Tokenizing, Tagging and Lemmatizing free running texts (TTL)

Table of Contents

1 BASIC INFORMATION

1.1 Tool name

The tool is called **TTL** which is the short for "Tokenizing, Tagging and Lemmatizing free running texts" in Romanian, English and French.

1.2 Overview and purpose of the tool

TTL performs sentence splitting, tokenization, POS tagging, lemmatization and shallow parsing (chunking) on Romanian, English and French texts. It can be extended to support other languages provided the necessary resources exist: POS tagged corpora and lexicons. It is written in Perl and it is encapsulated as a SOAP web service which is WSDL described at http://ws.racai.ro/ttlws.wsdl. The version number of TTL is currently 8.5.

1.3 A short description of the algorithm

TTL (Ion, 2007) is a text preprocessing module developed in Perl. Its functions are: Named Entity Recognition (by means of regular expressions defined over sequences of characters), sentence splitting, tokenization, POS tagging, lemmatization and chunking.

The NER function is included as a preprocessing stage to sentence splitting because end of sentence markers may constitute parts of an NE string (i.e. a period may be a part of an abbreviation). POS tagging is achieved through the HMM tagging technology.

The POS tagger of TTL follows the description of HMM tagger given in (Brants, 2000) but it extends it in several ways allowing for tiered tagging, for a more accurate processing of unknown words and also for tagging of named entities (which are practically labeled by the NER module before actual POS tagging). The TTL's tagset is the MSD[1] with its smaller superset CTAG. TTL tagging methodology follows the tiered tagging approach (Tufiş, 1999) where MSDs are recovered from an initial CTAG annotation.

Lemmatization is achieved after POS tagging by lexicon lookup (in general, a word form and its POS tag uniquely identify the lemma). In the case of out-of-lexicon word forms the lemmatization is performed by a statistical module which automatically learns normalization rules from the existing lexical stock (for details see (Ion, 2007)).

Chunking is implemented with regular expressions over sequences of POS tags. It is not recursive and it does not perform attachments (PPs to NPs for instance).

[1] http://nl.ijs.si/ME/V3/msd/html/

2 TECHNICAL INFORMATION

2.1 Software dependencies and system requirements

This kit contains Perl wrappers to the TTL web service basic operations: tokenization (which includes sentence splitting), POS tagging, lemmatization and chunking. To run these scripts one must install the following:

- `Perl`, a recent version from http://www.perl.org/get.html;
- Perl package `Unicode::String` from http://www.cpan.org/;
- Perl package `SOAP::Lite` from http://www.cpan.org/.

2.2 Installation

Other than what is mentioned in the previous section, there is no installation required for this tool.

2.3 Execution instructions

There are four operations that are available through TTL: tokenization (which includes sentence splitting), POS tagging, lemmatization and chunking. These operations can be chained through the standard UNIX pipe operator '|' in this **strict order**: tokenization (script name '`ttlws-tokenizer.pl`'), POS tagging ('`ttlws-postagger.pl`'), lemmatization ('`ttlws-lemmatizer.pl`') and chunking ('`ttlws-chunker.pl`').

Each script will receive the language code ('`en`', '`ro`' or '`fr`') and an input file (which may be the result of a previous processing step) and will write the result of the processing to STDOUT. If called with no arguments, the script will output the command line arguments it expects. The possible workflows which may be constructed with these operations are:

- **Tokenization**: call `ttlws-tokenizer.pl` and obtaind a list of tokenized sentences separated with '`\r\n`';
- **POS tagging**: call `ttlws-postagger.pl` on the result of `ttlws-tokenizer.pl` and obtain POS tags for each token;
- **Lemmatization**: call `ttlws-lemmatizer.pl` on the result of `ttlws-postagger.pl` to obtain lemmas for each word;
- **Chunking**: call `ttlws-chunker.pl` on the result of `ttlws-lemmatizer.pl` to obtain the chunk name and id to which the current word belongs;

For instance, if we want to POS tag a Romanian UTF-8 encoded file (all input files must be text and UTF-8 encoded) which resides in the '`test/`' directory under the current working directory in which the Perl scripts `ttlws-tokenizer.pl` and `ttlws-postagger.pl` are, then by calling

```
cat test/ro-test-file.txt | \
     ttlws-tokenizer.pl ro - | \
     ttlws-postagger.pl ro - >result.txt
```

the results of the tokenization and POS tagging of the file ‘`ro-test-file.txt`’ will be written in ‘`result.txt`’. The dash ‘`-`‘ stands for the input file read from STDIN. Alternatively, one can run the following to achieve the same result:

```
ttlws-tokenizer.pl ro test/ro-test-file.txt >result-1.txt
ttlws-postagger.pl ro result-1.txt >result.txt
```

The full workflow can be run as follows:

```
cat test/ro-test-file.txt | \
     ttlws-tokenizer.pl ro - | \
     ttlws-postagger.pl ro - | \
     ttlws-lemmatizer.pl ro - | \
     ttlws-chunker.pl ro - >result.txt
```

2.4 Input/Output data formats

The input data for the first step of the workflow, i.e. tokenization, is UTF-8 text file with no Byte Order Markings (BOM). Each processing step will output the data in text format, column style. Thus, for the input text “`This is a test sentence in English.`”, the tokenizer `ttlws-tokenizer.pl` will output the text:

```
This
is
a
test
sentence
in
English
.    PERIOD
```

in which each token is on a separate line (end of line follows Windows conventions: ‘`\r\n`’). If the token is a punctuation sequence or a named entity, after a tab character (‘`\t`’) follows a label that will be used by the POS tagger ttlws-postagger.pl. Sentences are separated by an empty line ‘`\r\n`’.

Each processing step produces an output that is used by the next step in line. The output are tokens per line with tab characters separating the annotations produced up the current processing step. The full pipeline for our example above produces the output (with extra tabs inserted to ease the reading):

```
This      Pd3-s     this
is        Vmip3s    be        Vp#1
a         Ti-s      a         Np#1
test      Ncns      test      Np#1
sentence  Ncns      sentence  Np#1
in        Sp        in        Ap#1
English   Afp       English   Ap#1
.         PERIOD    .
```

2.5 Integration with external tools

TTL is fully self-contained. There are no external dependencies other than the Perl packages required for the web service wrappers to run.

3 CONTENT INFORMATION

3.1 Test input files

See the distribution kit in the '`test/`' directory. There is one test file for each of the languages supported by TTL.

3.2 Output files

For each input file in the '`test/`' directory, there is the corresponding output file in the '`sample-output/`' directory in the distribution kit.

3.3 Running times

The speeds that we are going to report have been obtained by calling the TTL web service in the local network on a wireless connection of 54Mbps. One should consider the fact that the time the text is passed around the network is going to affect the overall performance of TTL. Also, TTL is running on a Ubuntu Server machine with a 8-core Intel(R) Xeon(R) CPU E5405 @ 2.00GHz CPU and 8 GB of RAM.

We have considered two types of measures: 'bytes/second' and 'tokens/second'. Bytes/second is going to help us predict the number of seconds after which an input UTF-8 text file of N bytes is going to be finished. Tokens/second gives the average processed tokens per second that the TTL web service is capable. We measure the speeds of all possible workflows presented in section 2.3. The server was not idle through the tests and it had 2GB of RAM and 1 core assigned to other process. This means that the reported speeds could be indicative of a low load of the server and could be higher if the server is assigned only to TTL. They can also be lower if the server is fully loaded.

	English	**Romanian**	**French**
1. Tokenization	2603	2255	2348
2. POS Tagging	882	1101	872
3. Lemmatization	782	914	755
4. Chunking	681	652	659

Table 1: TTL speeds in bytes/second on each language. Please note that the processing step no. N includes all processing steps from 1 to N - 1

	English	**Romanian**	**French**
1. Tokenization	456	391	439
2. POS Tagging	155	191	163
3. Lemmatization	137	158	141
4. Chunking	119	113	123

Table 2: TTL speeds in tokens/second on each language. Please note that the processing step no. N includes all processing steps from 1 to N - 1

4 ADMINISTRATIVE INFORMATION

4.1 Contact

For further information, please contact Radu ION (radu@racai.ro).

5 REFERENCES

Brants, T. (2000). TnT – A Statistical Part-Of-Speech Tagger. In *Proceedings of the 6th Applied NLP Conference ANLP-2000*. Seattle, WA, pp 224--231.

Ion, R. (2007). Word Sense Disambiguation Methods Applied to English and Romanian. PhD thesis (in Romanian). Romanian Academy, Bucharest.

Tufiş, D. (1999). Tiered Tagging and Combined Classifiers. In F. Jelinek, E. Nth (Eds.), *Text, Speech and Dialogue, Lecture Notes in Artificial Intelligence*. Springer, pp. 28--33.