M. Engin TOZAL

M.S. Thesis In Computer Engineering

September - 2005

CLASSIFICATION USING XSLT

by

M. Engin TOZAL

September 2005

CLASSIFICATION USING XSLT

by

M. Engin TOZAL

A thesis submitted to

the Graduate Institute of Sciences and Engineering

of

Fatih University

in partial fulfillment of the requirements for the degree of

Master of Science

in

Computer Engineering

September 2005
Istanbul, Turkey

I certify that this thesis satisfies all the requirements as a thesis for the degree of Master of Science.

Prof. Dr. Kemal FİDANBOYLU
Head of Department

This is to certify that I have read this thesis and that in my opinion it is fully adequate, in scope and quality, as a thesis for the degree of Master of Science.

Assist. Prof. Dr. Atakan KURT
Supervisor

Examining Committee Members

Assist. Prof. Dr. Atakan KURT

Assist. Prof. Dr. Erkan İMAL

Assist. Prof. Dr. Nahit EMANET

It is approved that this thesis has been written in compliance with the formatting rules laid down by the Graduate Institute of Sciences and Engineering.

Assist. Prof. Dr. Nurullah ARSLAN
Director

Date
September 2005

CLASSIFICATION USING XSLT

M. Engin TOZAL

M. S. Thesis - Computer Engineering
September 2005

Supervisor: Assist. Prof. Atakan KURT

ABSTRACT

XSLT classification is a hybrid technique that takes advantages of both web page (HTML) and semi-structured document (XML) classification. Although a number of organizations are working on standardizing XML markup for specific domains on behalf of electronic data interchange, XML doesn't force any predefined vocabulary like HTML. Nevertheless, XML markups generated by different sources for a specific domain usually have similarities in terms used as tag or attribute names and in structure used to represent content. Proposed XSLT classification is based on component tagging; each occurring word is prefixed with elements in ancestor hierarchy. Additionally, each ancestor element is prefixed with its ancestors as well. Furthermore; terms exist in HTML markups like *meta, title, anchor, img* and literal strings exist in HTML but not in XML are included into term frequency vector that represents document in classification process. Two different experiments are run over the dataset. The first experiment compares different component tagging models that represent XML documents. The models that are based on loose ancestor hierarchy are better than the ones based on strict ancestor hierarchy. The second experiment compares HTML, XML, and XSLT classification techniques. XSLT classification gives higher accuracy rates than XML and HTML.

Keywords: XML, XSLT, Semi-Structured Document, Classification, Component Tagging

XSLT ile SINIFLANDIRMA

M. Engin TOZAL

Yüksek Lisan Tezi – Bilgisayar Mühendisliği
Eylül 2005

Tez Yöneticisi: Yrd. Doç. Dr. Atakan KURT

ÖZ

XSLT sınıflandırma; yarı-yapılandırılmış (XML) ve web dökümanı (HTML) sınıflandırma tekniklerinin avantajlı yönlerini birleştiren melez bir sınıflandırma yöntemidir. Birçok organizasyonun belirli alanlara yönelik biçimlendirme dili oluşturma çalışmalarına rağmen XML biçimlendirme dili HTML biçimlendirme dili gibi herhangi bir standart sözlüğe bağlı değildir. Yinede farklı kaynaklar tarafından hazırlanan XML dökümanlarında, içeriği biçimlendirmek için kullanılan eleman yada özellik isimleri aynı alanlar içerisinde benzerlik göstermektedir. Bu çalışmada sunulan XSLT sınıflandırma tekniği, bileşen etiketlemek –içerikte geçen her kelimenin kendisini çevreleyen eleman/özellik etiketleriyle öneklendirilmesi- üzerine bina edilmiştir. Ayrıca her her ata-eleman kendisinin ata elemanlarıyla öneklendirilmiştir. HTML dökümanında geçen fakat XML dökümanında geçmeyen *meta, title, anchor, img* gibi etiketlerin içeriği ve yalın söz dizimleri sınıflandırma sürecinde terim sıklık dizisine eklenir. İki farklı deney veri kümesi üzerinde çalıştırılmıştır. Birinci deneyde birbirinden farklı XML bileşen etiketleme yöntemleri karşılaştırılmıştır. Deneyde, gevşek-ata-hiyerarşisine dayanan modellerin, sıkı-ata-hiyerarşisine dayanan modellerden daha iyi olduğu ispatlanmıştır. İkinci deneyde, HTML, XML, ve XSLT sınıflandırma teknikleri karşılaştırılmış ve XSLT yönteminin diğerlerinden daha iyi doğruluk oranı verdiği görülmüştür.

Anahtar Kelimeler: XML, XSLT, Yarı-Yapılandırılmış döküman, Sınıflandırma, Bilşen Etiketleme

ACKNOWLEDGEMENT

I would like to express my gratitude to my supervisor Assist. Prof. Dr. Atakan KURT for his immense help in planning and executing the works and insight throughout the research.

I am very grateful to Assist. Prof. Dr. Nahit EMANET, Assist. Prof. Dr. Erkan İMAL and Instructor Zeynep ORHAN for their valuable suggestions and comments.

TABLE OF CONTENTS

LIST OF FIGURES

CHAPTER 1

INTRODUCTION

The World Wide Web has already become the most important medium for publishing information, managing business, and communicating with people around the world. As the web rapidly gets richer and richer by the addition of new information it becomes more difficult to find the information that is demanded by people. Many automated tools such as crawlers, search engines and directories help people to search for data. Nevertheless analyzing, classifying, clustering and extracting useful information and rules through billions of web documents –HTML pages- based on their content, structure and usage is an active and challenging research problem in *data mining* called *web mining*. Web Mining consists of three different problem fields *web content mining* [3] studies extraction of useful information from web page contents like searching, retrieving and ranking of web pages, building web page categories. *Web structure mining* [1, 2] tries to derive information from the structure of hyperlinks; discovering authoritative (pages that are mostly linked from related sites) and hub (pages that has many links to related sites) pages, *web usage mining* [4] refers to discovery of user access patterns from web server logs.

On one hand, the World Wide Web is widening in its size and complexity on the other hand, recently invented technologies bring new challenges in the area of web mining. XML (Extensible Markup Language) and XSLT (Extensible Stylesheet Language Transformation) are among the important technologies that drive attention in www information publishing, so in web mining.

Although both XML and HTML derived from SGML, HTML describes the look and feel, whereas XML contains only content and its structure. So the same XML content can easily be incorporated into different visual presentations and presentation management can be done at a single point. XSLT technology is used to transform XML data into different formats like XHTML, Text, and WML. In the World Wide Web, XSLT is used to transform XML data into a presentable format that is suitable for client platform like XHTML or WML.

As the web exploits XML and XSL technologies, it becomes important to incorporate those technologies into web mining. A number of studies have been done for classification of semi-structured documents, and HTML classification draw attentions of people since it began to appear, however XSLT classification is not considered in web mining literature to the best of our knowledge.

Web classification techniques utilize presentation markup (HTML) and analyze link structure. Text only classification technique removes all HTML markup and classifies the web pages with respect to pure content. Hypertext classification technique considers the presentation markup to weight the importance of textual content. Link analysis technique examines pages that have direct links from the page and pages that have links to the page to classify a web page.

Markup vocabulary and structure of semi-structured documents hides valuable information about the nature of the content. A component is a fragment of semi-structured document which contains either structured or textual data. Semi-Structured Document classification models that are based on component tagging, do not consider the terms appear in the textual content as features by themselves but tags the terms with their surrounding markup structure to represent them as features. Models that are based on component splitting handle each component in the document independently and train over each of them.

XSLT classification is a hybrid classification technique that exploits both structural markup of XML document and presentational markup features embedded into the result XHTML document. Both specific markup vocabulary and structure flows from XML document and fixed presentation markup of HTML are utilized to generate feature set of classification. XML document itself contains metadata –data fragments that are not published to the end user- and those data are used in Semi-Structured Document classification. But XSLT classification utilizes XML fragments that are only referenced in the stylesheet. The published information in HTML document may be originated from multiple XML documents, XSLT classification considers them as a single document where Semi-Structured document classification handles each of them standalone. Some of the published data do not come from XML document but directly embedded into HTML tags. XSLT classification puts to good use such data as well.

This study discusses different data mining methods used to classify semi-structured documents and provides a framework which exploits the information embedded into XSLT stylesheets. Section 2 gives the background about XML, XSLT, text document classification, web page classification, web site classification and semi-structured document classification. Section 3 discusses the new framework proposed, and different methods used in semi-structured document classification. Section 4 discusses the experiments run over a set of documents with web page classification, semi-structured document classification and the new XSLT technique that exploits both XML and XSLT. Section 5 discusses future work and concludes the study.

CHAPTER 2

BACKGROUND

A document classifier simply maps a document to a predefined category (class) or estimates the probability that a given document belonging to one of those categories.

Given a document database $D=\{d_1,d_2,\ldots,d_n\}$ and a set of categories $C=\{c1,c2,\ldots,cm\}$ the classification problem is to define a mapping $f : D \rightarrow C$ where each d_i is assigned to one category. A category contains those documents mapped to it; that is $c_j = \{d_i \mid f(d_i) = c_j, 1 \leq i \leq n$ and $d_i \in D, cj \in C\}$.

Classification usually requires two step processing. First, a specific model is built by evaluating a set of training documents for each class which is called *learning phase*. Next, for a given document the classifier maps it to a class category based on the derived model. This phase is called *classification*. A *preprocessing phase* may be applied to fix missing items, inconsistent values, or noisy and irrelevant arguments and to integrate or transform data before learning and classification.

There are a number of text classification techniques, Naïve Bayes [5, 7, 8], Decision Trees [14], K-Nearest Neighbor [13], Neural Networks [15], Support Vector Machines [16]. Despite the Naïve Bayes is one of the simplest techniques; it is a common technique due to its clear and plain probability basis, strong output and efficiency.

2.1 XML and XSLT

Standard Generalized Markup Language (SGML) became an international standard for defining structure and content of different electronic documents in 1986. Its popularity increased rapidly in many industries for creating, managing, and distributing electronic documents. However it did not accepted in World Wide Web due to lacks such as; support of stylesheets, complex and unstable SGML software because of its broad range and powerful options.

Extensible Markup Language (XML) which is a subset of SGML was developed by XML Working Group under auspices of World Wide Web Consortium (W3C) in 1996. It is not designed to replace SGML, but removes complex features of SGML and retains beneficial aspects of SGML. XML is a public format and it is not a proprietary format of any company. Yet, big market players such as Microsoft, Sun, Adobe, and Netscape support the development of XML standard.

A markup language is a mechanism to identify structures in a document. XML specification defines a standard and flexible way to add markup to documents. Almost all recent browsers support use of XML over the Internet. There exists a lot of software to process XML documents. Elements, Attributes, Entity References, Comments, Processing Instructions, and CDATA Sections are building blocks of XML documents. Elements used to identify nature of the content that they surround. They are the most common form of XML markup. Elements are delimited by angle brackets and if an element is not empty it begins with a start-tag <*element*> and ends with an end-tag </*element*>. Empty elements have a modified syntax as <*element/*>. Attributes are used to assign properties to elements. They are name-value pairs occur inside start-tag after element name <*element attribute="value"*>. All attribute values must be quoted. Characters such as <*, >, &," , '* are reserved in XML. Entity references are used to represent these special characters appears in content, refer often repeated or varying text, and content from external documents. Comments begin with <*!*—and ends with --> character sequences. It can contain any literal string except -- character sequence. Processing instructions are used to provide information to particular applications. They are not textually part of document but the XML processor is required to pass them to applications. Processing instructions have the syntax *<?pitarget pidata?>*, pitarget identifies the application to whom the pidata should be passed. CDATA Sections inform the parser to pass the content to application without interpretation. CDATA sections start with *<![CDATA[* and ends with *]]>* any characters are allowed in CDATA sections except *]]>* which is perceived as end of section.

XML is a case sensitive language. It starts with the XML declaration and should have one root tag that surrounds all content of document. All XML elements must have a closing tag and the elements should be nested properly. Although it is not required, an XML document may conform to a DTD, which defines the rules of document structure, element and attribute names, entity references.

Apart from its logical structure, XML documents have a physical hierarchical structure, and should have only one top root element. Figure 2.1 presents a simple resume document in XML format that consists of markup and content

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE resume SYSTEM "sample.dtd">
<!-- Sample Resume for illustration purpose -->
<resume id="10050808" language="EN" category="Software Architect">
    <type>Functional</type>
    <create-date>March,13,2005</create-date>
    <modification-date>July,9,2005</modification-date>
    <copyright>All content contained within this document is protected
         by copyright laws &#x00A9; Acme Resume Corp. 2005</copyright>
    <personal name="Engin" last-name="Tozal">
        <objective>To hold a Ph.D. in Distributed Systems area and become
        an expert in Distributed Systems issues as a
researcher</objective>
```

```
        <education start="1997" end="1999">
            <collage>Fatih University</collage>
            <department>Mathematics</department>
        </education>
        <education start="1999" end="2003">
            <collage>Fatih University</collage>
            <department>Computer Engineering</department>
        </education>
        <language>Turkish</language>
        <language>English</language>
    </personal>
    <experience>
        <project>
            <name>High-Level Data Link Control Simulation</name>
            <technology>Berkeley  Sockets, C</technology>
        </project>
        <project>
            <name>Visual XPath</name>
            <technology>Java, XML, XPath, DOM, SAX, Xerces, Xalan,
XSLT</technology>
        </project>
    </experience>
</resume>
```

Figure 2.1 Sample XML Document

XML provides simplicity because it contains self-describing data and can easily be processed by machines and understood by human beings. XML is extensible because developers can generate their own elements and element rules. On the other hand, HTML comes with a set of predefined vocabulary. XML provides interoperability because many platforms and tools use XML and parsers that interpret them are already available free of charge.

Web applications use XML in two ways; storing the information with its structure in xml files is one way, but this scheme is not suitable for large and complex web applications that maintains large amounts of data. Relational Databases are used to store large volumes of relationally structured data, however the tendency is retrieving the data in XML format and pass it to the application layer. All dominant RDBMS's like Oracle, MS-SQL, DB2 provide XML information retrieval interface, and moreover there exists Native XML databases like, Tamino, Lore, Xyleme which is designed to handle semi-structured documents.

World Wide Web uses HTML as information publishing technology, but web applications are getting more complex due to interoperability problems among web browsers, diverse client devices such as mobile phones, PDAs, TV sets and other kind of desktop applications that support different Graphical User Interfaces, presentation of same content in different languages. All of these problems can be solved by separating content data from presentation markup.

XSLT[1] (XSL Transformations) is part of XSL (Extensible Stylesheet Language) technology that defines a set of rules used to transform XML documents into different formats. XML captures only the content and the structure. However World Wide Web consists of HTML pages that define how the content is presented to the user, so XSLT is used as a bridging technology to transform XML documents into HTML or more commonly XHTML format, as shown in Figure 2.2.

[1] http://www.w3.org/TR/xslt

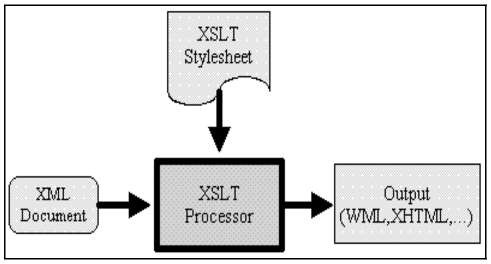
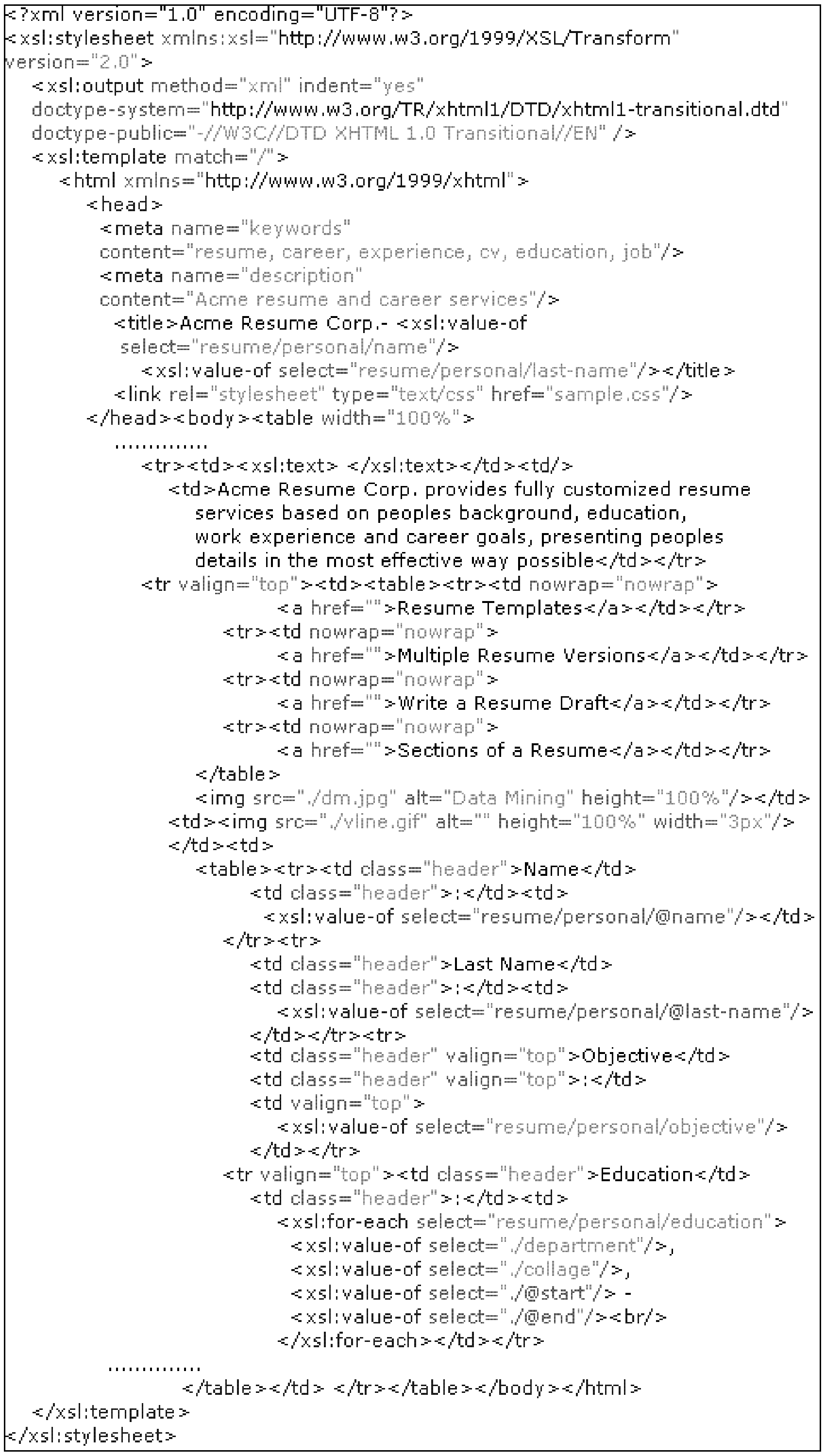


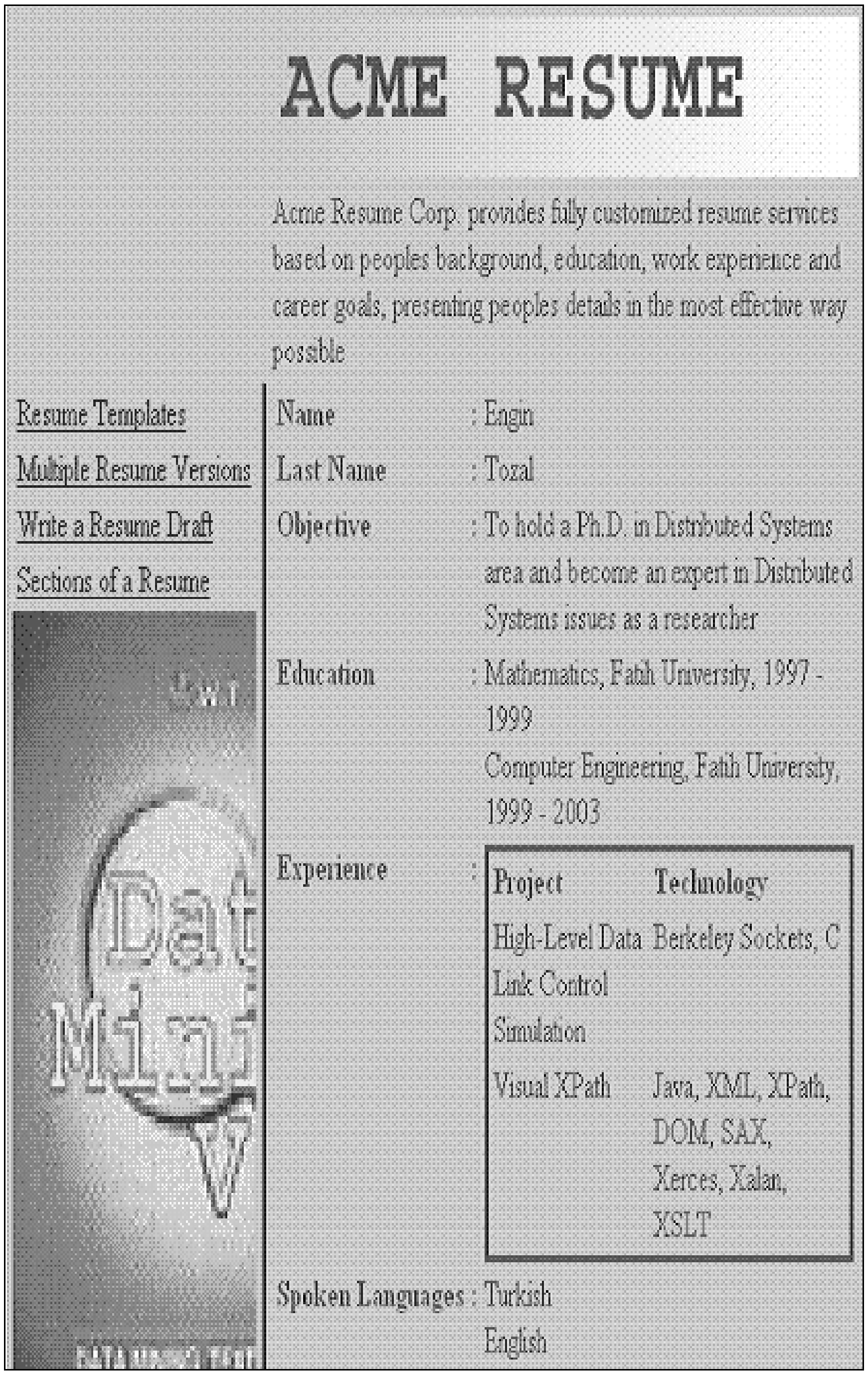
Figure 2.2 XSL Transformation

As the information is kept in XML format, a different stylesheet is used for each client application to transform the same information into different presentation formats according to client application’s capabilities. To illustrate, the content is transformed into XHTML for a web browser, whereas same content is transformed into WML for handheld mobile devices. The other clear advantage is the people developing business logic can work independently from those who are developing user interface. Figure 2.3 shows an XSLT stylesheet that transforms the sample xml document given at Figure 2.1 into XHTML. The Browser view of result is given at Figure 2.4.



```
<?xml version="1.0" encoding="UTF-8"?>
<xsl:stylesheet xmlns:xsl="http://www.w3.org/1999/XSL/Transform"
version="2.0">
   <xsl:output method="xml" indent="yes"
   doctype-system="http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd"
   doctype-public="-//W3C//DTD XHTML 1.0 Transitional//EN" />
   <xsl:template match="/">
      <html xmlns="http://www.w3.org/1999/xhtml">
         <head>
           <meta name="keywords"
           content="resume, career, experience, cv, education, job"/>
           <meta name="description"
           content="Acme resume and career services"/>
             <title>Acme Resume Corp.- <xsl:value-of
              select="resume/personal/name"/>
                <xsl:value-of select="resume/personal/last-name"/></title>
             <link rel="stylesheet" type="text/css" href="sample.css"/>
         </head><body><table width="100%">
             ..............
                <tr><td><xsl:text> </xsl:text></td><td/>
                   <td>Acme Resume Corp. provides fully customized resume
                      services based on peoples background, education,
                      work experience and career goals, presenting peoples
                      details in the most effective way possible</td></tr>
                <tr valign="top"><td><table><tr><td nowrap="nowrap">
                            <a href="">Resume Templates</a></td></tr>
                       <tr><td nowrap="nowrap">
                            <a href="">Multiple Resume Versions</a></td></tr>
                       <tr><td nowrap="nowrap">
                            <a href="">Write a Resume Draft</a></td></tr>
                       <tr><td nowrap="nowrap">
                            <a href="">Sections of a Resume</a></td></tr>
                      </table>
                      <img src="./dm.jpg" alt="Data Mining" height="100%"/></td>
                   <td><img src="./vline.gif" alt="" height="100%" width="3px"/>
                   </td><td>
                      <table><tr><td class="header">Name</td>
                            <td class="header">:</td><td>
                              <xsl:value-of select="resume/personal/@name"/></td>
                         </tr><tr>
                            <td class="header">Last Name</td>
                            <td class="header">:</td><td>
                               <xsl:value-of select="resume/personal/@last-name"/>
                            </td></tr><tr>
                            <td class="header" valign="top">Objective</td>
                            <td class="header" valign="top">:</td>
                            <td valign="top">
                               <xsl:value-of select="resume/personal/objective"/>
                            </td></tr>
                         <tr valign="top"><td class="header">Education</td>
                            <td class="header">:</td><td>
                               <xsl:for-each select="resume/personal/education">
                                 <xsl:value-of select="./department"/>,
                                 <xsl:value-of select="./collage"/>,
                                 <xsl:value-of select="./@start"/> -
                                 <xsl:value-of select="./@end"/><br/>
                               </xsl:for-each></td></tr>
            ..............
                  </table></td> </tr></table></body></html>
   </xsl:template>
</xsl:stylesheet>
```

Figure 2.3 Sample XSL Stylesheet



ACME RESUME

Acme Resume Corp. provides fully customized resume services based on peoples background, education, work experience and career goals, presenting peoples details in the most effective way possible

Resume Templates

Multiple Resume Versions

Write a Resume Draft

Sections of a Resume

Name	: Engin
Last Name	: Tozal
Objective	: To hold a Ph.D. in Distributed Systems area and become an expert in Distributed Systems issues as a researcher
Education	: Mathematics, Fatih University, 1997 - 1999 Computer Engineering, Fatih University, 1999 - 2003
Experience	:
Spoken Languages	: Turkish English

Project	Technology
High-Level Data Link Control Simulation	Berkeley Sockets, C
Visual XPath	Java, XML, XPath, DOM, SAX, Xerces, Xalan, XSLT

Figure 2.4 Sample XHTML Browser View

2.2 NAÏVE BAYES TEXT DOCUMENT CLASSIFICATION

Naïve Bayes [9] is a widely used classification technique based on a simple theorem of probability that is called Bayes' rule.

$$P(A \mid B) = \frac{P(A) * P(B \mid A)}{P(B)} \qquad 2.1$$

Simply our posterior belief *P(A|B)* is calculated by multiplying our prior belief *P(A)* and by the likelihood *P(B|A)* that *B* will occur if *A* is true.

In Naïve Bayes IR text classification technique [5] [7]; given the set of categories $c=\{c_1,c_2,\ldots,c_m\}$, and a set of documents $d=\{d_1,d_2,\ldots,d_k\}$ a document *d* is represented as a bag of unique words occurring in the document. As a matter of fact the document is represented as an *n* dimensional feature vector $d=\{w_1,w_2,\ldots.w_n\}$ where the occurrence frequency of i^{th} word is kept as value of a feature w_i, $1 \le i \le n$. An estimation of conditional class probability of document d_j belongs to category c_i is obtained by formula

$$P(c_i \mid d_j) = \frac{P(c_i) * P(d_j \mid c_i)}{P(d_j)}, \; 1 \le i \le m, \text{ and } 1 \le j \le k \qquad 2.2$$

$P(d_j)$ which is prior document probability is same for each class *i* so there is no need to calculate. Prior class probabilities $P(c_i)$ for each class *i* can be estimated from the frequencies of documents belonging to class c_i in the training data. Estimating the probability of the feature vector document d_j given the class c_i, $P(d_j|c_i)$ is expensive to compute, due to the fact that a feature w_r can take a big number of values. In order to make the calculation simple each feature is assumed to be independent, this is the core of Naïve Bayes Classifier model. So $P(d_j|c_i)$ is calculated under Naïve Bayes multinomial model [8] by formula;

$$P(d_j \mid c_i) = \prod_{w \in d_j} P(w \mid c_i)^{f(w,d_j)} \qquad 2.3$$

f(w,d$_j$) is the number of occurrences of word *w* in document *d$_j$*. However to avoid zero probabilities Laplace Smoothing [10] is commonly used.

$$P(w \mid c_i) = \frac{1 + f(w,c_i)}{|V| + \sum_{w' \in V} f(w',c)} \qquad 2.4$$

f(w,c$_i$) is the number of occurrences of word *w* in class *c$_i$*. The model adds 1 to occurrence of each word in the vocabulary *V*.

2.3 SEMI-STRUCTURED DOCUMENT CLASSIFICATION

Semi structured documents are text files that contain both textual data and content labels that marks up the textual data. Conventional IR text classifiers run over flat text documents, however the hierarchical structure of textual data contains valuable information about the nature of text, so exploiting structure information can increase accuracy of classification.

Component is part of a semi structured document which contains either structured or non-structured textual data. Due to the fact that; document structure is hierarchical, components may also include other components. In XML a component corresponds to element.

Yi, Sundaresan [11] propose a model called structured vector model, which is based on *component tagging* [10] it tags each word with the name of the component in which it resides. The tagging process is done hierarchically as a result each word occurrence is treated as different features. Moreover terms from the same component are grouped together and differentiated from the same term that appears in a different component.

Denoyer, Gallinari [12] propose a general model based on Bayesian Networks called *component splitting* [10]; each component in a document is handled independently and training is done over each of them. The total document probability is estimated by product of individual component probabilities. This model is used to classify not only complete documents but also document fragments.

$$P(d_j \mid c_i) = \prod_{s \in d} \prod_{w \in s} P_s(w \mid c_i)^{f_s(w|d_j)} \qquad 2.5$$

s represents each structural component in document d_j and P_s is trained for each component. $f_s(w,d_j)$ is the occurrence frequency of word w in structure s of document d_j

2.4 WEB CLASSIFICATION

Since World Wide Web grows rapidly, it has become necessary to develop automated tools for finding, searching and organizing web documents for end users. Some web portals like Yahoo [1] and Google [2] provides directory services which maps web sites and/or documents to predefined categories. Web pages consist of text content and tags for presentation of content. A number of methods which use text and/or context features are proposed to classify web pages [19, 20].

The simplest classification method is *text-only* classification. All markups are removed from document and classification is based on the remaining information content. Mladenic [17] proposes a method where a document is represented as feature vector which includes *n-grams* instead of *unigrams* (n is determined to be 5). The documents are analyzed and all stop words are removed, then the features are generated in n passes where *i-grams* are generated at i^{th} pass. At the end, all low frequency (*frequency* < *4*) features are deleted and Naïve Bayes algorithm is used to classify those documents.

Hypertext Approach analyzes presentation information as well as text information. The layout of a page which is defined by HTML tags are features that can be considered in classification [18], for example information between <b></b> tags or the text has greater font size seem to be more important than others. Paper [19] shows that the title and the anchor words exist in an HTML document are important and represents a web page *P* as $P= \{x.w_i, t.w_j, a.w_k\}$ where *x* is for textual content, *t* is for title content and *a* is for anchor content. It uses Support Vector Machines to classify web pages.

Link Analysis technique classifies the pages according to text on the link and the documents that those links refers [21, 22]. A web page has *in-neighbors*, pages refer to it and *out-neighbors*, pages that it refers. In-neighbors and out-neighbors may contain valuable information when it comes to classify a document. Experiments show that simply including textual content of neighbors degrades the accuracy of classification, because of a page may have neighbors that are not related to its content directly. Chakrabarti, Dom, and Indyk [22] suggest a method that uses the topics of pages instead of occurring words to determine the linking behavior.

CHAPTER 3

XSLT WEB CLASSIFICATION FRAMEWORK

Because of widespread availability of the Internet, web applications play an important role in conducting business, disseminating information and communicating with people. *Web Application* is a type of application which delivers services to users through the Internet. Classical approach delivers information in HTML format to end-user. But maintenance of HTML based applications is difficult due to drawbacks such as mixed presentation and content, different client applications, lack of advanced features etc. On the other hand, use of XSLT addresses the following problems:

- Information is delivered to different clients such as web browsers, mobile phones, PDAs, TV sets that have diverse visual capabilities. A different presentation is required for each type of client. This problem can be solved by storing information in XML format and preparing different XSLT stylesheets to transform them in to a suitable presentation format for each client.
- HTML mixes the presentation and actual content, so the graphical user interface designer and programmer have to work on the same document. On the other hand XSLT separates layout and presentation from content.
- Usually data is originated from different sources; XSLT can merge and present them in a single page easily.

- XSLT allows the developer to sort, filter or manipulate the content of XML, with use of loops, conditional switches, regular expressions, and other built-in functions.
- An XSLT stylesheet is applied to the XML document at client side so the overload at server side is greatly reduced.
- Because an XSLT stylesheet can be shared by a number of XML documents, presentation is managed at a single point.

All these advantages make XML/XSLT an important technology for web applications.

Web document classification classifies web pages or sites according to their content to build web directories. However differentiating a "automobile" web page that has content about an automobile under category **Shopping/Autos** from a "news" page about a new car model under **News/Magazine** would be difficult if content based classification was used.

The result of transforming an XML document with XSLT is an HTML document. This allows three web page classification options*; HTML Classification, Semi-Structured Document (XML) Classification or XSLT classification –*a classification scheme where XSLT related data is utilized-. There are many studies on the first two options, however the last is not considered yet to the best of our knowledge, and it is the starting point of our novel proposition.

HTML classification and Semi-Structured Document classifications are explained at sections 2.3 and 2.4. *XSLT classification;* is a hybrid classification technique that exploits both structure of XML document and markup features embedded in result XHTML document.

We believe that; XSLT classification is better than HTML classification because:

1. The XML document itself contains pretty valuable structure information about content.

2. Tag and attribute names in XML are important for text classification and can not be ignored in the process.

We believe that XSLT classification is better than XML classification because

1. An XML document usually contains meta-data that are not related to actual content but used internally for different purposes by the generators of document. Usually those data are not presented to end-user, so should be omitted in classification process. Elements; *type, id, create-date, modification-date and copyright* are examples of meta-data in Figure 2.1.
2. Some of the information presented to end-user does not come from any XML document but are string literals embedded directly into the HTML tags used in XSLT stylesheets. That data can be useful in classification process; the sample XSLT in Figure 2.3 embeds valuable literal content that can not be ignored.
3. Sometimes an XML document is a large document with lots of information, but different parts are presented to different users while the rest is suppressed, so each transformed version should be considered and classified as a different document, rather than classifying the complete document as a whole.
4. Sometimes the information generated to end-user is merged from different XML documents, so considering and classifying the transformed document as a single page can be more appropriate than classifying each document separately.

An XSLT Classifier Framework which is based on pluggable components is implemented. The framework consists of three modules: *Preprocessor, Semi-Structured Document Modeler, and Classifier* (Figure 3.1). The system accepts blocks of, one or more source XML documents and an XSLT stylesheet which transforms these documents into XHTML to be viewed by end-user.

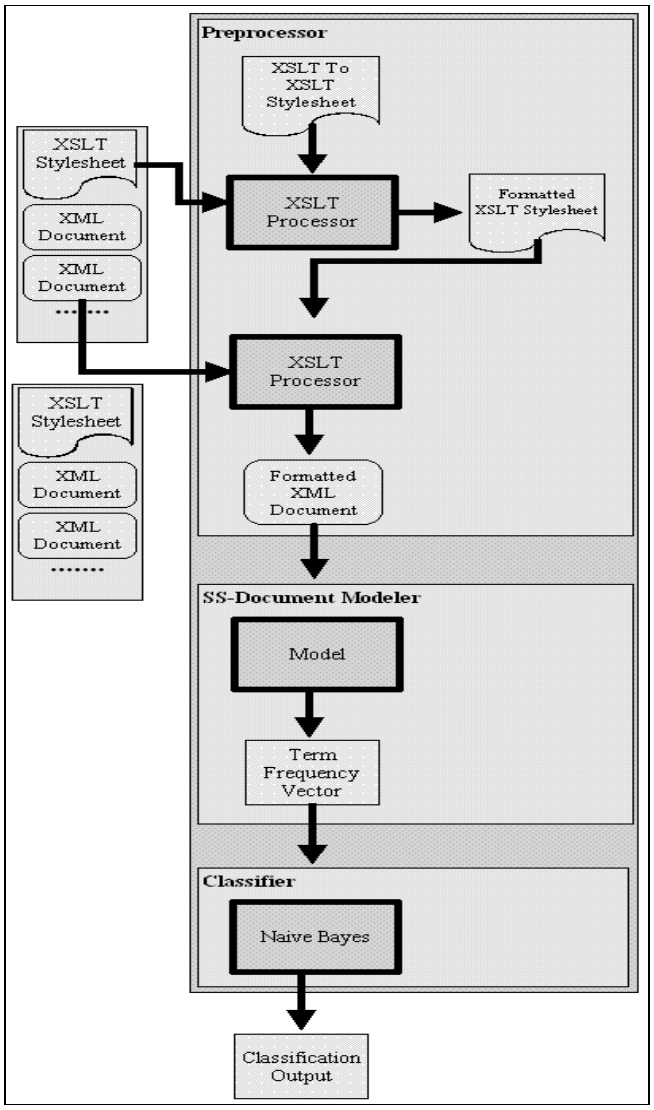


Figure 3.1 XSLT Classification Framework Architecture

The process can be summarized as:

1. The original XSLT document is passed to the Preprocessor which produces a different XSLT document named *formatted XSLT stylesheet.* Formatted XSLT is a version of original XSLT stylesheet which has xsl templates to produce ancestor-or-self hierarchies of referenced XML fragments. This

information will be used to prefix content with its ancestors while generating term frequency vectors for documents.

2. An XSLT Processor applies the formatted XSLT stylesheet to the original XML documents to generate *formatted XML documents*. Formatted XML documents consist of all string literals embedded into the original XSLT stylesheet, content of HTML *meta, title, anchor* tags and source XML fragments that are referenced only in original XSLT stylesheet. The textual content of each source XML fragment is surrounded with its ancestor-or-self-hierarchies separated by "-_-" character sequence as in Figure 3.2.
3. Formatted XML documents are given to Semi-Structured Document Modeler which generates term frequency vectors for each instance.
4. Term frequency vectors are given to Classifier for building classification model.

3.1 PREPROCESSOR

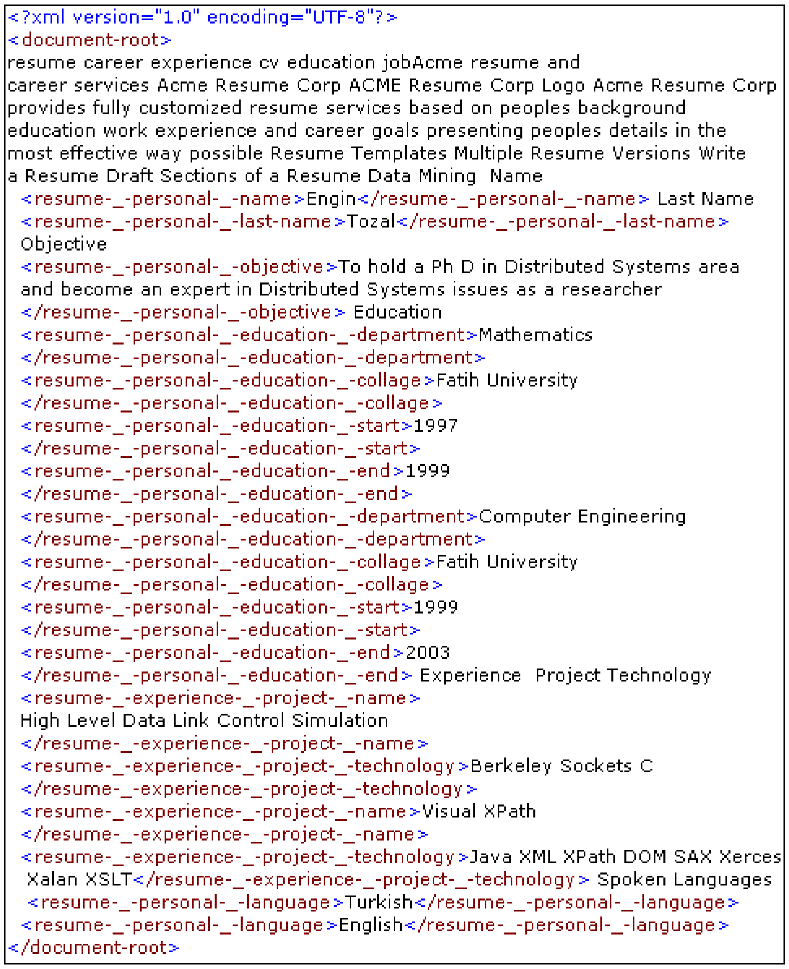
In the preprocessing step an XSLT-to-XSLT stylesheet is applied to the original XSLT stylesheet to generate another XSLT stylesheet called *formatted stylesheet*. Because an XSLT document is an XML document, XSLT-to-XSLT stylesheet simply traverses each element of the original XSLT document and does the following;

- If the current node is an *xsl:element* node, it is used to print out an HTML tag or a tagged text so if it is an HTML element remove it otherwise insert it into result tree and process its child-nodes.
- If the current node is an *xsl:vlaue-of* element than the *select* attribute of the *xsl:vlaue-of* can refer to an element or attribute node in the source XML document, so normalize space and remove all punctuation characters of the content and insert it into the result tree with all its *ancestor-or-self* hierarchy.

If the current node is an attribute, then the name of the attribute is considered to be part of *ancestor* hierarchy.

- If the current node is an *xsl:text* element normalize space and remove all punctuation characters of the content and insert it into the result tree with all its *ancestor-or-self* hierarchy. If the identified node is an attribute than the name of the attribute is considered to be part of *ancestor* hierarchy.
- If the current node is any other XSLT element -*xsl:variable, xsl:param, xsl:with-param, xsl:if, xsl:when, xsl:choose, xsl:otherwise, xsl:copy, xsl:copy-of, xsl:sort, xsl:for-each*- put it directly into the result tree and process its children.
- If the current node is an HTML *meta* tag whose name attribute is *keyword* or *description* insert its content into the result tree.
- If the current node is an HTML *img* tag inserts its *alt* attribute value into the result tree.
- If any other string literals exist in the XSLT document, simply normalize space and remove all punctuation characters of the string and insert it into the result tree.

The result of XSLT-to-XSLT transformation is formatted XSLT stylesheet which is used to transform source XML documents into formatted XML documents, instead of HTML. Figure 3.2 shows the formatted XML document generated by using XSL stylesheet in Figure 2.3 and original XML document in Figure 2.1.



```
<?xml version="1.0" encoding="UTF-8"?>
<document-root>
resume career experience cv education jobAcme resume and
career services Acme Resume Corp ACME Resume Corp Logo Acme Resume Corp
provides fully customized resume services based on peoples background
education work experience and career goals presenting peoples details in the
most effective way possible Resume Templates Multiple Resume Versions Write
a Resume Draft Sections of a Resume Data Mining  Name
 <resume-_-personal-_-name>Engin</resume-_-personal-_-name> Last Name
 <resume-_-personal-_-last-name>Tozal</resume-_-personal-_-last-name>
 Objective
 <resume-_-personal-_-objective>To hold a Ph D in Distributed Systems area
 and become an expert in Distributed Systems issues as a researcher
 </resume-_-personal-_-objective> Education
 <resume-_-personal-_-education-_-department>Mathematics
 </resume-_-personal-_-education-_-department>
 <resume-_-personal-_-education-_-collage>Fatih University
 </resume-_-personal-_-education-_-collage>
 <resume-_-personal-_-education-_-start>1997
 </resume-_-personal-_-education-_-start>
 <resume-_-personal-_-education-_-end>1999
 </resume-_-personal-_-education-_-end>
 <resume-_-personal-_-education-_-department>Computer Engineering
 </resume-_-personal-_-education-_-department>
 <resume-_-personal-_-education-_-collage>Fatih University
 </resume-_-personal-_-education-_-collage>
 <resume-_-personal-_-education-_-start>1999
 </resume-_-personal-_-education-_-start>
 <resume-_-personal-_-education-_-end>2003
 </resume-_-personal-_-education-_-end> Experience  Project Technology
 <resume-_-experience-_-project-_-name>
 High Level Data Link Control Simulation
 </resume-_-experience-_-project-_-name>
 <resume-_-experience-_-project-_-technology>Berkeley Sockets C
 </resume-_-experience-_-project-_-technology>
 <resume-_-experience-_-project-_-name>Visual XPath
 </resume-_-experience-_-project-_-name>
 <resume-_-experience-_-project-_-technology>Java XML XPath DOM SAX Xerces
  Xalan XSLT</resume-_-experience-_-project-_-technology> Spoken Languages
  <resume-_-personal-_-language>Turkish</resume-_-personal-_-language>
 <resume-_-personal-_-language>English</resume-_-personal-_-language>
</document-root>
```

Figure 3.2 Formatted XML Document

3.2 SEMI-STRUCTURED DOCUMENT MODELER

Most XML documents in the World Wide Web are generated by different people and do not conform to any common markup vocabulary or structure. Nevertheless two situations frequently occur in XML document collections that are to be classified by automated means; those documents which hold different content but have similar markup and documents which hold similar content but have different markup as explained below:

1 XML documents may have similar markup but different content: For example; XML documents from various *movie* sites usually use elements like *title, year, director, producer, actor, actress* etc. Consider 2 different web pages in 2 different movie sites which will usually have similar markup but different content. A flat text classifier may incorrectly classify these 2 web pages into 2 different categories since markup is ignored in those classifiers.

2 XML documents may have similar content but different markup: For example, an XML document about a new brand of automobile under "Shopping/Autos" category may have similar content with another XML document under "News/Magazine" category containing an article on the same automobile. As shown in Figure 3.3, despite the similarity in content, they extremely differ in markup. A flat text classifier may incorrectly classify the News document as Automobile or vice versa

```
<automobile>
<manufacturer>BMW</manufacturer>
    <model>BMW 525i Sedan</model>
    <year>2006</year>
    <technical>
        <feature>6-speed manual
transmission
        </feature>
        <feature>Compact spare tire
        </feature>
    </technical>
</automobile>
```

```
<article id="10010505">
    <author>Jim Smith</author>
    <date>10, September,2005</date>
    <title> BMW 525i Sedan</title>
      <paragraph>BMW announces BMW
525i Sedan 2006 model automobile.
Important aspects of this new brand
automobile are; 6-speed manual
transmission, Compact spare tire..
    </paragraph>
</article>
```

Figure 3.3 Sample XML Document Fragments from an Automobile and a Magazine Site

It is obvious that markup or structural variations may occur among documents in the same category, since they are created by different people most of the time. Those structural variations cane be summarized as follows:

- *Renaming:* Element or attribute names that tag the same data could be different in different documents. For example, last_name and family_name are different markup for the same content. Thus we say that last_name is renamed to family_name in the second document. Such renaming occurs frequently.
    - <student><last_name>…</last_name></student>
    - <student><family_name>…</family_name></student>
- *Insertion:* Let's consider the markup of 2 documents in the same category. In many cases we see that one document will have one or more element that doesn't occur in the other document. Put in other words, one document will have one or more element *inserted* into it, assuming the rest of markup is the same. In the following example, the second document has <name> element inserted into it:
    - <student><fname>… </fname><lname>…</lname> </student>
    - <student><name><fname>… </fname><lname>…</lname></name> </student>
- *Deletion:* Frequently some elements that occur in one document may be missing in some another which can be regarded as the deletion of an element. Since we are considering a set of documents, deletion can be seen as the opposite of insertion depending on the point of view. For example, <name> element is deleted from the second document to obtain the first in the example above.
- *Swap:* Elements in markup can swap places between documents. The simplest swap case is between a parent and a child element as shown in the example below:
    - <music><song><singer>…</singer></song></music>
    - <music><singer><song>…</song></singer></music>

Surely the modification listed above makes document classification more difficult. Any classification method that uses structure in addition to content must consider that such modifications are frequent in real life. As a result, modeling the structure under these modifications become important. A structure representation model should be flexible enough to recognize the similarity between document structures in such variations or modifications.

Eight different document representation models based on component tagging are presented in this section informally. We assume that documents are converted into term frequency vectors and fed to a classifier, which, in our experiments, is Naïve Bayes.

In capturing the structure, the main idea is to tag words in text content with element or attribute names in ancestor hierarchy which consists of all nodes from current node to the root of the hierarchy. For example; *"resume.personal.language"* is the ancestor hierarchy of word *"English"* in Figure 2.1 and *"resume.personal"* is the ancestor hierarchy of component or element *"language"*.

Different representation models can be obtained with different tagging techniques. These models can mainly be divided into 2 categories on (i) strict or (ii) loose ancestor hierarchies which will also be called strict and loose models. Strict ancestor hierarchy models opt for describing the nesting of elements surrounding text in a more direct or obvious manner, while loose ancestor hierarchy models chooses to denote the nesting or structure in an indirect or less noticeable manner.

In consideration of document modifications or variations described in the previous section, we can expect that strict models would produce lower classification rates because they fail recognize the modifications. Loose models, on the other hand, should be more flexible and adaptable to the modifications of the structure.

In cases of both strict and loose models, the structure can be fully or partially captured by the model. It is clear that as we go from partial to full in capturing the structure, model (term frequency vector) size increases resulting in lower performance time in classification. However we could expect to capture more of the structural similarities with models capturing full structure. This may not necessarily result in higher classifications rates, because one cannot make the assumption that overall document similarity increases with structural similarity between documents.

The models (itemized A thru H) are explained below. Model C, Model D, and Model G are based on *strict-ancestor hierarchy* in which an item is tagged with the complete ancestor path. Model E, Model F, and Model H are based on *loose-ancestor hierarchy* in which an item is prefixed with ancestor elements separately.

We use the example in Figure 2.1 The words are shown in italic and tags are in normal fonts.

A. The first model s straightforward. Element/attribute names are simply added to the term frequency vector. No structural relationship between text and components are captured in this model

B. Second model is based on prefixing each word with the surrounding element. E*.g. language.english, collage.fatih*. This model does not capture the complete logical structure but only the relationship of a word to its parent element.

```
collage.fatih
collage.university
department.mathematics
language.english
…
```

C. Third model is based on prefixing each word with its complete ancestor hierarchy *e.g. resume.personal.language.english* for the word *english*. The complete ancestor hierarchy is captured in this model. However this model is vulnerable to structural variations such as element renamings, insertions, deletions, and swaps.

```
resume.personal.language.english
resume.personal.education.department.mathematics
…
```

D. Fourth model is based on prefixing each word with all partial sub-paths between ancestors and a word. The word itself is added in term frequency vector too. The model captures the structure just in the context of the word; inter-element structure is not regarded in classification process. But it is not as vulnerable as the previous model under modifications. Model D subsumes Model C.

```
resume.personal.language.english
       personal.language.english
                language.english
                         english
…
```

E. In this model words are prefixed with each ancestor elements. The word itself is included in term frequency vector as well. The structure is kept in a very loose manner, it is not hierarchical, and inter-element structure is not captured. However it is not too vulnerable to structural alterations. Model E subsumes Model B.

```
resume.english
personal.english
language.english
english
…
```

F. Prefixing each word and element/attribute with the each ancestor is the idea in this model. The words and element/attributes are also added to term frequency vector. Although the structure is captured in a loose manner, complete document hierarchy is captured. Inter-element structure is captured as well. Again it is resistant to structural alterations to some degree. Model F subsumes Model E.

```
resume
personal
resume.personal
language
```

```
personal.language
resume.language
english
language.english
personal.english
resume.english
…
```

G. Model G prefixes each word/attribute/element with all unique sub-paths in the ancestor hierarchy. The word and element/attributes are added into term frequency vector as well. The complete document structure is captured. This model is extremely vulnerable to structural alterations. Model G is a combination of Model D and Model B.

```
resume
personal
resume.personal
language
personal.language
resume.personal.language
english
language.english
personal.language.english
resume.personal.language.english
…
```

H. Prefixing each word/element/attribute including the ancestor elements with each ancestor element is the basis of this model. The word and element/attributes are added into term frequency vector as well. Ancestors are prefixed with their own ancestors, too. Also descendants of elements in ancestor hierarchy used to prefix the element e.g. *language.personal, language.resume* for the element *language*. Although the structure is captured in a loose manner, complete document hierarchy is captured. Inter-element structure is captured in two ways (i.e. from ancestor to descendant and from descendant to ancestor). This model is, too, resistant to structural alterations to some degree. Moreover, this model is resistant to inter element swaps. However, the number of terms in term frequency vector is increased compared to Model F. Model H is a superset of Model F.

```
resume
personal
resume.personal
personal.resume
language
personal.language
language.personal
resume.language
language.resume
english
language.english
personal.english
resume.english
…
```

Since each model captures structure at different levels and in different ways, we can expect to see different classifications rates as discussed in the chapter 4.

3.3 CLASSIFIER

The Classifier accepts term frequency vectors produced by Semi-Structured Document Modeler and builds a classification model before classifying newly arrived documents.

Naïve Bayes Classifier is a widely used classification technique in IR community due to both its simplicity and accuracy, so we prefer to use Naïve Bayes as classifier in our framework. However any other classifier can be plugged instead of Naïve Bayes.

CHAPTER 4

EXPERIMENTAL RESULTS AND EVALUATION

Three different experiments are performed in the frame of this study; the first experiment is performed to figure out how the semi-structured document models are affected under structural variations of *deletion, renaming, insertion, and swap*. The second one is for comparing seven different semi-structured document representations mentioned at section 3.2. The third is for comparing XSLT, XML and HTML classifications. The Semi-Structured document model that gives lowest error rate in the second experiment is used to represent formatted XML documents in the third experiment. Instead of applying any advanced HTML classification technique mentioned at section 2.4, a simple hypertext approach that incorporates values of HTML *meta* and *img* tags is used in third experiment. However, because of XSLT classification is a hybrid technique that exploits advantages of both Semi-Structured Document and HTML classification techniques any specific HTML classification method will improve accuracy of XSLT method as well.

4.1 ENVIRONMENT

Weka[2] is an open source software issued under the GNU General Public License which contains a collection of machine learning algorithms and data mining tasks. It is completely developed in Java and provides many other tools for non-programmers.

Saxon-B[3] is an open source, non-schema-aware XSLT and XQuery processor which is designed to conform to the basic conformance level of XSLT 2.0, and the equivalent level of functionality in XQuery 1.0. The framework implemented in this study uses Weka 3.4 and Saxon-B 8.4.

4.2 DATASET

Most web applications perform XML to XHTML transformations on server side because of browsers does not always support transformations on client side at the times XSLT came out. Although popular web browsers like IE, Firefox, Netscape, Mozilla support XSLT now, legacy systems that perform transformation on server side still exists. Moreover current dataset repositories on the web do not provide a proper dataset for our research. So we generated XML/XSLT version of 20 different sites belonging to 4 different categories; Automotive, Movie, Software, News & Reference. The sites belong to News & Reference contains news and articles about movies, automobiles and software health and literature to make the classification more difficult. The list of sites and all dataset can be viewed and downloaded[4] from the web site. One hundred XML documents that hold the information published on web sites are generated. These documents are evenly distributed among categories. Headers exist in HTML page are used to surround content as attributes or elements in XML documents. *XML documents have variant structures, element and attribute names, and order to mimic that they are generated by different people*. For each

[2] http://www.cs.waikato.ac.nz/~ml/weka/index.html
[3] http://www.saxonica.com
[4] http://www.fatih.edu.tr/~engin

site an XSLT stylesheet which produces exactly the same presentation with all links, images, embedded objects, literal strings and non-printable data like meta, style, script tags and their contents of actual HTML page is generated. When the XSLT is applied to the XML document it produces all static content that exists at each page of a site and brings the dynamic content from XML documents to produce a valid XHTML document.

4.3 TRAINING AND TEST SELECTION

Any supervised classification algorithm should have some knowledge of data to build a model before testing any instance. To establish a model the algorithm is given a set of *training data* as well as correct classification assignments of the data. To perform the test, entire data set is divided into two partitions the first one is *training set* and the latter is *test set*, it is common in literature that 2/3 of data set is used for training and the rest is for test. However randomly selecting the 66 percent of data as training set is not robust to develop any hypothesis based on test data. Cross Validation [23] is a common technique used to estimate generalization error. The data set is divided *n* mutually exclusive subsets of approximately equal size. The training algorithm runs *n* times, each time *n-1* subsets are used as training set and the subset leaved-out is used as test set. The estimated accuracy of cross validation is computed; dividing total number of correct classifications by number of all instances in the data set. N-Fold Cross Validation is used in our experiments where *n* is 10.

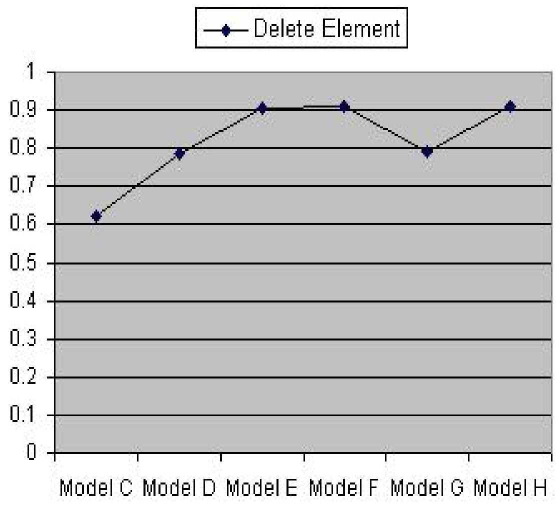
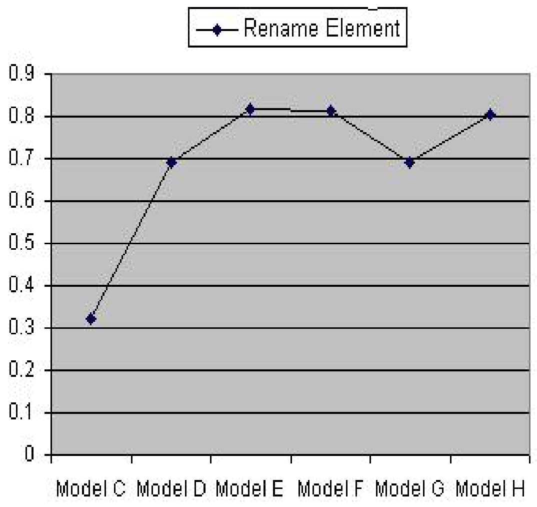
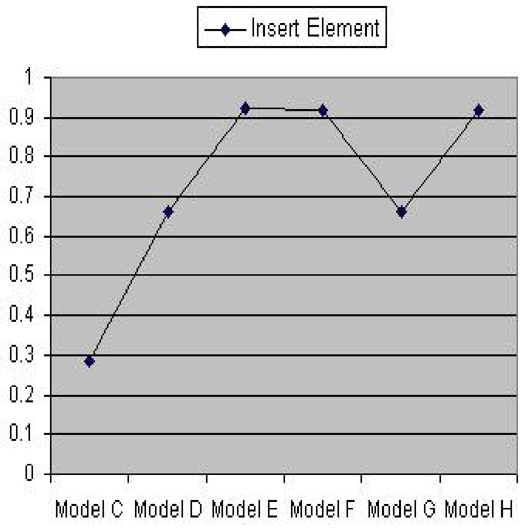
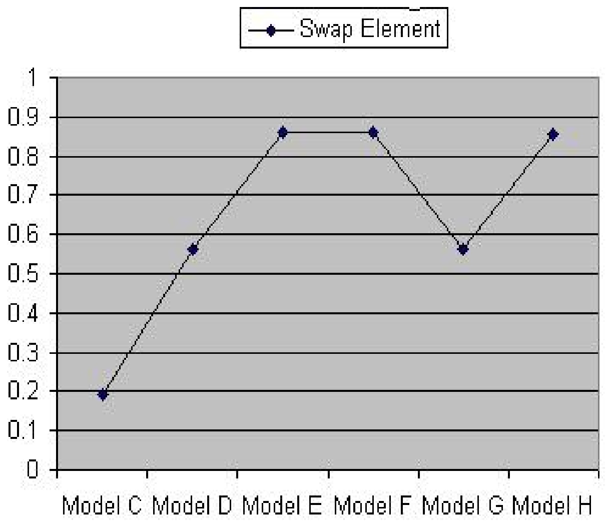
4.4 EVALUATION

Experiment 1

These experiments are performed to figure out how the models are affected under structural variations of *deletion, renaming, insertion, and swap*. 33% of instances from the dataset are altered manually. For each variation a new version of the file is generated. Only one single element is changed in each variation. The changes are done at points in documents where we felt it could produce highest effect on the outcome of the experiment.. The cosine similarity is used to compare the original document with its altered versions. A document d is represented as term frequency vector, which holds the occurrence frequency of each unique stemmed word w in document. The cosine similarity of two documents is computed by cosine of angle between two vectors as denoted below.

$$\cos ine\_similarity(dj, dk) = \frac{\sum_{i=1}^{n} w_{i,j} . w_{i,k}}{\sqrt{\sum_{i=1}^{n} w^2{}_{i,j}} . \sqrt{\sum_{i=1}^{n} w^2{}_{i,k}}} \qquad 4.1$$

Figure 4.1 thru Figure 4.4 show comparison of Model C, D, E, F, G, and H under structural variations.

 **Figure 4.1 Deletion**	 **Figure 4.2 Rename**
 **Figure 4.3 Insertion**	 **Figure 4.4 Swap**

As shown in the figures, model E, F, and H which are based on loose-ancestor hierarchy are, in general, more robust to alterations than Model C, D, and G, making loose models clearly superior to strict models. All models display a similar similarity behavior with regards to all four operations. Note that in these experiments only a single modification is performed. With more of the same modifications, or with the combinations of different modifications we can expect similar results, since operation-wise behavior is similar.

Experiment 2

This experiment is performed to discover the best semi-structured document representation model with regards to classification rates. We took the data set and created 8 different versions of term frequency vectors corresponding to 8 different models. 2/3 of dataset is used for training, 1/3 of dataset is used for testing with 10-Fold Cross Validation using Naïve Bayes in Weka. The results are shown in Figure 4.5.

As shown in the figure; Model B, in which words are tagged just with their parent element label, is better than Model A. Even though documents have various element/attribute names, and different structures, similarity of labels used to format hierarchy of textual content increases from top to down (i.e. from general node to specific leaf node).

Model C, Model D, and Model G which are based on strict-ancestor hierarchy have higher error rates. Although those models capture complete document structure, they are vulnerable to structural alterations. Yet Model G is better than previous two because; it captures inter-element structure. Furthermore if alteration occurs at parent element of a term Model D results worse. Model D is better than Model C because not only it is a super set of Model C but also similarities in elements used for hierarchical structure, increases from root node to leaf node. Again Model B is better than Model D even it is subset of Model D; because of the same claim holds for Model D and Model C. Model A and Model B does not capture complete structure and they have lower accuracy rates than the others.

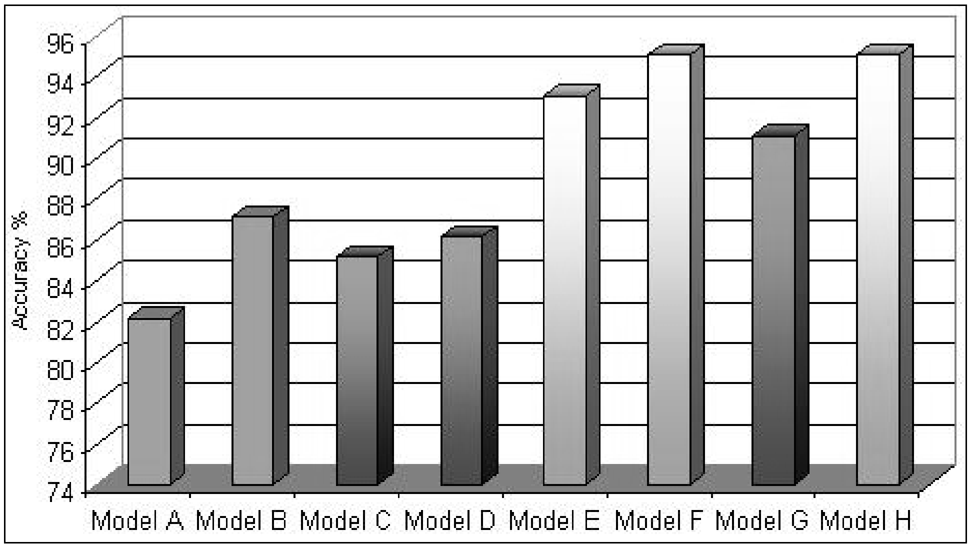


Figure 4.5 Document Representations Comparison

Model E, Model F, and Model H are based on loose-ancestor hierarchy and are not vulnerable to structural alterations as much as the models based on strict-ancestor hierarchy. Moreover Model H is more resistant to inter element swaps due to it captures inter element hierarchy both from ancestor to descendant and from descendant to ancestor where Model F captures only from ancestor to descendant. So they give better results than the others. Because Model E does not capture the inter-element structure of the ancestor path of an occurring word, Model F is better than Model E. Although Model H and Model F give same accuracy rates, Model F generates less terms to be added into term frequency vector. This makes Model F preferable compared to Model H in classification process.

Experiment 3

Third experiment compares three types of classification approaches as shown in Figure 4.6.

Simple HTML classification is based on web page content, and does not incorporate structural information exists in source XML document. So it has lowest accuracy rate -94.05%- compared to others.

XML classification uses Model 6 to represent XML document, however it does not include string literals exist in HTML tags, contents of *hyperlinks*, *meta* and *img* tags that exist in XSLT stylesheets. It also adds all meta and non-content related data (e.g. copyright notice, author, creation and last modification date, type, and category in Figure 2.1) exists in source XML documents, which should be ignored. Because, non-content related data is included and content comes from XSLT presentation is suppressed; XML classification is only 1 point -95.04%- better than HTML classification.

XSLT classification runs over *formatted XML* documents, generated by applying *formatted XSL* stylesheets to original source XML documents at preprocessing step. The preprocessing step is explained at section 3.1. All string literals in HTML tags, literals of *anchor, meta* and *img* tags that exist in XSLT stylesheet are included in this classification. In addition, only the referenced XML fragments in XSLT stylesheet are added to classification, so all meta and non-referenced data fragments exist in source XML document are suppressed. XSLT classification uses Model F to represent formatted XML documents.

The experiment shows that XSLT classification has 100% accuracy rate because we have a small data set. If we have a larger data set with more documents and more categories, it is obvious that the accuracy rates of all three approaches will decrease. Yet XSLT classification will have better accuracy rate than others.

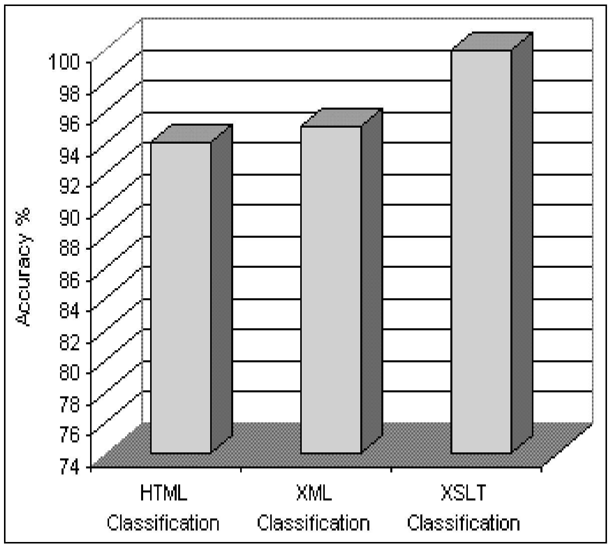


Figure 4.6 HTML, XML, and XSLT Classifications

CHAPTER 5

CONCLUSIONS

Web applications based on XML/XSLT technology allows three types of classification options; conventional Web Page classification based on HTML, Semi-Structured Document classification based on XML, and XSLT classification based on XSLT stylesheet. While HTML classification is blind to original XML document's structure, Semi-Structured Document classification ignores HTML markup and string literals exists in stylesheet, moreover includes any kind of data exists in XML document whether or not related to content presented to end-user. XSLT classification is a hybrid method that exploits both Semi-Structured Document and HTML classification.

The proposed framework consists of three modules; Preprocessor, Semi-Structured Document Modeler, and Classifier. The Preprocessor generates formatted XML documents which contain all string literals embedded into the original XSLT stylesheet, content of HTML *meta, title, anchor* tags and source XML fragments that are referenced only in original XSLT stylesheet. Besides, the textual content of source XML fragments is surrounded with its ancestor-or-self hierarchy. The Semi-Structured Document Modeler generates term frequency vectors which represent formatted XML documents based on the component tagging model plugged.

Experiments show that, component tagging models based on strict ancestor-or-self hierarchy captures document structure but are vulnerable to structural alterations. Model F results better in experiments, because it is based on loose ancestor-or-self hierarchy and captures inter-element structure. Furthermore, although HTML and Semi-Structured Document classifications have closer accuracy rates, XSLT classification is better than both.

More advanced Web Page or Web Site classification techniques may be incorporated to classification process as future work. Those techniques increases the accuracy rate of XSLT classification as well due to XSLT takes advantages of both HTML and Semi-Structured Document classification methods.

APPENDIX XSLT TO XSLT STYLE SHEET

```
<?xml version="1.0" encoding="UTF-8"?>
<xsl:stylesheet xmlns:xsl="http://www.w3.org/1999/XSL/Transform"
    xmlns:myxsl="http://MyxslPrefixAlias" version="2.0">
    <xsl:variable name="html401">abbr acronym address applet area b base
        basefont bdo big blockquote body br button caption center cite
code col
        colgroup dd del dir div dfn dl dt em fieldset font form frame
frameset
        h1 to h6 head hr html i iframe img input ins isindex kbd label
legend li
        link map menu meta noframes noscript object ol optgroup option p
param
        pre q s samp script select small span strike strong style sub sup
table
        tbody td textarea tfoot th thead title tr tt u ul var
xmp</xsl:variable>
    <!-- define output method which is xml -->
    <xsl:output method="xml" indent="yes"/>
    <!-- strip the elements that consists of only white-space characters
-->
    <xsl:strip-space elements="*"/>
    <!-- use namespace-alias to convert myxsl->xsl at the output tree -->
    <xsl:namespace-alias stylesheet-prefix="myxsl" result-prefix="xsl"/>
    <!-- start from root to traverse the input xsl document tree -->
    <xsl:template match="/">
        <!-- process child nodes -->
        <xsl:apply-templates select="child::*"/>
    </xsl:template>
    <!-- the backbone template rule which analyzes,processses and calls
itself again
    to process children of a current node -->
    <xsl:template match="*">
        <xsl:variable name="currentNode" select="."/>
        <xsl:choose>
            <!-- *** if the elemet is a top stlesheet element -->
            <xsl:when test="name()='xsl:stylesheet'">
                <!-- create a new element named xsl:stylesheet and copy
all of its attributes -->
                <xsl:element name="{name()}">
                    <xsl:copy-of select="@*"/>
                    <myxsl:output method="text"/>
                    <xsl:apply-templates select="./child::*"/>
                    <!-- insert a new template rule that processos
ancestor path to result tree -->
                    <myxsl:template match="*" mode="START-ANCESTOR-FOR-
ELEMENT">
                        <xsl:text>
                        &lt;</xsl:text>
                        <myxsl:for-each select="ancestor::*">
                            <myxsl:value-of select="concat(local-
name(),'-_-')"
                            />
                        </myxsl:for-each>
                        <myxsl:value-of select="local-name()"/>
                        <xsl:text>&gt;</xsl:text>
                    </myxsl:template>
                    <!-- insert a new template rule that processos
ancestor path to result tree -->
                    <myxsl:template match="*" mode="END-ANCESTOR-FOR-
ELEMENT">
                        <xsl:text>&lt;/</xsl:text>
                        <myxsl:for-each select="ancestor::*">
                            <myxsl:value-of select="concat(local-
name(),'-_-')"
                            />
                        </myxsl:for-each>
                        <myxsl:value-of select="local-name()"/>
                        <xsl:text>&gt;</xsl:text>
                    </myxsl:template>
                    <!-- insert a new template rule that processos
ancestor path to result tree FOR ATTÄ°BUTE-->
                    <myxsl:template match="*"
                        mode="START-ANCESTOR-FOR-ATTRIBUTE">
                        <myxsl:param
name="attributeName">none</myxsl:param>
                        <xsl:text>
                        &lt;</xsl:text>
                        <myxsl:for-each select="ancestor-or-self::*">
                            <myxsl:value-of select="concat(local-
name(),'-_-')"
                            />
                        </myxsl:for-each>
                        <myxsl:value-of select="$attributeName"/>
                        <xsl:text>&gt;</xsl:text>
                    </myxsl:template>
                    <!-- insert a new template rule that processos
ancestor path to result tree FOR ATTÄ°BUTE-->
                    <myxsl:template match="*" mode="END-ANCESTOR-FOR-
ATTRIBUTE">
                        <myxsl:param
name="attributeName">none</myxsl:param>
                        <xsl:text>&lt;/</xsl:text>
                        <myxsl:for-each select="ancestor-or-self::*">
                            <myxsl:value-of select="concat(local-
name(),'-_-')"
                            />
                        </myxsl:for-each>
                        <myxsl:value-of select="$attributeName"/>
                        <xsl:text>&gt;</xsl:text>
                    </myxsl:template>
                </xsl:element>
            </xsl:when>
            <!-- *** if the element is an xsl:element element this is an
HTML tag or a tagged text so ignore it if it is an HTML tag, print as
text if it is an tagged text but process its content-->
            <xsl:when test="name()='xsl:element'">
                <xsl:if test="contains($html401,concat(' ',local-name(),'
'))">
                    <xsl:value-of select="local-name()"/>
                </xsl:if>
                <xsl:apply-templates select="$currentNode/child::*"/>
            </xsl:when>
            <!-- *** if the element is an xsl:value-of element -->
            <xsl:when test="name()='xsl:value-of'">
                <xsl:choose>
                    <!-- the value of select attribute of xsl:element is
a path to an attribute node -->
                    <xsl:when

test="matches(./@select,'.*(@|attribute::)[^\]]+$')">
                        <xsl:analyze-string select="./@select"
                            regex="(.*)(@|attribute::)([^\]]+)$">
                            <xsl:matching-substring>
                                <xsl:choose>
                                    <!-- if the select has a path before
attribute (i.e path/@attribute) -->
                                    <xsl:when test="regex-group(1)">
                                        <!-- insert a new apply-templates
processing istruction to result tree -->
                                        <myxsl:apply-templates
                                        mode="START-ANCESTOR-FOR-
ATTRIBUTE"

                                        select="{replace(regex-
group(1),'/$','')}">

                                        <myxsl:with-param
                                        name="attributeName">
                                        <xsl:value-of
                                        select="regex-group(3)"/>
                                        </myxsl:with-param>
                                        </myxsl:apply-templates>
                                        <!-- create the xsl:value-of
element it has no children element
                            so no need to process children-->
                                        <xsl:element
name="{name($currentNode)}">

                                        <xsl:copy-of
                                        select="$currentNode/@*"/>
                                        </xsl:element>
                                        <!-- insert a new apply-templates
processing istruction to result tree -->
                                        <myxsl:apply-templates
                                        mode="END-ANCESTOR-FOR-ATTRIBUTE"
                                        select="{replace(regex-
group(1),'/$','')}">

                                        <myxsl:with-param
                                        name="attributeName">
                                        <xsl:value-of
                                        select="regex-group(3)"/>
                                        </myxsl:with-param>
                                        </myxsl:apply-templates>
                                    </xsl:when>
                                    <!-- if the select has NO path before
attribute (i.e @attribute) -->
                                    <xsl:otherwise>
                                        <!-- insert a new apply-templates
processing istruction to result tree -->
                                        <myxsl:apply-templates
                                        mode="START-ANCESTOR-FOR-
ATTRIBUTE"
                                        select="{'.'}">
                                        <myxsl:with-param
                                        name="attributeName">
                                        <xsl:value-of
                                        select="regex-group(3)"/>
                                        </myxsl:with-param>
                                        </myxsl:apply-templates>
                                        <!-- create the xsl:value-of
element it has no children element
                            so no need to process children-->
                                        <xsl:element
name="{name($currentNode)}">
                                        <xsl:copy-of
                                        select="$currentNode/@*"/>
                                        </xsl:element>
                                        <!-- insert a new apply-templates
processing istruction to result tree -->
                                        <myxsl:apply-templates
                                        mode="END-ANCESTOR-FOR-ATTRIBUTE"
                                        select="{'.'}">
                                        <myxsl:with-param
                                        name="attributeName">
                                        <xsl:value-of
                                        select="regex-group(3)"/>
                                        </myxsl:with-param>
                                        </myxsl:apply-templates>
                                    </xsl:otherwise>
                                </xsl:choose>
                            </xsl:matching-substring>
                        </xsl:analyze-string>
                    </xsl:when>
                    <!-- the value of select attribute of xsl:element is
a path to an element node -->
                    <xsl:otherwise>
                        <!-- insert a new apply-templates processing
istruction to result tree -->
                        <myxsl:apply-templates mode="START-ANCESTOR-FOR-
ELEMENT"
                            select="{./@select}"/>
                        <!-- create the xsl:value-of element it has no
children element
                            so no need to process children-->
                        <xsl:element name="{name()}">
                            <xsl:copy-of select="@*"/>
                            <xsl:attribute name="select"
                                    >normalize-space(replace(<xsl:value-
of
                                    select="$currentNode/@select"
                                />,'[&gt;&lt;\p{P}]','
                            '))</xsl:attribute>
                        </xsl:element>
                        <!-- insert a new apply-templates processing
istruction to result tree -->
                        <myxsl:apply-templates mode="END-ANCESTOR-FOR-
ELEMENT"
                            select="{./@select}"/>
                    </xsl:otherwise>
                </xsl:choose>
            </xsl:when>
            <!-- *** if the eleemtn is xsl:text it has only CDATA content
so simply print it out -->
            <xsl:when test="name()='xsl:text'">
                <xsl:value-of
                    select="normalize-space(replace(.,'[&gt;&lt;\p{P}]','
'))"
                />
            </xsl:when>
            <!-- *** if the element name starts with "xsl:" (i.e. other
xsl processing instructions) -->
            <xsl:when test="starts-with(name(),'xsl:')">
                <!-- directly create the same element and process its
children -->
                <xsl:element name="{name()}">
                    <xsl:copy-of select="@*"/>
                    <xsl:apply-templates select="$currentNode/child::*"/>
                </xsl:element>
            </xsl:when>
            <!-- *** if the element is a text element -->
            <xsl:when test="text()">
                <!-- directly send the text to result tree-->
                <xsl:if test="name()!='style' and name()!='script'">
                    <xsl:text> </xsl:text>
                    <xsl:value-of
                        select="normalize-
space(replace(.,'[&gt;&lt;\p{P}]',' '))"/>
                    <xsl:apply-templates select="$currentNode/child::*"/>
                </xsl:if>
            </xsl:when>
            <!-- *** if otherwise (i.e the element is an html tag) -->
            <xsl:otherwise>
                <!-- directly process its children no need to process the
element itself -->
                <xsl:if test="upper-case(name())='IMG'">
                    <xsl:text> </xsl:text>
                    <xsl:value-of
                        select="normalize-
space(replace(./@alt,'[&gt;&lt;\p{P}]',' '))"
                    />
                </xsl:if>
                <xsl:if test="upper-case(name())='META'">
                    <xsl:if
```

```
                        test="upper-case($currentNode/@NAME)='KEYWORDS'
or upper-case($currentNode/@name)='KEYWORDS' or upper-
case($currentNode/@NAME)='DESCRIPTION' or upper-
case($currentNode/@name)='DESCRIPTION'">
                        <xsl:value-of
                            select="normalize-
space(replace(./@CONTENT,'[&gt;&lt;\p{P}]',' '))"/>
                        <xsl:value-of
                            select="normalize-
space(replace(./@content,'[&gt;&lt;\p{P}]',' '))"
                        />
                    </xsl:if>
                </xsl:if>
                <xsl:apply-templates select="$currentNode/child::*"/>
            </xsl:otherwise>
        </xsl:choose>
    </xsl:template>
    <!-- if the element is simply a text directly print it -->
    <xsl:template match="text()">
        <xsl:value-of select="."/>
    </xsl:template>
</xsl:stylesheet>
```

REFERENCES

[1] Soumen Chakrabarti and Byron E. Dom and S. Ravi Kumar and Prabhakar Raghavan and Sridhar Rajagopalan and Andrew Tomkins and David Gibson and Jon Kleinberg, "*Mining the Web's Link Structure",* ACM, Computer Volume 32, Issue 8, 1999.

[2] Hyo-Jung Oh and Sung Hyon Myaeng and Mann-Ho Lee, "*A practical hypertext categorization method using links and incrementally available class information"*, Annual ACM Conference on Research and Development in Information Retrieval, 2000.

[3] Ke Wang and Senqiang Zhou and Shiang Chen Liew "*Building Hierarchical Classifiers Using Class Proximity"*, Proceedings of the 25th International Conference on Very Large Data Bases, 1999.

[4] Mike Perkowitz and Oren Etzioni, "*Adaptive Web Sites: Conceptual Cluster Mining",* Proceedings of the Sixteenth International Joint Conference on Artificial Intelligence, 1999.

[5] David D. Lewis, *"Naive (Bayes) at Forty: The Independence Assumption in Information Retrieval"* Lecture Notes in Computer Science; Vol. 1398, 1998.

[6] Ludovic Denoyer and Patrick Gallinari, "Bayesian network model for semi-structured document classification", Information Processing and Management, Volume 40, Issue 5, 2004

[7] Irina Rish, *"An empirical study of the naive Bayes classifier",* IJCAI 2001 Workshop on Empirical Methods in Artificial Intelligence, 2001.

[8] Andrew McCallum and K. Nigam, *"A comparision of event models for naive bayes text classification"*, AAAI-*98 Workshop on Learning for Text Categorization, 1998*.

[9] Hand, DJ, and Yu, K. *"Idiot's Bayes - not so stupid after all?",* International Statistical Review, Vol 69 part 3, pages 385-399, ISSN 0306 7734, 2001.

[10] Andrej Bratko, and Bogdan Filipi, *"Exploiting Structural Information in Semi-structured Document Classification"*, Proc. 13th International Electrotechnical and Computer Science Conference, 2004

[11] Jeonghee Yi and Neel Sundaresan, *"A classifier for semi-structured documents"*, Proceedings of the sixth ACM SIGKDD international conference on Knowledge discovery and data mining, 2000.

[12] Ludovic Denoyer and Patrick Gallinari, *"Bayesian network model for semi-structured document classification"*, Information Processing and Management, Volume 40, Issue 5, 2004.

[13] Yiming Yang and Xin Liu,*"A re-examination of text categorization methods"*, Proceedings of ACM SIGIR Conference on Research and Development in Information Retrieval (SIGIR'99, pp 42--49), 1999

[14] David D. Lewis and Mark Ringuette*, "A comparison of two learning algorithms for text categorization"*, Proceedings of SDAIR-94, 3rd Annual Symposium on Document Analysis and Information Retrieval, 1994

[15] Weigend, A.S., Weiner, E.D. and Peterson*, "Exploiting Hierarchy on Text Categorization"*, Information Retrieval I(3), 1999

[16] S. Dumais, J. Platt, D. Heckerman, and M. Sahami, "*Inductive learning algorithms and representations for text categorization"*, CIKM, 1998

[17] Dunja Mladenic, *"Turning Yahoo to Automatic Web-Page Classifier"*, European Conference on Artificial Intelligence, 1998

[18] F. Esposto, D. Malerba, L. D. Pace, and P. Leo. *"A machine learning apporach to web mining",* In Proc. Of the 6th Congress of the Italian Association for Artificial Intelligence, 1999

[19] A. Sun and E. Lim and W. Ng, *"Web classification using support vector machine",* Proceedings of the fourth international workshop on Web information and data management. ACM Press, 2002

[20] Arul Prakash Asirvatham, Kranthi Kumar Ravi, *"Web Page Classification based on Document Structure"*, 2001

[21] H.-J. Oh, S. H. Myaeng, and M.-H. Lee, *"A practical hypertext categorization method using links and incrementally available class information",* Proceedings of the 23rd ACM International Conference on Research and Development in Information Retrieval, 2000

[22] Soumen Chakrabarti and Byron E. Dom and Piotr Indyk*, "Enhanced hypertext categorization using hyperlinks"*, Proceedings of {SIGMOD}-98, {ACM} International Conference on Management of Data, 1998

[23] Ron Kohavi, *"A Study of Cross-Validation and Bootstrap for Accuracy Estimation and Model Selection"*, International Joint Conference on Artificial Intelligence (IJCAI), 1995