# XGTagger, a generic interface for analysing XML content

Xavier Tannier
Aude Garnier

## Contents

## 1 Introduction

XGTagger is a generic interface dealing with text contained in XML documents. It has been designed for part-of-speech tagging especially, but can be used for other applications, among which lexical operations, syntactic analyses and automatic translation. Section 2 explains what is part-of-speech tagging and why this operation is made more difficult when working with XML documents. Some other issues raised by XML content analysing are detailed in [9]. In section 3, we give the definition of a *"reading context"* and describe a categorization of XML tags proposed by [5]. The rest of this report is devoted to XGTagger[1].

## 2 POS tagging in XML documents

### 2.1 Introduction

#### 2.1.1 POS tagging

A part-of-speech (POS), or word class, is the role played by a word in the sentence (*e.g.:* noun, verb, adjective...). POS tagging is the process of marking up words in a text with their corresponding roles.

The field of automatic part-of-speech (POS) tagging has been extensively studied [2, 4, 1, 3, 8], particularly in the case of European languages, with accuracy rates of about 95%. Many researchers consider this problem as solved from written language.

#### 2.1.2 XML

XML (eXtensible Markup Language [11]) documents provide textual information with some structural and semantic annotations represented by element tags and attributes. Readers that would not be familiar with XML can find some basic terminologic indications in appendix A.

### 2.2 POS tagging in XML documents

Here we focus on a *document-centric* view on XML documents. In this view, mark-up serves for giving information about logical structure and/or about form of a traditional document. This is the case of all texts intended for human people, such as manuals, books, articles or static web pages. This view is opposed to *data-centric* view, used for more database-oriented applications (flight schedules, catalogues, etc.), where POS tagging is pointless.

XML is more and more widely used to store and exchange information, and raises some new problems for POS tagging. This is due to the fact that an XML document does not necessarily respect the linearity of the text.

#### 2.2.1 Context loss

Whatever the POS tagging technique is, a tagger uses the context of a word (the terms around it) to disambiguate its lexical category. That is why the words must be given to the system in the order a human could read them.

This is a problem in example 1, where a gloss is inserted in the middle of a sentence.

(1) <book *title="Learn English"*>

...
<sentence>The trash can <gloss>A trash can is a place where you can put anything you don't want any more</gloss> lies near the fridge</sentence>
...

</book>

[1] `http://www.emse.fr/~tannier/en/xgtagger.html`

In this case the ambiguous words are *"can"* (either a modal verb or a noun) and *"lies"* (a verb or a noun). The following is the output of TreeTagger [7] when the gloss is removed (1.a) and not (1.b). We see that the insertion of the `'gloss'` element prevents the tagger from attributing the correct part-of-speech (*NN(S)* means "noun", *VVZ* stands for "verb" and *MD* for "modal verb". A more complete description of Penn Tagset can be found in [6].

| | | | | | | | |
|---|---|---|---|---|---|---|---|
| (1.a) | *The* | *trash* | ***can*** | ***lies*** | *near* | *the* | *fridge* |
| | DT | NN | **NN** | **VVZ** | IN | DT | NN |

| | | | | | | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| (1.b) | *The* | *trash* | ***can*** | *this* | *is* | *a* | *place* | *where* | … *any* | *more* | ***lies*** | *near* | *the* | *fridge* |
| | DT | NN | **MD** | DT | VBZ | DT | NN | WRB | … DT | JJR | **NNS** | IN | DT | NN |

Example 2 shows that non-overlapping elements can raise the same kind of issue.

(2) <news>

<item>US and Iraqi forces detain a new suspect</item>
<item>Girl rescued from trash bin helps police arrest attacker</item>

</news>

Each `'item'` element taken separately leads to a correct analysis (2.a and 2.b). But laying both elements together causes an error on *"suspect"* (2.c). This error can occur when no punctuation ends the element.

| | | | | |
|---|---|---|---|---|
| (2.a) | … *a* | *new* | ***suspect*** | |
| | … DT | JJ | **NN** | |

| | | | | | |
|---|---|---|---|---|---|
| (2.b) | *Girl* | *rescued* | *from* | *trash* | … |
| | NN | VVD | IN | NN | |

| | | | | | | | | |
|---|---|---|---|---|---|---|---|---|
| (2.c) | … *a* | *new* | ***suspect*** | *Girl* | *rescued* | *from* | *trash* | … |
| | … DT | JJ | **JJ** | NN | VVD | IN | NN | … |

### 2.2.2 Cut words

In some XML collections, words are quite frequently cut by certain elements. This is the case of the `'comment'` in example 3, and of small capitals (`'sc'`) in example 4. In these both cases a special treatment has to be done in order to retrieve the entire words.

(3) <oral_transcription>

I heard the news today about United States elec<comment>a door snaps</comment>tions.

</oral_transcription>

(4) <title>

R<sc>omeo</sc> and J<sc>uliet</sc>

</title>

## 3 Three types of XML markup

A characteristic of XML documents is that they contain several consecutive, potentially overlapping *"reading contexts"* [9]. A *reading context* is a small part of text, syntactically and semantically self-sufficient, that a person can read in a go, without any interruption. Certain types of tags interrupt a reading context, some others do not.

An Information Retrieval oriented division of tags has been proposed by [5], in order to identify different categories that it would be important to distinguish within the framework of XML document retrieval. The original idea was to allow different treatments while searching for a pattern (sequence of characters).

### 3.1 Hard, soft and jump tags

The three different classes are the following:

- *"Hard" tags* are the most frequent, they interrupt the "linearity" of a text, they generally contribute to the structuring of the document. Examples of this type are titles, chapters, paragraphs. Tags `'news'` and `'item'` are both "hard" in the following example (and in example 2):

  (5) <news>

      <item>A new study about evolution of
      tourism in the United States</item>
      <item>Elections in Ukraine: the Central
      Commission has published the official
      results</item>

  </news>

  A hard element contains an entire reading context.

- *"Soft" tags* identify significant parts of a text, like quotations, appearance effects, but become "transparent" while reading the text. A soft element lies within the current reading context and does not interrupt it. This is the case of tags `'bold'`, `'italics'`, `'underlined'` and `'sc'` (small capitals) in examples 4, 6.a, 6.b, and 6.c.

  (6) a. <par>

      <bold>United States elections</bold>
      are administered at the state and local levels.

  </par>

  b. <title>

      Noam Chomski's comments about
      <italics>United States</italics>
      <underlined>elections</underlined>.

  </title>

  c. <title>

      U<sc>nited</sc> S<sc>tates</sc>
      E<sc>lections</sc>.

  </title>

- *"Jump" tags* are used to represent particular elements, like margin notes, references to bibliography, or glosses. They are detached from the surrounding text. A jump element is inserted into an existing reading context and composes a new one. Elements `'comment'` and `'footnote'` in the following examples are "jump" elements. This is also the case of element `'gloss'` in example 1.

(7) a. <oral_transcription>

I heard the news today about United
States elec<comment>a door
snaps</comment>tions.

</oral_transcription>

b. <paragraph>

The 2004 United States<footnote>See
an article about the United States of
America on page 142</footnote> elections caused less controversy than in 2000.

</paragraph>

c. <abstract>

This document deals with 1995 and 2002 Jacques Chirac <footnote>J. Chirac, the french president, is, by the way, not a really good friend of the president of the United States</footnote> elections.

</abstract>

## 4 XGTagger

XGTagger is a generic interface dealing with reading contexts.

XGTagger allows the user to tag the textual content of an XML document with the system of her own choosing. It parses the document and rebuilds the *reading contexts*, by using the textual content of tags and a list of DTD-dependant *soft* and *jump* tags (given by the user[2]). Reading contexts (text only) are sent to the part-of-speech tagger (or any other appropriate system), and the output of this tagger is used to compose a new XML document (see figure 1).

There is not any loss of information between the initial XML document and the final output. The operation is totally reversible (excepted in some cases for blank characters), and the input can be found back with a simple stylesheet[3].

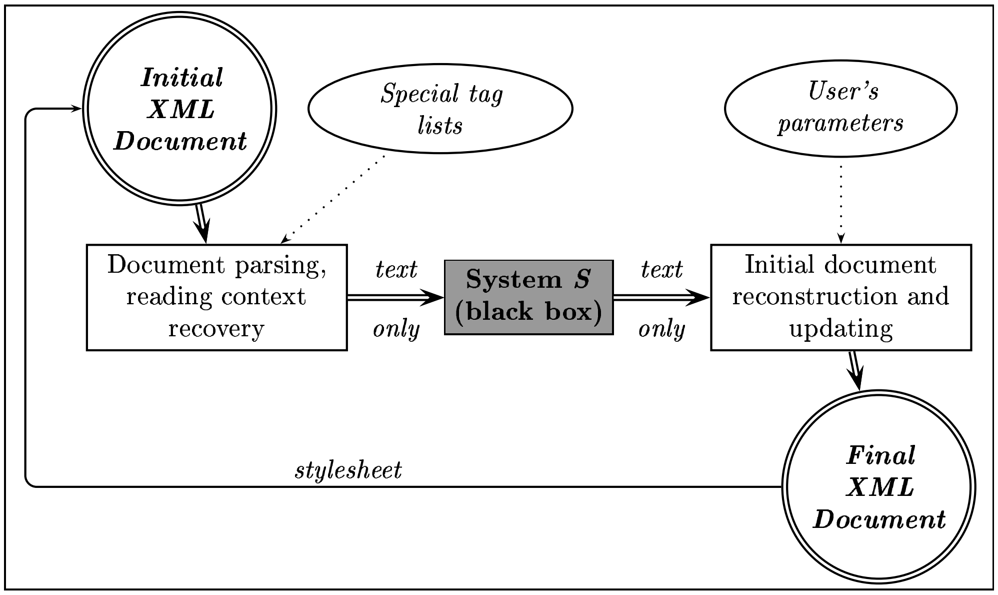


Figure 1: XGTagger general fonctioning scheme.

[2]We showed in [9] that it was possible in some cases to detect automatically tag classes, but this is outside the scope of the presented system.

[3]XGTagger can be freely downloaded: `http://www.emse.fr/~tannier/en/xgtagger.html`.

### 4.1 Short example

- Input: <sentence>The <bold>cat</bold> jumps on the table</sentence>
- Output obtained with TreeTagger [8, 7] and the default configuration file given in XGTagger distribution:

```
<sentence>

    <w id=”1” pos=”DT” lem=”the”>The</w>
    <bold>

        <w id=”2” pos=”NN” lem=”cat”>cat</w>

    </bold>
    <w id=”3” pos=”VVZ” lem=”jump”>jumps</w>
    <w id=”4” pos=”IN” lem=”on”>on</w>
    <w id=”5” pos=”DT” lem=”the”>the</w>
    <w id=”6” pos=”NN” lem=”table”>table</w>

</sentence>
```

A more complete example, with jump tags and words cut by XML elements, is presented in appendix B.

In some existing POS taggers, it is necessary to remove certain punctuation from words. This is not done by XGTagger. But as it accepts any command, this command can contain a punctuation preprocessing.

### 4.2 Constraints

The main constraints that our system had to respect were:

- Reading contexts preservations (see above);
- Reversibility;
- As far as possible, genericity;
- Only one call to the user's system by document. Indeed part-of-speech taggers have often a very long initialisation phase, and we wanted this process to be run only once.

### 4.3 What does XGTagger exactly?

XGTagger parses the XML documents twice, a first time before running the users's system and a second time after. Documents are parsed top-down and depth-first.

The first parsing consists in collecting document text, in a way that preserve reading contexts. This text (called $T$ below) is intended to be the input of the user's system (named $S$). $T$ is constituted as follows:

1. Each textual node (text content of an XML element) of the initial document gets a unique identifier.
2. 
   - If the textual node is contained by a hard tag, two dots are inserted before and after its text, which is added to $T$. A dot is a strong punctuation that will prevent the part-of-speech tagger to mix two reading contexts.
   - If the textual node is contained by a jump tag, two dots are also added before and after the text, for the same reason. In addition, this text will be treated after all “non-jump” tags in the element, in order to avoid overlapping reading contexts.
   - If the textual node is contained by a soft tag, it can be part of a bigger reading context; the text is simply appended to $T$.

3. For each textual node identifier, two values are kept in a dictionnary (hash table): the positions of the first and of the last characters corresponding to this node in the built text $T$. Blank characters are not considered while counting.

These actions are illustrated by steps 5 and 6 of the example given in appendix B.

The text $T$ is then written into a file, and the user's system command line is called (see manual [10]). The only constraint for the output of this system is that one of its fields must contain all non-blank characters of the original text $T$. Some spaces can be added or removed[4].

The XML document is then parsed a second time, in exactly the same order than the first time. This precaution, as well as the hash table containing information about the position of the text, allows to find back which element corresponds to which fields in $S$ output. Each word is replaced by a tag containing the same text. This new tag has also some additional attributes (depending on user's preferences and $S$ output). This is exemplified is steps 8 and 9 in appendix B.

In order to respect the constraint of reversibility, jump and soft tags are kept at the same position than in original document. If some words were cut, they remain cut in the final output. However it is possible to know that two separated elements are parts of the same word; this information is given by an identifier attribute (see for example identifiers 2 and 4 in B.9). In the same way, reading contexts can be found back with identifiers (see elements 12 and 13 in B.9).

## 4.4 Genericity

As explained above, the only big constraint imposed to user's system in that it must return (among other data) the initial text given in input. Beyond that, the output format is free. Field and word separators can be specified by the user, as well as the semantics of each information contained by different fields, and the command that should be called.

For example, TreeTagger's output consists of one line per analysed word (word separator is '`\n`'), and three fields per line (field separator '`\t`'), corresponding to the initial word, the part-of-speech and the lemma (this is specified by program parameters). On the other hand, Brill Tagger [1] just adds a slash '/' and the part-of-speech to each input word. Report to XGTagger manual to see how to handle these both formats.

## 5 Other uses of XGTagger

XGTagger can also be used for any other applications. This requires a good knowledge of the functionning of the program. The user can refer to the following suggestions, to the long example given in appendix B or to the program sources. The manual [10] also gives other examples of lexical enrichment.

*N.B.:* Recall that an important constraint of XGTagger is that at least one field of the user's system must contain the initial text (blank characters excepted).

## 5.1 POS tagging upgrading: locution handling

If the system $S$ is able to detect locutions, XGTagger can deal with that with a special option (called `special separator`). With this option the user can specify that a sequence of characters represents a separation between words.

- Let's take the following XML element:
  <sentence>I did it in order to clarify matters</sentence>

- XGTagger will input the following text into the system:
  `I did it in order to clarify matters`

[4]This is done by some taggers, that separate punctuation or apostrophes.

- With the special separator '///', $S$ can return:

```
I                PP
did              VVD
it               PP
in///order///to  LOC
clarify          VV
matters          NNS
```

- With options `-i -w 1 -1 word -2 pos -d "///"`, XGTagger final output is:
  <sentence>

      <w id="1" pos="PP" word="I">**I**</w>
      <w id="2" pos="VVD" word="do">**did**</w>
      <w id="3" pos="PP" word="it">**it**</w>
      <w id="4" pos="LOC" word="in///order///to">**in**</w>
      <w id="4" pos="LOC" word="in///order///to">**order**</w>
      <w id="4" pos="LOC" word="in///order///to">**to**</w>
      <w id="5" pos="VV" word="clarify">**clarify**</w>
      <w id="6" pos="NNS" word="matter">**matters**</w>

  </sentence>

## 5.2 Syntactic analysis

With the same `special separator` option, a syntactic analysis can be performed. Suppose that $S$ groups together noun phrases of the form "`NOUN PREPOSITION NOUN`".

- For the following XML element:
  <english_sentence>He has a taste<gloss>Taste: preference, a strong liking</gloss> for danger</english_sentence>

- ... XGTagger will give this text into the system (considering that `'gloss'` is a jump tag):
  `He has a taste for danger .  Taste:  preference, a strong liking .`

- $S$ can perform a simple syntactic analysis and return, by example:
  `He has a taste_for_danger/NP . Taste:  preference, a strong liking .`

- With XGTagger options `-i -w 1 -2 pos -f "/" -d " " -e "_"`, the final output is:
  <english_sentence>

      <w id="1">**He**</w>
      <w id="2">**has**</w>
      <w id="3">**a**</w>
      <w id="4" pos="NP">**taste**</w>
      <gloss>
          <w id="6">**Taste:**</w>
          <w id="7">**preference,**</w>
          ...
          <w id="10">**liking**</w>
      </gloss>
      <w id="4" pos="NP">**for**</w>
      <w id="4" pos="NP">**danger**</w>

  </english_sentence>

## 5.3 Lexical enrichment

The user's system can also return any information about words. For example, a translation of each noun:

- XML Input:
  <sentence>I had a conversation with my brother</sentence>
- *S* output (suggestion):

```
I
had
a
conversation/entretien, conversation/Gespräch
with
my
brother/frère/Bruder
```

- Options: `-w 1 -2 french -3 german -f` "/"; Output:
  <sentence>

  <w>**I**</w>
  <w>**had**</w>
  <w>**a**</w>
  <w *french="entretien, conversation" german="Gespräch"*>**conversation**</w>
  <w>**with**</w>
  <w>**my**</w>
  <w *french="frère" german="Bruder"*>**brother**</w>

  </sentence>

## 5.4 Reading Contexts finding

Finally, the *S* can just repeat the input text (possibly with a simple separation of punctuation). The result is that words are enclosed between tags, reading contexts are brought together (by ids) and cut words are reassembled. This operation can be a first step before indexing XML documents[5] or operating researchs taking logical proximity [9] into account.

- XML Input:
  <title>U<sc>nited</sc> S<sc>tates</sc> E<sc>lections</sc></title>
- *S* output (same as the input):
  `United States Elections`
- Possible final output:
  <title>

  <w *id="1" rc="United"*>**U**</w>
  <sc><w *id="1" rc="United"*>**nited**</w></sc>
  <w *id="2" rc="States"*>**S**</w>
  <sc><w *id="2" rc="States"*>**tates**</w></sc>
  <w *id="3" rc="Elections"*>**E**</w>
  <sc><w *id="3" rc="Elections"*>**lections**</w></sc>

  </title>

[5]An option of XGTagger adds the path of each element as one of its argument.

## References

[1] E. Brill. A Simple Rule-Based Part of Speech Tagger. pages 152–155.

[2] K. W. Church. A stochastic parts program and noun phrase parser for unrestricted text. In *Proceedings of the second conference on Applied natural language processing*, pages 136–143, Austin, Texas, USA, 1988. Association for Computational Linguistics, Morristown, NJ, USA.

[3] D. Cutting, J. Kupiec, J. Pedersen, and P. Sibun. A practical part-of-speech tagger. pages 133–140.

[4] S. J. DeRose. Grammatical category disambiguation by statistical optimization. *Computational Linguistics*, 14(1):31–39, 1988.

[5] L. Lini, D. Lombardini, M. Paoli, D. Colazzo, and C. Sartiani. XTReSy: A Text Retrieval System for XML documents. In D. Buzzetti, H. Short, and G. Pancalddella, editors, *Augmenting Comprehension: Digital Tools for the History of Ideas*. Office for Humanities Communication Publications, King's College, London, 2001.

[6] B. S. Mitchell P. Marcus and M. A. Marcinkiewicz. Building a Large Annotated Corpus of English: The Penn Treebank. *Computational Linguistics (Special Issue on Using Large Corpora)*, 19(2):313–330, June 1993.

[7] H. Schmid. TreeTagger - a language independent part-of-speech tagger. http://www.ims.uni-stuttgart.de/projekte/corplex/TreeTagger/ DecisionTreeTagger.html.

[8] H. Schmid. Probabilistic Part-of-Speech Tagging Using Decision Trees. In *International Conference on New Methods in Language Processing*, Sept. 1994.

[9] X. Tannier. Dealing with XML structure through "Reading Contexts". Technical Report 2005-400-007, Ecole Nationale Supérieure des Mines de Saint-Etienne, Apr. 2005.

[10] X. Tannier. XGTagger User Manual. http://www.emse.fr/~tannier/XGTagger/Manual/, June 2005.

[11] Extensible Markup Language (XML). World Wide Web Consortium (W3C) Recommandation, 2004. http://www.w3.org/TR/2004/REC-xml-20040204/.

## A XML terminology

The following is some useful and basic terminologic indications that are important to understand the article:

(8) <book>

<title>Le Petit Prince</title>
<author>Antoine de St-Exupery</author>
<chapter *number="1"*>

When I was six, . . .

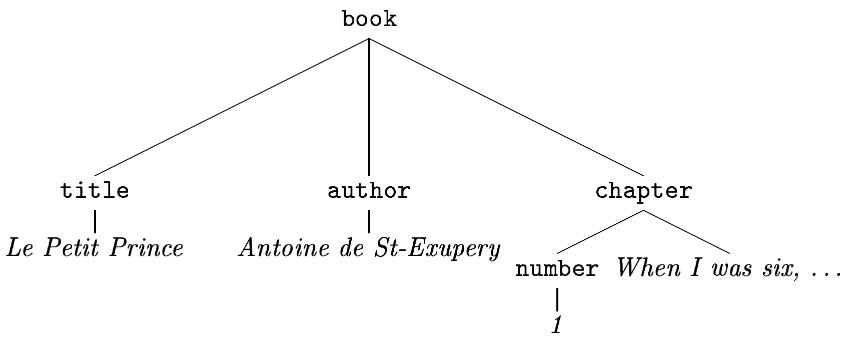
</chapter>

</book>

In this example:

- "book", "title", "author" and "chapters" are *tag names*;
- `<book>` and `</books>` are *tags* (respectively start and end tags);
- start/end tags and their content (between them) constitute an *element*.
- The document can be represented by a tree (example 8').
- In this tree representation, elements are *nodes* and text parts (leaf nodes) are *textual nodes*.
- Finally, the Document Type Definition (DTD) is a document defining the elements that can be used in the XML file as well as their structure (imbrication, number, sequences, etc.).

(8')



## B A long example

This is an example of an XML document tagged by TreeTagger.

1. Input:
<article>

<title>Visit I<sc>stanbul</sc> and M<sc>armara</sc> Region</title>
<par>

This former capital of three empires<footnote>Istanbul has successively been the capital of Roman, Byzantine and Ottoman empires</footnote> is now the capital of <bold>Turkey</bold>
. . .

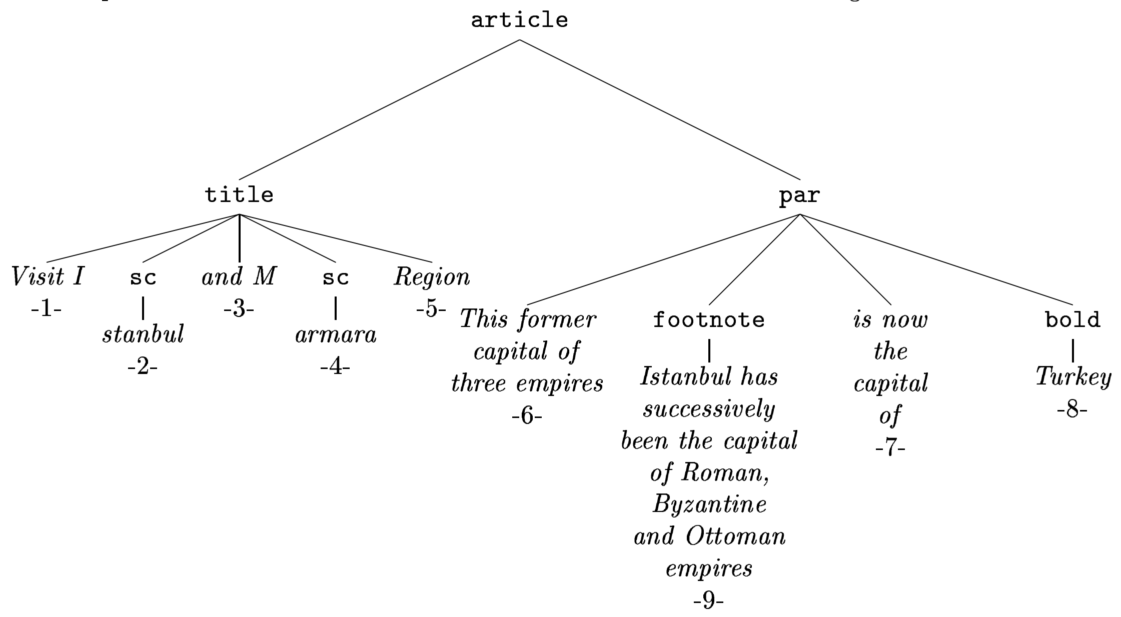
</par>

</article>

2. Options in configuration file:

```
WORD_TAG_NAME = w
WORD_FIELD = 1
$2 = pos
$3 = lem
FIELD_DELIMITERS = "\t"
WORD_DELIMITERS = "\n"
JUMP_TAG_FILE = jump_tags.txt
SOFT_TAG_FILE = soft_tags.txt
WORD_PATH = false
TAGS_PATH = false
LOG = false
READING_CONTEXT_LOG = false
ID = true
OUTPUT_FILE = result.xml
COMMAND = "tree-tagger-english "
```

3. `jump_tags.txt` contains `footnote`.
4. `soft_tags.txt` contains `bold`, `sc` (small capitals).
5. Tree representation of the XML document and textual nodes numbering:


article
title
par
Visit I
-1-
sc
stanbul
-2-
and M
-3-
sc
armara
-4-
Region
-5-
This former capital of three empires
-6-
footnote
Istanbul has successively been the capital of Roman, Byzantine and Ottoman empires
-9-
is now the capital of
-7-
bold
Turkey
-8-

6. Information kept about each textual node (recall that blank characters are not counted):

| Tag number | Begin position | End position | Text |
|---|---|---|---|
| 1 | 1 | 6 | Visit I |
| 2 | 7 | 14 | stanbul |
| 3 | 15 | 18 | and M |
| 4 | 19 | 24 | armara |
| 5 | 25 | 31 | Region |
| - | - | - | . |
| 6 | 33 | 63 | This former capital of three empires |
| 7 | 64 | 80 | is now the capital of |
| 8 | 81 | 86 | Turkey |
| - | - | - | . |
| 9 | 88 | 158 | Istanbul has successively been the capital of Roman, Byzantine and Ottoman empires |

7. Text given to TreeTagger by XGTagger:
```
Visit Istanbul and Marmara Region .  This former capital of three
empires is now the capital of Turkey .  Istanbul has successively been
the capital of Roman, Byzantine and Ottoman empires
```

8. TreeTagger output (presented in 2 columns):

| | | | | | |
|---|---|---|---|---|---|
| Visit | VV | visit | of | IN | of |
| Istanbul | NP | Istanbul | Turkey | NP | Turkey |
| and | CC | and | . | SENT | . |
| Marmara | NP | Marmara | Istanbul | NP | Istanbul |
| Region | NN | region | has | VHZ | have |
| . | SENT | . | successively | RB | successively |
| This | DT | this | been | VBN | be |
| former | JJ | former | the | DT | the |
| capital | NN | capital | capital | NN | capital |
| of | IN | of | of | IN | of |
| three | CD | three | Roman | NP | Roman |
| empires | NNS | empire | , | , | , |
| is | VBZ | be | Byzantine | JJ | Byzantine |
| now | RB | now | and | CC | and |
| the | DT | the | Ottoman | NP | Ottoman |
| capital | NN | capital | empires | NNS | empire |

9. Final output:
<article>

    <title>
        <w *id="1" pos="VV" lem="visit"*>**Visit**</w>
        <w *id="2" pos="NP" lem="Istanbul"*>**I**</w>
        <sc>
            <w *id="2" pos="NP" lem="Istanbul"*>**stanbul**</w>
        </sc>
        <w *id="3" pos="CC" lem="and"*>**and**</w>
        <w *id="4" pos="NP" lem="Marmara"*>**M**</w>
        <sc>
            <w *id="4" pos="NP" lem="Marmara"*>**armara**</w>
        </sc>
        <w *id="5" pos="NN" lem="region"*>**region**</w>
    </title>
    <par>

```
    <w  id="7" pos="DT" lem="this">This</w>
    <w  id="8" pos="JJ" lem="former">former</w>
    <w  id="9" pos="NN" lem="capital">capital</w>
    <w  id="10" pos="IN" lem="of">of</w>
    <w  id="11" pos="CD" lem="three">three</w>
    <w  id="12" pos="NNS" lem="empire">empires</w>
    <footnote>
      <w  id="20" pos="NP" lem="Istanbul">Istanbul</w>
      <w  id="21" pos="VHZ" lem="have">has</w>
      <w  id="22" pos="RB" lem="successively">successively</w>
      ...
      <w  id="31" pos="NP" lem="Ottoman">Ottoman</w>
      <w  id="32" pos="NNS" lem="empire">empires</w>
    </footnote>
    <w  id="13" pos="VBZ" lem="be">is</w>
    <w  id="14" pos="RB" lem="now">now</w>
    <w  id="15" pos="DT" lem="the">the</w>
    <w  id="16" pos="NN" lem="capital">capital</w>
    <w  id="17" pos="IN" lem="of">of</w>
    <bold>
      <w  id="18" pos="NP" lem="Turkey">Turkey</w>
    </bold>
  </par>
</article>
```