International Journal of Applied Information Systems (IJAIS) – ISSN : 2249-0868
Foundation of Computer Science FCS, New York, USA
Volume1– No.9, April 2012 – www.ijais.org

Approach for Transforming Monolingual Text Corpus into XML Corpus

Deepak Sharma
Student
Department of Information Technology
Bharati Vidyapeeth College of Engineering & Research,
Pune 411043, India

Prakash.R.Devale
Associate Professor
Department of Information Technology
Bharati Vidyapeeth College of Engineering & Research,
Pune 411043, India

ABSTRACT

In this paper, we are presenting the approach to convert the text based monolingual corpus to Part-Of-Speech tagging using an standard tagging tool in tagged file and then convert tagged file in the XML format as per defined DTD (Document Type Definition). The tagged text document is parsed through the logic to generate the corpus in XML and also, it can be further used for Information Retrieval, Text-To-Speech conversion, Word Sense Disambiguation and also useful for preprocessing step of parsing by providing unique tag to each word which reduces the number of parses.

General Terms
Natural Language Processing

Keywords
Part-Of-Speech tagging, Java XML library, DOM Parser.

1. INTRODUCTION

In corpus linguistics, part-of-speech tagging (POS tagging), also called word-category disambiguation or grammatical tagging, is the process of marking up a word in a text (corpus) as corresponding to a particular part of speech, based on both its definition, as well as its context—i.e. relationship with adjacent and related words in a phrase, sentence, or paragraph. A simplified form of this is commonly taught to school-age children, in the identification of words as nouns, verbs, adjectives, adverbs, etc.

Once performed by hand, POS tagging is now done in the context of computational linguistics, using algorithms which associate discrete terms, as well as hidden parts of speech, in accordance with a set of descriptive tags.

Definition: The process of assigning a part-of-speech or other lexical class marker or tags to each word in a corpus is called Part-Of-Speech**.** Figure 1.1 shows the example of Part-Of-Speech tagging.

In this paper, we have implementing the logic to build the corpus in XML as per user defined DTD for any

2. CLASSIFICATION OF PART-OF-SPEECH TAGGING

The approach for Part-Of-Speech tagging is classified into three types as follows:[1]

2.1 Rule-Based Tagging

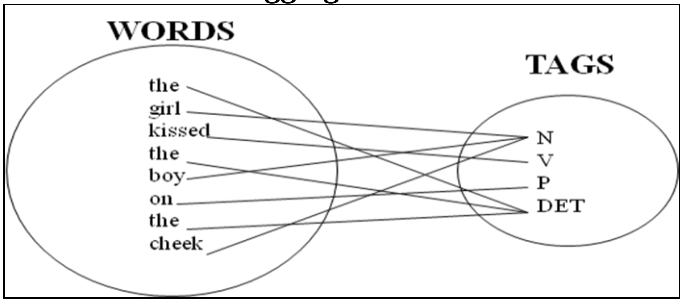


Figure 1.1: Example of Part-Of-Speech Tags.

The basic idea behind in Rule based Tagging is to assign all possible tags to all words. Remove tags according to set of rules of type: if word+1 is an adj, adv, or quantifier and the following is a sentence boundary and word-1 is not a verb like “consider” then eliminate non-adv else eliminate adv. Typically more than 1000 hand-written rules, but may be machine-learned.

2.2 Stochastic Tagging

Based on probability of certain tag occurring given various possibilities. In this approach for pos-tagging, it requires a training corpus. There would be no probabilities for words not in corpus. Also, Training corpus may be different from test corpus. For stochastic tagging, choose most frequent tag in training text for each word.

2.3 Transformation-Based Tagging

It is also as Brill-Tagging. This approach is a combination of Rules-based and Stochastic tagging methodologies as:

a) Like rule-based because rules are used to specify tags in a certain environment.

b) Like stochastic approach because machine learning is used—with tagged corpus as input

The basic idea behind this approach as it Set the most probable tag for each word as a start value. Also, Change tags according to rules of type “if word-1 is a determiner and word is a verb then change the tag to noun” in a specific order. Training is done on tagged corpus by write a set of rule templates. Among the set of rules, find one with highest score then continue from 2 until lowest score threshold is passed. Keep the ordered set of rules, whereas rules make errors that are corrected by later rules as: Tagger labels every word with its most-likely tag. For example: race has the following probabilities in the Brown corpus: P(NN|race) = .98 and P(VB|race)= .02

3. PREPROCESSING TEXT CORPUS

Before converting the part-of-speech tagged text file into XML. We need to follow some steps to achieve the desire result. Prerequisite for tool: Java 1.5 or above must be installed at your computer.

i. Download the standard part-of-speech tagger[2]

For English language from the following url:

http://nlp.stanford.edu/software/tagger.shtml#Download

From here download the "**Download basic English Stanford Tagger version 3.1.0**".

ii. Unzip the tagger "Download basic English Stanford Tagger version 3.1.0"
iii. Prepare your corpus as input.txt as shown in Figure 3.1.
iv. Execute the following command using terminal as follows to tag a file using the pre-trained bidirectional model:

```
java -mx300m -classpath stanford-postagger.jar edu.stanford.nlp.tagger.maxent.MaxentTagger –model models/english-bidirectional-distsim.tagger –textFile input.txt > output.txt
```

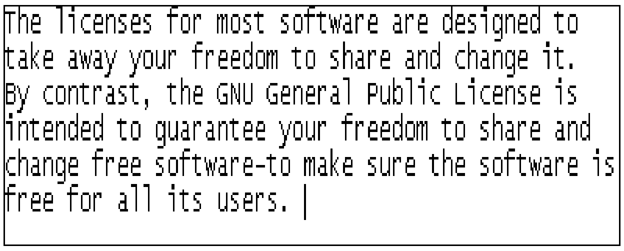
Whereas,

–mx300m: It is used for JVM memory allocation depending on the size of corpus.

–classpath: It is setting up the classpath.

–model: It is used for referencing a directory containing trained POS taggers i.e. english-bidirectional-distsim.tagger.

–textFile: It is a input textFile format, as in this case it is .txt file as shown in Figure 3.1.



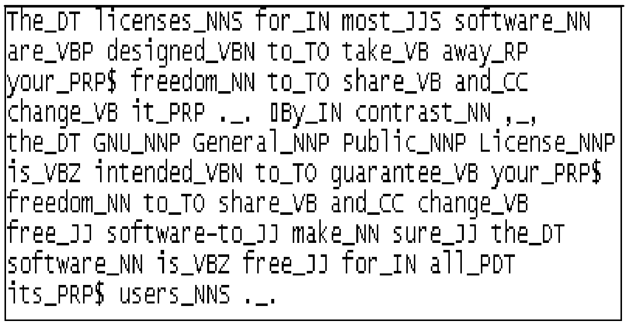
```
The licenses for most software are designed to
take away your freedom to share and change it.
By contrast, the GNU General Public License is
intended to guarantee your freedom to share and
change free software-to make sure the software is
free for all its users. |
```

Figure 3.1: Sample Input Text File (input.txt).

The English Stanford Tagger has three modes: tagging, training, and testing. Tagging allows you to use a pre-trained model (two English models are included) to assign part of speech tags to unlabeled text. Training allows you to save a new model based on a set of tagged data that you provide.

Testing allows you to see how well a tagger performs by tagging labeled data and evaluating the results against the correct tags. Here, we are only referring tagging mode for our experimentation purpose. Figure 3.1 shows the sample input text corpus file, which need to be preprocessed for XML generation. After following the above steps by executing the tagger as mentioned in step iv, we get an output tagged file as shown in Figure 3.2. Now, the output file i.e. output.txt is ready as an input for our XML generation.

The output file consists of the word and part-of-speech of word separated by underscore '_'.



```
The_DT licenses_NNS for_IN most_JJS software_NN
are_VBP designed_VBN to_TO take_VB away_RP
your_PRP$ freedom_NN to_TO share_VB and_CC
change_VB it_PRP ._. By_IN contrast_NN ,_,
the_DT GNU_NNP General_NNP Public_NNP License_NNP
is_VBZ intended_VBN to_TO guarantee_VB your_PRP$
freedom_NN to_TO share_VB and_CC change_VB
free_JJ software-to_JJ make_NN sure_JJ the_DT
software_NN is_VBZ free_JJ for_IN all_PDT
its_PRP$ users_NNS ._.
```

Figure 3.2: Sample Output Text File (ouput.txt).

Now, output.txt file is act as an input for XML Generation as used in section 4.

4. XML GENERATION

Before designing any XML, we need to declare the Document Type Definition (DTD) that defines the legal building blocks of an XML document. Document Type Definition defines the document structure with a list of legal element and attributes.

The DTD(Document Type Definition) as corpus.dtd for the XML as shown below in Figure 4.1

```
<!ELEMENT corpus (s)*>
<!ELEMENT s (w)*>
<!ATTLIST s
   s_id CDATA #IMPLIED
>
<!ELEMENT w EMPTY>
<!ATTLIST w
   surface CDATA #REQUIRED
   pos CDATA #IMPLIED
>
```

Figure 4.1: DTD for XML Generation(corpus.dtd).

Whereas,

surface: Surface form of a word, like each word in the sentence "He went to the streets"

pos: Morphosyntactic category of a word, like each class name (N-Noun, V-Verb, etc.) in the sentence "PP V P DT N"

We have been used the DOM Parser API for XML generation. DOM Parser is a hierarchy-based parser that creates an object model of the entire XML document, and hands that model to us to work with. DOM Parser has tree-based API. It is easy to navigate the tree model. Entire tree is loaded once into the memory. It allows random access to XML document and consists of rich set of API's. [3]

DOM stands for Document Object Model. Many applications want a tree representation of an XML document instead of a series of callbacks from the parser. One API that provides this is DOM. DOM is a standard produced by the W3C. As of this writing, the current version is DOM Level 1, but Level 2 is a Proposed REC and should soon be a full REC.[4]

DOM based parsers load the entire XML stream into memory, creating a hierarchical object that is referenced within the application logic. [5]

From the section 3, we got a preprocessed tagged file as output.txt, which can be used as an input file for the below logic for xml generation.

```
import java.io.BufferedReader;
import java.io.FileNotFoundException;
import java.io.FileReader;
import java.io.IOException;
import java.util.Scanner;
import javax.xml.parsers.DocumentBuilder;
import javax.xml.parsers.DocumentBuilderFactory;
import javax.xml.parsers.ParserConfigurationException;
import javax.xml.transform.OutputKeys;
import javax.xml.transform.Transformer;
import javax.xml.transform.TransformerException;
import javax.xml.transform.TransformerFactory;
import javax.xml.transform.dom.DOMSource;
import javax.xml.transform.stream.StreamResult;
import org.w3c.dom.Attr;
import org.w3c.dom.DOMException;
import org.w3c.dom.Document;
import org.w3c.dom.Element;
public class Text2XMLConv {
    public static void main(String[] args) {
        new Text2XMLConv().doit();
    }
    public void doit(){
        try {
            int iSenCount=0;
            int iLoop;
            char ch;
            Scanner sn = null;
            String str;
            BufferedReader in=null;
            try {
                in = new BufferedReader(new FileReader("[Path of input tagged  file]/output.txt"));
             }
            catch (FileNotFoundException e) {
                // TODO Auto-generated catch block
                e.printStackTrace();
            }
            DocumentBuilderFactory docFactory = DocumentBuilderFactory.newInstance();
            DocumentBuilder docBuilder = docFactory.newDocumentBuilder();
           // root elements
           Document doc =  docBuilder.newDocument();
           Element rootElement = doc.createElement("corpus");
           doc.appendChild(rootElement);
           try {
               while ((str = in.readLine()) != null) {
                   Element sen =  doc.createElement("s");
                   rootElement.appendChild(sen);
                   Attr attr = doc.createAttribute("s_id");
                   attr.setValue(""+iSenCount);
                   sen.setAttributeNode(attr);
                  String tokens[]=str.split(" ");
```

```
                        for(iLoop=0;iLoop<tokens.length;iLoop++){
                              Attr attrs=null;
                           Attr attrp=null;
                           Element word = doc.createElement("w");
                           sen.appendChild(word);
                           attrs = doc.createAttribute("surface");
                           attrp = doc.createAttribute("pos");
                           if(!"._.".equals(tokens[iLoop]) && !",_,".equals(tokens[iLoop])){
                                  sn = new Scanner(tokens[iLoop]).useDelimiter("_");
                                  while(sn.hasNext()){
                                         attrs.setValue(sn.next());
                                         attrp.setValue(sn.next());
                                  }
                           }
                           else{
                                ch= tokens[iLoop].charAt(0);
                                attrs.setValue(""+ch);
                                attrp.setValue("PCT");
                           }
                     word.setAttributeNode(attrp);
                     word.setAttributeNode(attrs);
            }
                     iSenCount++;

         }
      } catch (DOMException e) {
    / TODO Auto-generated catch block
           e.printStackTrace();
      } catch (IOException e) {
    / TODO Auto-generated catch block
           e.printStackTrace();
    }
  // write the content into xml file
        TransformerFactory transformerFactory = TransformerFactory.newInstance();
        Transformer transformer = transformerFactory.newTransformer();
        transformer.setOutputProperty(OutputKeys.ENCODING, "UTF-8");//ISO-8859-1
        transformer.setOutputProperty(OutputKeys.DOCTYPE_SYSTEM,"corpus.dtd");
        transformer.setOutputProperty("{http://xml.apache.org/xslt}indent-amount", "4");
        transformer.setOutputProperty(OutputKeys.OMIT_XML_DECLARATION,"yes");
        transformer.setOutputProperty(OutputKeys.INDENT, "yes");
        DOMSource source = new DOMSource(doc);
        StreamResult out = new StreamResult("[Path of output xml file]/corpus.xml");
        transformer.transform(source, out);
        System.out.println("File saved!");
 } catch (ParserConfigurationException pce) {
     pce.printStackTrace();
 } catch (TransformerException tfe) {
                     tfe.printStackTrace();
 }
}
}
```

Figure 4.2: Java Code for XML Generation

On successful execution of code in Figure 4.2, we would get corpus.xml as shown below in Figure 4.3.

```
DOCTYPE corpus (View Source for full doctype...)>
-<corpus>
  - <s s_id="0">
        <w pos="DT" surface="The" />
        <w pos="NNS" surface="licenses" />
        <w pos="IN" surface="for" />
        <w pos="JJS" surface
        <w pos="NN" surface="software" />
        <w pos="VBP" surface="are" />
        <w pos="VBN" surface="designed" />
        <w pos="TO" surface="to" />
        <w pos="VB" surface="take" />
        <w pos="RP" surface="away" />
        <w pos="PRP$" surface="your" />
        <w pos="NN" surface="freedom" />
        <w pos="TO" surface="to" />
        <w pos="VB" surface="share" />
        <w pos="CC" surface="and" />
        <w pos="VB" surface="change" />
        <w pos="PRP" surface="it" />
        <w pos="PCT" surface="." />
    </s>
</corpus>
```

Figure 4.3: Resultant XML file as corpus.xml

5. CONCLUSION

We are presenting an approach for generating corpus in xml format. It can be further used for Information Retrieval, Text-To-Speech conversion, Word Sense Disambiguation and also useful for preprocessing step of parsing by providing unique tag to each word which reduces the number of parses. Also, we are designing a toolkit for multiword extraction for a monolingual corpus. This paper is simply an initial step for the multiword extraction toolkit[6].

6. REFERENCES

[1] Andrew MacKinlay and Timothy Baldwin, “POS Tagging with a More Informative Tagset”, at Proceedings of the Australasian Language Technology Workshop 2005, pages 40–48, Sydney, Australia, December 2005.

[2] Christopher D. Manning, Part-Of-Speech Tagging From 97% To 100%: Is It Time For Some Linguistics?, in CICLing2011.

[3] Su Cheng Haw, G. S. V. Radha Krishna Rao,,”A Comparative Study and Benchmarking on XML Parsers”, *Faculty of Information Technology, Multimedia University, 63100 Cyberjaya.*

[4] Edwin Goei, Software Engineer, Sun Microsystems,” Java and XML Parsing Using Standard APIs”, September 11, 2000

[5] Nishchal Bhalla, Sahba Kazerooni,”Web Services Vulnerabilities”, at Security Compass Inc 2007.

[6] C. Ramisch, A. Villavicencio, C. Boitet, Mwetoolkit: A Framework For Multiword Expression Identification”, in: Proceedings of the Seventh International Conference on Language Resources and Evaluation (LREC 2010), Valetta, Malta, May 2010