Extended tokenizer for Polish

Tomasz Bartosiak
Konrad Gołuchowski
Katarzyna Krasnowska

1 Method description

The tokenization (augmented with simple tagging with token type) implemented in our program consists of 4 main steps described in the following sections:

- basic tokenization,
- filters cascade,
- date parsing,
- tags clean up.

1.1 Basic tokenization

At this stage, the most basic splitting operations are performed on the input text. Each sentence is split with spaces. Additionally, when the resulting tokens begin or end with interpunction characters, the leading and trailing interpunction are stripped and added as separate tokens. This allows, e.g., for separating parentheses, colons and semicolons from neighbouring tokens. An exception from this is the treatment of a dot preceded by a non-interpunction character. Such a dot is kept within the same token for later processing of abbreviations.

1.2 Filters cascade

In the second stage, the most of the tagging is performed. A series of token-type filters is defined together with an order in which they are applied, forming what we called a *filters cascade*. Each filter may either:

- Recognise a token as belonging to one of the defined types and tag it. In this case, the tagging is done for the given token and the cascade is run on the next token.
- Recognise a token as a concatenation of proper tokens, split them, tag some of them if possible and leave the rest to be recursively passed through the cascade.

- Fail to recognise the token: in this case, the next filter from the cascade is applied.

Therefore, at this stage, token boundaries may be either left as they were determined in the previous stage, or tokens may be further split.

The simplest filters use regular expressions. In that way, e.g., roman/arabic numerals, e-mail or WWW addresses can be tagged. A little more complicated ones may split the token based on a regular expression and assign all resulting parts a tag, as is done, e.g., in the case of arabic numerals followed by a dot.

1.2.1 Abbreviations

The most complex filter is the one used for recognising abbreviations. It uses some general heuristics and makes use of some predefined list of valid abbreviations of different type, included in `.txt` files (see the files description at the end of this document) for special cases.

The first heuristic marks everything that ends with a dot (other than a sentence-ending dot) as abbreviation. Those are either non-inflecting abbreviations or inflected forms of some other abbreviation (e.g., *do dr. Pawłowskiego*). In the case of multi-part abbreviations (e.g., *m.in.*) all parts are treated individually (so there are 4 tokes in this example: *m*/`abbrev` ./`punct` *in*/`abbrev` ./`punct`).

The second heuristic marks every token which begins with an upper-case capital letter and has at least one more inside (not counting those which appear after a dash, like in *Austro-Węgry*). This heuristic aims at recognising abbreviations such as *PKO*, *PKiN*. Inflected forms of such abbreviations are also (mostly) recognised with this simple filter, e.g., *w OSiR-ze*.

All words not ending with dots are looked up on our abbreviations list. The list includes physical units with optional prefixes (`unit_prefixes.txt` + `unit_names.txt`), uninflected abbreviations (`uninflected.txt`) and naively created forms of inflected abbreviations (`inflect_base.txt` + `inflect_ending.txt`).

A dot-ended token at the end of a sentence may also be an abbreviation. It is looked up among the abbreviations not ending with a dot, and those ending with dots (`dots_sorted.txt` and `multi_part_dot_abbr.txt`).

1.2.2 List of applied filters

Below is the list of filters in the order they are applied:

- Arabic integer numbers are temporarily assigned a helper tag `int` (to distinguish them from fractional numbers when parsing dates).

- Fractional numbers are assigned the `ara` tag.
- Arabic numbers followed by a dot are split into two tokens and tagged `ara` and `punct` respectively.
- Upper-case *I* at the beginning of a sentence and lower-case *i* are tagged `word` (so that they are not passed to the roman numbers filter).
- *W*, *A*, *U*, *Z*, *O* at the beginning of a sentence are tagged as `word` (so that they are not passed to the abbreviations filter).
- Roman numbers are assigned the `rom` tag.
- Roman numbers followed by a dot are split into two tokens and tagged `rom` and `punct` respectively.
- Dates in format *dd.mm.yyyy* and *dd/mm/yyyy* are temporarily assigned a helper tag `date`.
- Month names in nominative[1] and genitive are temporarily assigned helper tags, e.g., *kwietnia* is tagged as `m-iv` (for the purpose of date parsing).
- Abbreviations and their dots are recognised, split and tagged.
- Tokens consisting only of letters are tagged as `word`.
- Interpunction characters are tagged as `interp`.
- A dot-ended token at the end of a sentence is split and tagged.
- WWW addresses are tagged as `www`.
- Comma-separated sequences are split and tagged.
- Hyphen-separated sequences are split and tagged.
- E-mail addresses are tagged as `e-mail`.
- Remaining tokens are tagged `punct` if they are single characters and `word` otherwise.

1.3 Date parsing

Last but not least, dates in various text and number format are recognised based on the processing in the previous stage and appropriate tags are determined, which amounts to parsing the dates and producing their representation in the normalised format `yyyy.mm.dd`. At this stage, several tokens may be joined back together to form a single one.

[1] Although dates written as *1 kwiecień 2014* are considered incorrect, they often appear in written and spoken Polish, so they are taken into account.

In the simplest case, day, month and year are retrieved from tokens recognised as `date` by the filter cascade. Other date formats are recognised as specific token patterns, e.g., tag=`int` - tag=`m-`$*$ - tag=`int` - tok=`"r"` - tok=`"."` (this matches, for instance, *14 marca 2014 r.*).

1.4 Clean-up

To complete the extended tokenization process, the helper tags `int` and `m-`$*$ tags are replaced with `ara` and `word` respectively. As the last small step, *I* and *i* tokens followed by a closing parenthesis *)* are assigned the `rom` tag (those are most likely items on a numbered list, but the heuristic for distinguishing between conjunction and roman 1 applied at the filter stage did not look at the next token and tagged them as `word`).

2 Authors' contributions

Tomasz Bartosiak: handling XML format of input/output files, abbreviations.

Konrad Gołuchowski: filters cascade stage: project and particular filters.

Katarzyna Krasnowska: filters cascade stage: particular filters, dates.

Besides the above, each author provided some testing input files, repeatedly tested the method against those files and fixed or reported detected problems.

3 Files description

Python source code:

- `main.py` – the main program file, contains high-level code for handling input/output files and code for first-stage tokenization.
- `token.py` – contains filters used in the filters cascade stage.
- `date.py` – contains code for handling dates.
- `tags.py` – tag names defined as constants for convenience.
- `ext_tokenizer_xml_parsing.py` – contains code for parsing and printing XML files.

Other files used by the program:

- `dots_sorted.txt` – list of abbreviations ending with dot.
- `multi_part_dot_abbr.txt` – list of multi-part abbreviations ending with dot.
- `unit_names.txt` – list of abbreviations for physical units.
- `unit_prefixes.txt` – list of prefixes for physical units.
- `uninflected.txt` – list of uninflected abbreviations not ending with dot
- `inflect_base.txt` – list of stems for inflected abbreviations.
- `inflect_ending.txt` – list of possible endings for inflected forms of abbreviations.