Text Tokenisation Using `unitok`

Jan Michelfeit, Jan Pomikálek, and Vít Suchomel

Natural Language Processing Centre
Faculty of Informatics, Masaryk University
Botanická 68a, 602 00 Brno, Czech Republic
`{xmichelf,xsuchom2}@fi.muni.cz`

Lexical Computing Ltd.
Brighton, United Kingdom
`name.surname@sketchengine.co.uk`

Abstract. This paper presents `unitok`, a tool for tokenisation of text in many languages. Although a simple idea – exploiting spaces in the text to separate tokens – works well most of the time, the rest of observed cases is quite complicated, language dependent and requires a special treatment. The paper covers the overall design of `unitok` as well as the way the tool deals with some language or web data specific tokenisation cases. The rule what to consider a token is briefly described. The tool is compared to two other tokenisers in terms of output token count and tokenising speed. `unitok` is publicly available under the GPL licence at `http://corpus.tools`.

Keywords: tokenisation, corpus tool

1 Introduction

Tokenisation of a text is the process of splitting the text to units suitable for further computational processing. It is an important data preparation step allowing to perform more advanced tasks like morphological tagging or parsing. Applying a good tokenisation method is necessary for building usable text corpora.

Using spaces in the text as token delimiters is a simple and quick way to tokenise text. In fact, the presented approach is based on this observation. However, there are many complicated cases to deal with which cannot be solved just by splitting tokens by space characters. Some cases are language dependent and require a special treatment. Other sequences to recognize as single or multiple tokens come from the web pages – a rich yielding source of data in text corpora [4].

The aim of this work was to develop a tokeniser

– fast – able to process big data in billion word sized corpora,
– reliable – robust to deal with messy web data,

- universal – allowing at least basic support for all writing systems utilizing a whitespace to separate words[1],
- easy to maintain – adding new tokenisation rules or making corrections based on evaluation should be straightforward,
- text stream operating – text in, tokenised text (one token per line) out[2],
- reversible – the tokenised output must contain all information needed for reconstructing the original text[3]

The resulting tool called `unitok` has been developed since 2009. It has become a part of corpus processing pipeline used for many corpora [3,1,6] since then.

2 The Problem of Words

There are legitimate questions and related problems concerning the desired behaviour of a good tokeniser: What is a word, what is a sentence? [2] Our approach to `unitok` is based on the point of view of a corpus user. It is important to know what tokens the users search for in the corpus concordancer (and other corpus inspection tools) and what tokens they expect to figure in the corpus based analysis (such as word frequency lists, collocations (Word Sketches), thesaurus, etc.).

The answer is the users search for sequences of letters. Sequences of numbers, marks, punctuation, symbols, separators and other characters should be clustered together in order to be counted as single tokens in corpus statistics. The definition of the charater classes and mapping of each letter to a class has been made by The Unicode Consortium.[4]

3 Implementation

3.1 Related Work

A set of rules in flex[5] has been used by [5] to implement a tokeniser. We chose to write a Python script heavily utilising the `re` library for manipulation with

[1] The dependency of the tool on space characters separating words is a prerequisite for a broad coverage of languages/writing systems but rules out applicability to processing text in languages such as Chinese, Korean, Japanese and Thai. We have to employ other tools made for the particular language/script in these cases to tokenise texts well.

[2] The stream operating feature is necessary for making the tool a part of a corpus processing tool pipeline: the source plain text goes in, each processing tool is applied in a defined order to the result of the previous tool, the fully processed (tokenised, tagged, annotated, etc. data comes out.

[3] Reconstructing the original text can be useful for applying the tokenisation again after improving the tokenisation rules. Other tokenisers we use do not offer this option, therefore a copy of the original plain text has to be kept along with the tokenised vertical.

[4] `http://unicode.org`, the character class mapping is published at `http://www.unicode.org/Public/UNIDATA/UnicodeData.txt` (accessed 2014-11-17).

[5] A fast lexical analyzer, `http://flex.sourceforge.net/`

regular expressions[6]. The reason is we wanted to deal with some practical issues (described later on in this chapter) programmatically rather than by string/regexp matching and in the way which is more familiar to us.

We would like to acknowledge Laurent Pointal's Tree Tagger wrapper[7] which contributed the regular experessions to match the web related tokens. Unlike the Tree Tagger wrapper, `unitok` does just the tokenisation. It works independently of a particular tagger and thus can be combined with any tagger operating on text streams. The subsequent tagging, if required, is defined by the whole text processing pipeline.

3.2 The Method

As has been stated, `unitok` comes as a self standing Python script utilising the `re` library and operating on a text stream. The input text is decoded to unicode, normalised, scanned for sequences forming tokens, the tokens are separated by line breaks and the result vertical (one token per line) is encoded into the original encoding.

The normalisation deals with SGML entities by replacing them with unicode equivalents (e.g. '*&amp;*' by '*&*' or '*&ndash;*' by '*–*'). Unimportant control characters (the 'C' class in the Unicode specification) are stripped off. All whitespace (the 'S' class, e.g. a zero width space or a paragraph separator) is replaced by a single ordinary space.

Sequences of letters (i.e. words) are kept together. Sequences of numbers, marks, punctuation, and symbols kept by the normalisation are clustered together. The present SGML markup (usually HTML/XML) is preserved. The script also handles web related tokens: URLs, e-mail addreses, DNS domains, IP addresses are recognized. General abbreviations (uppercase characters optionally combined with numbers) are recognized.

Predefined language specific rules are applied. These are recognizing clitics (e.g. '*d'accord*' in French)[8], matching abbreviations (e.g. '*Prof.*' generally, '*např.*' in Czech)[9]) or special character rules (e.g. Unicode positions from 0780 to 07bf are considered word characters in Maldivian script Thaana).

The reversibility of tokenisation is ensured by inserting a 'glue' XML element between tokens not separated by a space in the input data. The most common case is the punctuation. An example showing the placement of the glue tag is given by Figure 1. The vertical with glue tags can be reverted to the original plain text using a short accompanying script `vert2plain`.

The language specific data (clitics, abbreviations, special word characters) are currently available for Czech, Danish, Dutch, English, French, Finnish,

[6] `https://docs.python.org/2/library/re.html`

[7] `http://perso.limsi.fr/pointal/dev:treetaggerwrapper`

[8] Lists of clictics were taken from the TreeTagger: `http://www.ims.uni-stuttgart.de/projekte/corplex/TreeTagger/`

[9] The lists of language specific abbreviations were taken from the respective Wikipedia page, e.g. `http://cs.wiktionary.org/wiki/Kategorie:%C4%8Cesk%C3%A9_zkratky` for Czech.

```
The
"
<g/>
end
<g/>
"
<g/>
.
```

Fig. 1: Example of tokenised and verticalised string *'The "end".'* showing the placement of the glue tag <g/>.

German, Greek, Hindi, Italian, Maldivian, Spanish, Swedish, Yoruba. Texts in other language are processed with default setting (making e.g. the *'prof.'* abbreviation always recognizable).

The output can be tagged (or generally further processed) by any tool operating on text streams with one token per line and XML tags.

4 Evaluation

A comparison of the output token count and the speed (expressed in output tokens per second) of three tokenisers – *unitok*, *TreeTaggerWrapper* and *Freeling* – can be found in Table 1. Two measurements were carried out using 1 GB sized plain texts in six European languages on a single core of Intel Xeon CPU E5-2650 v2 2.60 GHz. Specific recognition of clitics and abbreviations was on for all except Russian where the general setting was in effect. Only languages supported by TreeTaggerWrapper and Freeling were included in the test.

We found there is a noticeable difference between the tools in the number of output tokens. The speed tests revealed unitok was the slowest of the three tools but still quite sufficient for fast processing of large text data. Part of the performance drop is caused by providing the glue marks.

5 Conclusion

unitok is a fast processing tool for tokenisation of texts with spaces between words. The main benefits are good coverage of various sequences of characters, especially web phenomena, normalisation of messy control or whitespace characters, reversibility of the tokenised output and extensibility by language specific rules.

The tool has been successfully used for tokenising source texts for building large web corpora. The evaluation suggests we should consider improving the script to increase the speed of tokenisation in the future.

Acknowledgements This work has been partly supported by the Ministry of Education of CR within the LINDAT-Clarin project LM2010013 and by the Czech-Norwegian Research Programme within the HaBiT Project 7F14047.

Table 1: Comparison of tokenising speed and token count of `unitok`, *Freeling* and *TreeTaggerWrapper*. `unitok` is the base of the relative token count and the relative tokenising speed.

Language	tool	output tokens	rel tok	duration	tok/s	rel tok/s
English	Unitok	207,806,261	100%	6,880 s	30,200	100%
	TTWrapper	200,122,178	−3.70%	2,380 s	84,100	+178%
	Freeling	215,790,562	+3.84%	2,670 s	80,800	+168%
Spanish	Unitok	196,385,184	100%	6,250 s	31,400	100%
	TTWrapper	204,867,056	+4.32%	2,260 s	90,600	+188%
	Freeling	201,413,272	+2.56%	2,040 s	98,700	+214%
German	Unitok	171,354,427	100%	5,530 s	31,000	100%
	TTWrapper	179,120,243	+4.53%	2,360 s	75,900	+145%
French	Unitok	202,542,294	100%	6,400 s	31,600	100%
	TTWrapper	242,965,328	+20.0%	2,870 s	84,700	+168%
	Freeling	211,517,995	+4.43%	2,300 s	92,000	+191%
Russian	Unitok	98,343,308	100%	3,170 s	31,023	100%
	Freeling	102,565,908	+4.29%	1,450 s	70,800	+128%
Czech	Unitok	183,986,726		5,960 s	30,900	

References

1. Vít Baisa, Vít Suchomel, et al. Large corpora for turkic languages and unsupervised morphological analysis. In *Proceedings of the Eighth conference on International Language Resources and Evaluation (LREC'12), Istanbul, Turkey. European Language Resources Association (ELRA)*, 2012.
2. Gregory Grefenstette and Pasi Tapanainen. *What is a word, what is a sentence?: Problems of Tokenisation*. Rank Xerox Research Centre, 1994.
3. Miloš Jakubíček, Adam Kilgarriff, Vojtěch Kovář, Pavel Rychlý, Vít Suchomel, et al. The tenten corpus family. In *Proc. Int. Conf. on Corpus Linguistics*, 2013.
4. Adam Kilgarriff and Gregory Grefenstette. Introduction to the special issue on the web as corpus. *Computational linguistics*, 29(3):333–347, 2003.
5. David D Palmer. Tokenisation and sentence segmentation. *Handbook of natural language processing*, pages 11–35, 2000.
6. Vít Suchomel. Recent czech web corpora. In Pavel Rychlý Aleš Horák, editor, *6th Workshop on Recent Advances in Slavonic Natural Language Processing*, pages 77–83, Brno, 2012. Tribun EU.