SoMaJo: State-of-the-art tokenization for German web and social media texts

Thomas Proisl
FAU Erlangen-Nürnberg
Professur für Korpuslinguistik
Bismarckstr. 6
91054 Erlangen, Germany
thomas.proisl@fau.de

Peter Uhrig
FAU Erlangen-Nürnberg
Lehrstuhl für Anglistik, insbesondere Linguistik
Bismarckstr. 1
91054 Erlangen, Germany
peter.uhrig@fau.de

Abstract

In this paper we describe SoMaJo, a rule-based tokenizer for German web and social media texts that was the best-performing system in the EmpiriST 2015 shared task with an average F_1-score of 99.57. We give an overview of the system and the phenomena its rules cover, as well as a detailed error analysis. The tokenizer is available as free software.

1 Introduction

At first sight, tokenization is not only boring but also trivial. Humans have few problems with this task for at least two reasons: (1) They are experts at pattern-finding (see, for example, Tomasello, 2003). Thus, whether the form "your" in an English Facebook post is to be read as one unit (the possessive determiner) or as two (a common misspelling of "you're"), usually causes less problems due to the highly disambiguating grammatical context. (2) They are happy to accept meaningful units without having to determine the exact number of units. While most tokenization guidelines force us to treat "ice cream" as two tokens and "ice-cream" as one token, there often is no difference to native speakers – though it is possible to predict the spelling to some extent based on linguistic context, frequency, etc. (cf. Sanchez-Stockhammer, in preparation).

However, given the layered approach typically taken by NLP pipelines, no analysis of the grammatical context is available at the time when tokenization takes place since tokenization is one of the first steps in an NLP text processing pipeline, often only preceded by sentence splitting.[1] However, tokenization is not fully independent of sentence splitting due to the ambiguity of some punctuation marks, most notoriously the baseline dot, which can for instance occur as (1) period/full stop to mark the end of a sentence, (2) marker of abbreviated forms, (3) decimal mark separating the integer from the fractional part of a number, (4) separator of host name, subdomain, domain, top-level domain in Internet addresses, (5) part of a so-called horizontal ellipsis ("..."). When all these restrictions are in place, tokenization immediately becomes more challenging as a task, also for humans. Thus whether the string "No." should be treated as one token or as two is impossible to decide out of context, since it could be a short answer to a question ("Would you like to join us for lunch?" – "No.") or it can be an abbreviation for "number" ("No. 6"). In the former case, tokenization should identify two tokens, in the latter only one. Thus the challenge for any tokenizer is to make use of the linguistic context to disambiguate potentially ambiguous forms even though no higher-level grammatical analysis (i. e. PoS-tagging, lemmatization or even syntactic or semantic analysis) is available. In a way, some of the work done by these high-level tools is thus duplicated in the tokenizer, e. g. identifying numbers, identifying punctuation, identifying proper names (in English) or nouns in general (in German) based on capitalization, where necessary for the tokenization.

Of course, an extremely large proportion of tokenization is indeed straightforward. A simple split on white space and common punctuation marks will result in an average F_1-score of 96.73 on the test data set used for the present task (cf. Section 4). However, the amount of work that is required to get closer to 100% is inversely proportional to the effect size of the improvements that can be achieved, which means that the bulk of this paper is devoted to the remaining 3.27%.

[1] In order to arrive at a sensible text corpus, there may of course be other preprocessing steps involved, such as boilerplate removal or duplicate detection.

Proceedings of the 10th Web as Corpus Workshop (WAC-X) and the EmpiriST Shared Task, pages 57–62,
Berlin, Germany, August 7-12, 2016.

The EmpiriST 2015 shared task on automatic linguistic annotation of computer-mediated communication / social media (Beißwenger et al., 2016) consists of two subtasks that deal with NLP for web and social media texts: (1) Tokenization and (2) part-of-speech tagging. We participated in the first subtask and developed a rule-based tokenizer that implements the EmpiriST 2015 tokenization guidelines (Beißwenger et al., 2015; EmpiriST team, 2015). Our system, SoMaJo, won the shared task and is freely available from PyPI, the Python Package Index.[2]

2 Related work

The most widespread approach to tokenization is probably the application of substitutions based on regular expressions, as examplified by the simple sed script for Penn-Treebank-style tokenization.[3] Typically, every piece of software that relies on tokenized input ships with its own tokenizer (usually rule-based), e. g. TreeTagger (Schmid, 1994; Schmid, 1995) or the Stanford Parser (Klein and Manning, 2003). There are, however, also systems that use supervised or unsupervised machine learning techniques, e. g. the maximum entropy tokenizer offered by the Apache OpenNLP project[4] or the HMM-based one presented by Jurish and Würzner (2013). For an overview of existing approaches to tokenization (and the related task of sentence splitting), see Jurish and Würzner (2013).

3 System description

3.1 General approach

SoMaJo is a rule-based tokenizer that applies a cascade of regular expressions to the input text to arrive at a tokenized version. In that process, recognized tokens that could be "problematic" further down the rule chain are replaced with unique pseudotokens. The major reason for why tokens could be problematic for subsequent rules is that they can contain certain characters that trigger those rules. URLs, for example, should be treated as single tokens and should not be split at dots, hyphens, slashes, etc. After all the rules have been applied, the original tokens are restored from the pseudotokens. Additionally, SoMaJo can output the token class for each token, e. g. if it is a number, an XML tag, an abbreviation, etc.

3.2 Specifics

In this subsection, we will give a high-level overview of the most important rules, in the order in which they are applied. The ultimate reference to what the tokenizer does is of course its freely available source code.

- The identification of **XML tags** was performed with a regular expression taken from Goyvaerts (2012). XML tags are special because they are among the few tokens that can contain spaces. Spaces are normally unambiguous token delimiters, therefore we want to deal with XML tags as early as possible. Since there is no syntax check, non-XML conforming standalone tags without trailing slash, eg. "
" as used in traditional HTML/SGML will also be detected. Attributes without quotation marks – as allowed in SGML – are not covered.
- The regular expression for **email addresses** is a revised version of the pattern given in Goyvaerts (2012) available from Goyvaerts website,[5] where he claims that it covers "99% of the email addresses in use today". As discussed in Section 4.4, email address obfuscation was not taken into consideration in the original system, but a basic detection has now been incorporated for the release.
- The detection of **URLs** that include the protocol used is relatively straightforward. Our system currently detects "http", "https", "ftp", "svn", "doi" and treats strings with a leading "www." the same, even though it is not technically a protocol.
  URLs without a protocol and the "www" giveaway are detected based on a very conservative list of top-level-domains in order to minimize false positives that could occur when spaces at the end of a sentence are omitted, which often occurs in CMC, particularly in restricted-length messages such as the Twitter messages given in the training and test data. A small list of three-letter file extensions was also added to detect **file names** with internal dots.

[2] https://pypi.python.org/pypi/SoMaJo

[3] https://www.cis.upenn.edu/~treebank/tokenizer.sed

[4] https://opennlp.apache.org/

[5] http://www.regular-expressions.info/email.html

- For **emoticons** it was possible to build on top of a list taken from the SentiKLUE polarity classifier (Proisl et al., 2013; Evert et al., 2014), which was extended based on websites with technology-mediated communication such as *Chat von gestern Nacht*[6] and complemented by a generic regular expression to account for further emoticons consisting of eyes, optional nose and/or tear and mouth.
- Further phenomena that are specific to **Twitter and chat** are also identified with relatively simple regular expressions and treated according to the tokenization guidelines. These include mentions ("@MimiSchmitz"), hashtags ("#lyrik") and actions words ("*kopfkratz*").
- In order to be able to distinguish between ad-hoc **combinations with plus signs ("+") or ampersands ("&")** such as "Thomas&Peter", and institutionalized combinations such as "Taylor&Francis", a lexicon of the latter was constructed based on a manually curated list of all Wikipedia page titles in the German Wikipedia that contain a plus sign and/or an ampersand, which results in a total of 643 items.
- Items written in **CamelCase**, i. e. single orthographic words with internal capitalization ("deineMutter"), had to be split up according to the tokenization guidelines unless they were proper names ("MySpace") or textual representations of emojis ("emojiQcatFaceWithWrySmile"). In order to distinguish the two cases, a lexicon of potential proper names (in a broad sense) and established forms was created based on a list of all words in Wikipedia page titles in the German Wikipedia that include an internal upper-case letter following at least one lower-case letter. The lexicon comprises 7,005 such items.

  However, the splitting of CamelCase can be switched off in our system since the behaviour propagated by the tokenization guidelines is in fact highly problematic in unrestriced input. Thus CamelCase is used in certain wikis, in particular the original wiki software Wiki Wiki Web by Ward Cunningham[7] to create links to other pages and it is often found in naming conventions of programming languages such as Java or C#. So if the input to tokenize contains computer-mediated communication from sources such as stackoverflow.com, it would be advisable to switch CamelCase splitting off. Furthermore, CamelCase splitting makes corpora useless for reasearch on non-standard graphemics.

  An exception was also made for the German internal *I* as in "StudentInnen", which is never split up. URLs written in CamelCase, e. g. "ImmobilienScout24.de", are already recognized as a single token by the earlier rule identifying URLs.
- According to the tokenization guidelines, **abbreviations** representing multiple tokens (e. g. "d. h." for "das heißt") have to be split up unless they are established netspeak units such as "aka" or "cu". Thus three cases have to be distinguished: (1) Abbreviations that do not consist exclusively of single letters followed by a full stop have to be listed in a lexicon in order to not mistake them for sentence boundaries. For this, all 4,027 abbreviations listed in the German Wiktionary[8] on 10 February 2016 were downloaded and then manually checked for candidates that represented a single token (and did so unambiguously), which resulted in a total of 1,104 such abbreviations (e. g. "altröm" for "altrömisch"). (2) A further list of 29 multi-dot abbreviations that represent single tokens was created – 8 from the training data and the tokenization guidelines, 21 from the Wiktionary list of abbreviations mentioned above (e. g. "Dipl.-Ing." for "Diplomingenieur"). (3) A single letter followed by a full stop was always treated as an abbreviation, so single letters at the end of a sentence ("Ich kaufe ein E.") will be analysed erroneously. However, since such occurrences are rather rare, the decision to treat them as abbreviations will definitely lead to higher recognition rates.
- **Dates** had to be split up according to the tokenization guidelines so that day, month and year are treated as separate tokens. The matter is complicated by the fact that separators have to be in the same token as day or month ("05/15/2016" is tokenized as "05/ 15/ 2016" but "2016-05-15" is tokenized as "2016 -05 -15"), so mutliple regular expressions were needed to account for all typical cases.

[6] http://www.chatvongesternnacht.de

[7] http://c2.com/cgi/wiki?WikiWikiWeb

[8] https://de.wiktionary.org/

- Other combinations with **numbers** closely follow the tokenization guidelines, so indications of time (e. g. "12:30"), ordinal numbers and fractions are treated as one token as long as there is no space intervening.
  To be able to split numbers from their unit of measurement (e. g. "80kg"), a list of such units was compiled manually which is certainly far from complete and would need to be expanded particularly if CMC data from science domains is to be processed.
  Cardinal numbers were matched with both a dot or a comma as decimal mark since in CMC, the English format can often be found in German texts, despite the comma being the standard. Our number identifier further allows for signed and unsigned numbers and an optional exponent.
- Our treatment of **punctuation** is fairly standard. We allow for arbitrary combinations of question and exclamation marks, detect arrows, various styles of parentheses, quotation marks (including Unicode quotation marks and LaTeX-style quoation marks using backticks and apostrophes), ellipses (both as combinations of dots and as Unicode entities) and of course standard full stops.

4 Results and error analysis

4.1 Evaluation metrics

The performance of the systems participating in the shared task was evaluated using precision, recall and F_1-score (Jurish and Würzner, 2013, 72–73). These measures are based on the actual token boundaries (B_{actual}), i. e. the token boundaries in the gold standard, and the token boundaries identified by the system ($B_{\text{identified}}$). Correctly detected token boundaries that are both in the system output and in the gold standard are true positives, erroneously introduced token boundaries that are not in the gold standard are false positives and token boundaries in the gold standard that the system fails to detect are false negatives:

$$\text{tp} = |B_{\text{actual}} \cap B_{\text{identified}}|$$

$$\text{fp} = |B_{\text{identified}} \setminus B_{\text{actual}}|$$

$$\text{fn} = |B_{\text{actual}} \setminus B_{\text{identified}}|$$

Precision measures how many of the token boundaries that the system has detected are true token boundaries, recall measures how many of the true token boundaries have been found and the F_1-score is the harmonic mean of precision and recall:[9]

$$\text{precision} = \frac{|B_{\text{actual}} \cap B_{\text{identified}}|}{|B_{\text{identified}}|} = \frac{\text{tp}}{\text{tp}+\text{fp}}$$

$$\text{recall} = \frac{|B_{\text{actual}} \cap B_{\text{identified}}|}{|B_{\text{actual}}|} = \frac{\text{tp}}{\text{tp}+\text{fn}}$$

$$F_1 = \frac{2 \cdot \text{precision} \cdot \text{recall}}{\text{precision} + \text{recall}}$$

The ranking of the participating systems was based on macro-averaged F_1-scores, i. e. the arithmetic mean of the F_1-scores for the two datasets.

4.2 Ad-hoc baseline

As mentioned in Section 1, tokenization is not usually regarded as a terribly hard problem and depending on the task at hand, ad-hoc solutions centered around simple regular expressions often yield sufficiently good results. Therefore, we will use such a primitive ad-hoc tokenizer as a baseline. This simple tokenizer is a sed one-liner that ignores lines that look like they consist of an XML tag (because such lines are not part of the evaluation) and introduces token boundaries at whitespace and a couple of common punctuation symbols:

```
sed -re "/^<[^>]+>$/! {
    s/([.!?,;:+*()\"'-])/ \1 /g;
    s/\s+/\n/g }"
```

4.3 Results

Results for the baseline tokenizer, our submitted system and a revised version of our system fixing some of the most frequent types of errors (cf. next section) are summarized in Table 1.

For the CMC dataset with samples from different CMC genres, the submitted systems have F_1-scores ranging from 97.83 to 99.54, clearly outperforming the baseline tokenizer's F_1-score of 94.91. Our system outperformed all others with an F_1-score of 99.54 and a lead of 0.58 to the second-ranked system.

For the web corpus dataset with samples from text genres on the web, the F_1-scores of the submitted systems range from 99.39 to 99.77, still outperforming the baseline's 98.55 but by a much smaller margin. Our system ranks third with an F_1-score of 99.60 and a difference of 0.17 to the best-performing system.

[9] Note that the precision, recall and F_1-scores reported in this paper are all multiplied by 100 for better readability.

	CMC			Web corpora			macro average
	P	R	F	P	R	F	F
baseline	91.84	98.20	94.91	98.27	98.84	98.55	96.73
submission	99.52	99.56	99.54	99.57	99.64	99.60	99.57
revised	99.62	99.56	99.59	99.83	99.92	99.87	99.73

Table 1: Results

The averaged F_1-scores of the participating systems range from 98.61 to 99.57, with our submission leading the field by a 0.21 margin.

With some of the major remaining error sources fixed, the revised version of our system would also rank first on the web corpus dataset with an F_1-score of 99.87.

4.4 Error analysis

The submitted version of our system had 25 false positives and 23 false negatives in the CMC dataset and 33 false positives and 27 false negatives in the web corpus dataset. In the remainder of this section we will have a closer look at these errors, categorize them and fix the obvious ones. Results for the revised version of our system have been given in Section 4.3.

- 6 false positives and 3 false negatives are due to tokenization errors in the gold standard data. These errors have been pointed out to the task organizers and will be corrected in the next release of the data.
- 21 false negatives are due to our system not being aware of the en dash (–) that is used for example as *Streckenstrich* in "Herford–Lage–Detmold–Altenbeken–Paderborn".
- Our system was also not aware of file names containing slashes (/), which results in 8 false positives.
- Email address obfuscation using, for example, "[at]" and "[dot]" instead of the at (@) and dot (.) characters accounts for 8 false positives.
- 7 false positives are due to emoticons not in our lexicon (":!:", ":p" and ":;-))").
- The list of tokens containing an ampersand (&) was accidentally used case sensitively, resulting in 2 false positives.
- In some cases, a hyphen (-) is used as a *Bis-Strich* to indicate a range instead of the typographically correct en dash (–). This accounts for 12 false negatives and is difficult to fix since hyphens are normally used in compounds (*Bindestrichkomposita*) that should not be split up.
- 9 false positives are due to abbreviations that could also be words, e. g. "automat." or "zum."
- The ambiguity between a cardinal number at the end of a sentence and an ordinal number accounts for 3 false positives and 1 false negative.
- 5 false negatives and 7 false positives are due to tokens written without spaces between them and follow-up errors.
- Citations, e. g. "Storrer2007", are responsible for 2 false negatives and are difficult to distinguish from proper names like "Blume2000".
- Sometimes, two consecutive years are given as "1829/30" or "2009/2010". This accounts for 6 false negatives and is potentially problematic because of the ambiguity with fractions (that are single tokens) and term specifications like "WS05/06" that are tokenized as "WS 05/06", i. e. the two consecutive years are a single token.
- The remaining 8 false positives are due to other rare and unsystematic problems.

5 Conclusion

Tokenization is clearly one of the easier NLP problems, as should be obvious from the fairly good results that can be achieved even with the most primitive methods. Improving upon that baseline takes considerably more effort, however.

In this paper we presented SoMaJo, a rule-based tokenizer that won the EmpiriST 2015 shared task on automatic linguistic annotation of computer-mediated communication / social media. Since it is a rule-based system it is easy to maintain and adapt. Thanks to this flexibility it was easy to create a revised version of the system that incorporates insights from the error analysis and achieves even better results.

References

Michael Beißwenger, Sabine Bartsch, Stefan Evert, and Kay-Michael Würzner. 2015. Richtlinie für die manuelle Tokenisierung von Sprachdaten aus Genres internetbasierter Kommunikation. Guideline document from the Empirikom shared task on automatic linguistic annotation of internet-based communication (EmpiriST 2015). `https://sites.google.com/site/empirist2015/`.

Michael Beißwenger, Sabine Bartsch, Stefan Evert, and Kay-Michael Würzner. 2016. EmpiriST 2015: A shared task on the automatic linguistic annotation of computer-mediated communication, social media and web corpora. In *Proceedings of the 10th Web as Corpus Workshop (WAC-X)*, Berlin, Germany.

EmpiriST team. 2015. Ergänzungsdokument zu den Annotationsrichtlinien. Additional instructions and examples for selected PoS categories and tricky phenomena in CMC and social media data. `https://sites.google.com/site/empirist2015/`.

Stefan Evert, Thomas Proisl, Paul Greiner, and Besim Kabashi. 2014. SentiKLUE: Updating a polarity classifier in 48 hours. In *Proceedings of the 8th International Workshop on Semantic Evaluation (SemEval 2014)*, pages 551–555. ACL.

Jan Goyvaerts. 2012. *Regular Expressions Cookbook*. O'Reilly, Sebastopol, CA, 2nd edition.

Bryan Jurish and Kay-Michael Würzner. 2013. Word and sentence tokenization with Hidden Markov Models. *JLCL*, 28(2):61–83.

Dan Klein and Christopher D. Manning. 2003. Accurate unlexicalized parsing. In *Proceedings of the 41st Annual Meeting of the Association for Computational Linguistics*, pages 423–430.

Thomas Proisl, Paul Greiner, Stefan Evert, and Besim Kabashi. 2013. KLUE: Simple and robust methods for polarity classification. In *Proceedings of the 7th International Workshop on Semantic Evaluation (SemEval 2013)*, pages 395–401. ACL.

Christina Sanchez-Stockhammer. In preparation. *Determinants of English Compound Spelling*.

Helmut Schmid. 1994. Probabilistic part-of-speech tagging using decision trees. In *Proceedings of the International Conference on New Methods in Language Processing*, pages 44–49.

Helmut Schmid. 1995. Improvements in part-of-speech tagging with an application to German. In *Proceedings of the EACL SIGDAT-Workshop*, pages 47–50, Dublin.

Michael Tomasello. 2003. *Constructing a Language: A Usage-Based Theory of Language Acquisition*. Harvard University Press, Cambridge, MA.