. The algorithm then enters an iteration which performs node selection and node splitting.

Function Predicate takes inputs of the TP, FP, and FN of the current tree T, the set of training examples in the selected leaf and the predicate rewrite system and outputs the best predicate to split ε. An openlist is used to keep all the candidate predicates. Variable predicate is used to keep the current best predicate, and bestScore is used to keep the F1 of the current best predicate.



```
function $Predicate(TP, FP, FN, \mathcal{E}, \rightarrowtail)$ returns a predicate;
inputs: $TP$, $TP$ of the current existing tree;
        $FP$, $FP$ of the current existing tree;
        $FN$, $FN$ of the current existing tree;
        $\mathcal{E}$, a set of examples;
        $\rightarrowtail$, a predicate rewrite system;

$openList := [top]$;
$predicate := top$;
calculate $Pr$ and $Rc$ of the current existing tree;
$bestScore := \frac{2Pr \times Rc}{Pr + Rc}$;
while $openList \neq []$
    $p :=$ head($openList$);
    $openList := tail(openList)$;
    for each LR redex $r$ via $r \rightarrowtail b$, for some $b$, in $p$ do
        $q := p[r/b]$;
        if $q$ is regular then
            $\mathcal{P} :=$ partition of $\mathcal{E}$ induced by $q$;
            update $TP$, $FP$, $FN$;
            update $Pr$, $Rc$;
            $F_1 := \frac{2Pr \times Rc}{Pr + Rc}$;
            if $F_1 > bestScore$ then
                $predicate := q$;
                $bestScore := F_1$;
            $openList := Insert(q, openList)$;
return $predicate$;
```

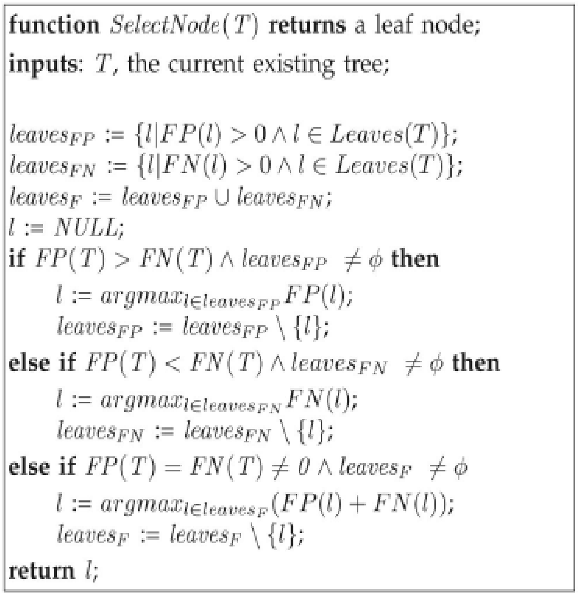
Fig 4.Algorithm for finding a predicate to spilt a node

b) The predicate selection Algorithm

Initially, the openList contains only the weakest predicate top, and the bestScore is set to be the F1 of the current tree T. The algorithm then enters an iteration. In each iteration, the first candidate predicate is drawn from the openlist and a subsearch space is generated from this predicate. Each predicate q in this subsearch space is tested by creating a new partition P using it. The F1 of the tree after adding the new partition P is computed using the updated TP, FP, and FN. If F1 is higher than the current bestScore, the best predicate and the best score are set as q and its corresponding F1, respectively. Predicate q is also inserted into the openlist as a candidate for further subsearch space. The iteration terminates when the openlist is empty. Finally the predicate that best improves the F1 of the tree by splitting the selected leaf is returned.

c) The Node Selection Algorithm

In our PRDT algorithm, we use a novel node selection method to control the system to work toward reducing the difference between the precision and recall. Next, we give some theoretical analysis for our node selection method..
First, we define the TP, FP, and FN values for the node associated with ε, where ε is a(non-empty) set of example.



```
function $SelectNode(T)$ returns a leaf node;
inputs: $T$, the current existing tree;

$leaves_{FP} := \{l | FP(l) > 0 \wedge l \in Leaves(T)\}$;
$leaves_{FN} := \{l | FN(l) > 0 \wedge l \in Leaves(T)\}$;
$leaves_{F} := leaves_{FP} \cup leaves_{FN}$;
$l := NULL$;
if $FP(T) > FN(T) \wedge leaves_{FP} \neq \phi$ then
    $l := argmax_{l \in leaves_{FP}} FP(l)$;
    $leaves_{FP} := leaves_{FP} \setminus \{l\}$;
else if $FP(T) < FN(T) \wedge leaves_{FN} \neq \phi$ then
    $l := argmax_{l \in leaves_{FN}} FN(l)$;
    $leaves_{FN} := leaves_{FN} \setminus \{l\}$;
else if $FP(T) = FN(T) \neq 0 \wedge leaves_{F} \neq \phi$
    $l := argmax_{l \in leaves_{F}} (FP(l) + FN(l))$;
    $leaves_{F} := leaves_{F} \setminus \{l\}$;
return $l$;
```

Fig 5. Algorithm for finding a predicate to spilt a node

V. CONCLUSION

Rapid growth in e-world has lead to a significant importance to the XML Documents such that automatic learning with these kinds of data by machines is predominant. This paper explores the learning issues with XML documents from three major research areas: knowledge representation, symbolic machine learning, and semi-supervised learning. In the knowledge representation part, we explored how to represent an XML document using higher order logic terms which can easily capture both the structure and the content of the data. In the symbolic machine learning part, we presented a new decision-tree learning algorithm driven by the precision/recall breakeven point for XML document classification. In the semi-supervised learning part, we explored how to use a small number of labeled data to get a good classifier with the help of a large number of unlabeled data by using the co-training algorithm and the PRDT algorithm. We conclude this paper with addressing its limitations with possible solutions.

REFERENCES

[1] J.X. Wu and J. Zhang, "Knowledge Representation and Learning for Semistructured Data," Technical Report, CSIRO ICT Centre, 2009.

[2] Bouchachia.A, Hassler.M, **"**Classification of XML Documents",2007

[3] Clark.M,Watt.S "Classifying XML Documents by Using Genre Features",2007

[4] Qingjiu Zhang, "Shiliang sun, "Evolutionary classifier ensembles for semi-supervised learning",2010

[5] Yuanyuan Guo, Xiaoda Niu ; Zhang.H "An Extensive Empirical Study on Semi-supervised Learning",2010

[6] A. McCallum and K. Nigam, "Employing EM and Pool-Based Active Learning for Text Classification," Proc. 15th Int'l Conf. Machine Learning, 1998.

[7] X. Zhu and A.B. Goldberg, "Introduction to Semi-Supervised Learning", 2009

[8] Nodes coupling in a Bayesian network for the automatic classification of XML documents, International Conference on machine and Web Intelligence (ICMWI), 2010. Ecole Nat. Super. d'Inf. ESI, Algiers, Algeria

[9] Classification Tree Embedded XML Document Structure Design for Enhanced Web Document Utilization, Sixth International Conference on Advanced Language Processing and Web Information Technology, 2007,pages: 542-547.

[10] Applications of Data Mining in the Education Resource Based on XML, International Conference on advanced Computer Theory and Engineering, 2008, ICACTE'08. Pages: 943-946.

[11] Graph-based Semi-supervised Learning Algorithm for Web Page Classification, Sixth International Conference on Intelligent Systems Design and Applications,2006. Pages:856-860.

[12] Research on Multi-View Semi-Supervised Learning Algorithm Based on Co-Learning, International Conference on Machine Learning and Cybernatics, 2006. Xing-Qi wang

[13] A Passive-Aggressive Algorithm for Semi-supervised Learning, International Conference on Technologies and Applications of Artificial Intelligence, 2010. Chien-chung Chang. Pages: 335-341.

[14] A new semi-supervised support vector machine learning algorithm based on active learning , International Conference on Future Computer and Communication (ICFCC), 2010. Li Cunhe. Vol 3, May 2010.