Simple and Accountable Segmentation of Marked-up Text

Jonathon Read,[*♠] *Rebecca Dridan,*[♣] *Stephan Oepen*[♣]

[♠] School of Computing, Teesside University, UK

[♣] Department of Informatics, University of Oslo, Norway

j.read@tees.ac.uk, {rdridan,oe}@ifi.uio.no

Abstract

Segmenting documents into discrete, sentence-like units is usually a first step in any natural language processing pipeline. However, current segmentation tools perform poorly on text that contains markup. While stripping markup is a simple solution, we argue for the utility of the extra-linguistic information encoded by markup and present a scheme for normalising markup across disparate formats. We further argue for the need to maintain *accountability* when preprocessing text, such that a record of modifications to source documents is maintained. Such records are necessary in order to augment documents with information derived from subsequent processing. To facilitate adoption of these principles we present a novel tool for segmenting text that contains inline markup. By converting to plain text and tracking alignment, the tool is capable of state-of-the-art sentence boundary detection using any external segmenter, while producing segments containing normalised markup, with an account of how to recreate the original form.

Keywords: Accountability, Markup, Normalisation, Sentence Boundary Detection, Traceability.

[*] Work carried out at the University of Oslo.

1 Introduction

Attention in Natural Language Processing (NLP) research increasingly focuses on text from the World Wide Web, the so-called 'Web as Corpus' (Kilgarriff and Grefenstette, 2003). This fertile source of language use has enriched the field by greatly expanding the genres and styles of text being analysed by standard NLP techniques, including much information available as user-generated content. One outcome of this recent attention is a host of research showing the degradation of performance on web text when using tools trained on, for example, the venerable Penn Treebank (Marcus et al., 1993). Techniques such as domain adaptation (Foster et al., 2011) and normalisation (Gimpel et al., 2011) have been shown to reduce the impact on performance of tools such as statistical parsers and part-of-speech taggers. One particular aspect that has not had much attention in this regard is sentence segmentation, frequently a fundamental piece of any NLP pipeline, from corpus building to machine translation. In the following sections, we look at a different technique for handling issues at this often-overlooked stage.

While the differences in style and register can have an effect on sentence segmentation, the major impact is from the markup (such as HTML or WikiText) that is present in web-sourced text. Reuse of standard tools is generally desirable, since we can take advantage of previous tuning on plain text, but using tools designed for plain text on text with markup can result in a major loss in recall of sentence boundaries. This occurs when markup elements obscure sentence-ending punctuation (Read et al., 2012b). For instance, in the example given in Figure 1 the sentence terminating full-stop is attached to the closing tag. Thus, any tool that operates under the assumption that full-stops must have whitespace to the right will fail to detect this boundary.

To avoid this problem, markup could be—and often is—stripped and discarded during processing. However, another frequently ignored aspect of preprocessing is *accountability*: the recording of actions taken that alter the text, in order to preserve the link to the original data. Many applications of NLP augment the original document with new information. For example, the ACL Anthology Searchbench (Schäfer et al., 2011) highlights search hits in text from scientific papers—such annotations would also be useful in HTML and other source documents. When unilaterally throwing away markup, it becomes difficult to trace back to the original source and hence one loses some of the utility of the annotations from downstream processing. Alternatively, in corpus creation, discarding information without record can unnecessarily limit future unforeseen uses of the corpus, whereas full accountability leaves open possibilities that cannot be predicted at creation time.

To facilitate the accountability of preprocessing when building corpora from marked-up documents, in Section 2 we present a novel tool[1] for sentence boundary detection of text with inline markup. By tracking alignment while creating a plain text version, the tool is able to utilise any third-party software for sentence segmentation. Our tool then recovers sentence positions in the original, and produces one sentence per line along with an account of how the text corresponds to the original document. While exemplified in application to sentence boundary detection here, this general technique is applicable of course to other processing downstream as well, where it is deemed desirable to invoke off-the-shelf tools that lack support for markup processing (tokenisation, for example).

Another issue with simply stripping markup is that it can be of relevance for natural language processing tasks. For instance, some markup (e.g. block elements such as paragraphs, quotes and list elements) strongly influences sentence segmentation (Read et al., 2012a). Other elements are of

[1] The tool is open-source and available to download from `www.delph-in.net/deler/`.

```
<p>The name <b>Clanfield</b> is derived from the <a>Old English</a>
and means “field clean of weeds”.</p><p>Clanfield was historically
a small <a>farming</a> community.</p>
```

Figure 1: A passage illustrating how HTML markup can obscure sentence-ending punctuation.

relevance to part-of-speech tagging and parsing (e.g. italics may indicate foreign language words and links may offer clues to constituent structure). However, exactly which markup elements are relevant for which tasks in natural language processing remains to be explored. In order to answer comprehensively, it is important to normalise the disparate types of markup (such as LaTeX, WikiText, or XML) so that they can be treated generally. For such normalisation we provide options that allow a user to specify how different elements of markup should be treated, including a configurable mapping option. Following previous work (Flickinger et al., 2010; Solberg, 2012), we have used this option to map certain source elements into an abstraction of these markup languages dubbed the Grammar Markup Language (described in Section 3).

In Section 4 we evaluate the tool using two collections of text from blogs, and provide an error analysis. We then summarise our work and outline the future directions for improvements in Section 5.

2 A Generic Approach to Segmenting Text with Inline Markup

In this section we describe our generic approach to sentence boundary detection in text with inline markup. The approach is generic in that it is (a) applicable across other markup formats, (b) operates independently of the strategy for sentence boundary detection, and (c) could conceivably be applied to segmentation at other levels (e.g. tokenisation). The approach is based on the notion that, in order to achieve optimum performance, extant tools for sentence boundary detection must be presented with text that is free from markup. However, as described above, our goal is to obtain sentences with original markup preserved.

A naïve approach one might consider is to conceal markup with whitespace. Then, using a tool which produces character offsets, it would be possible to use such offsets to point back into the original text with markup. However, this is inadvisable because the superfluous whitespace could confound the model employed by the tool. Furthermore, many segmentation tools do not output character offsets, making it difficult to recover sentence positions in the original document.

Instead, our approach (depicted in Figure 2) involves creating a reduced form of the raw original text that is free from markup (♦1A) but replicates the appearance of the rendered document in terms of hard line breaks and paragraph breaks (as represented by double new lines). Since certain markup elements (such as paragraphs, headings and list items) can inform sentence boundary detection, at this stage we force segmentation by inserting paragraph breaks in place of certain sentence-forcing elements.[2] In the case of XML, some characters may be encoded as entities. These can also inform sentence segmentation, so we convert entities into their Unicode equivalents (for example, both `&8230;` and `&hellips;` are transformed to ‘...’). While making the reduction, we record a character-by-character alignment from the reduced form back to the raw (♦1B).

[2] As not all tools enforce sentence breaks at the end of paragraphs (Read et al., 2012a) we provide an optional *paragraph-mode*, wherein documents are split into paragraphs (as delimited by double new lines), which are presented separately to the external tool.

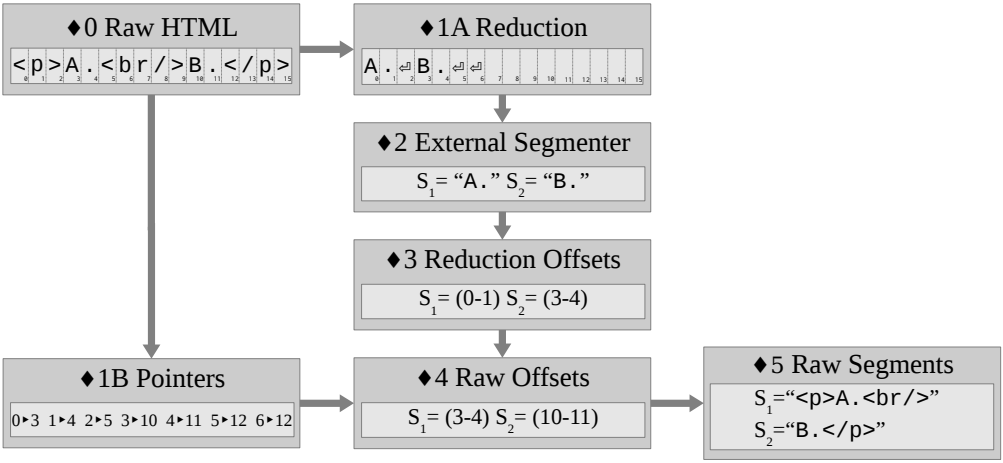

Figure 2: A generic approach for segmenting text with inline markup, with an HTML example.

In the next stage (♦2) one is free to run any tool for sentence boundary detection (assuming its output is mapped to our expectations of a single sentence on each new line). We then find the character offsets of the segments produced by the external tool (♦3), from which we obtain offsets in the raw form (♦4) by following the pointers collected in ♦1B.

Segments in the raw form can be easily obtained using these offsets. However the segments are not yet complete, because it is necessary to recover tags at beginning and end of each (♦5). In the case of HTML we achieve this heuristically—only opening tags should occur at the start of a segment and, conversely, only closing tags should occur at the end of a segment. Thus we expand to the left of a segment until we observe either text or a closing element, and expand to the right until we observe either text or an opening element.

With segmentation complete we can now carry out various normalising actions. As stated above, there is evidence that some elements of markup can be useful to downstream processing tools. For this reason, we provide the mechanism to re-incorporate the markup (optionally in a normalised form) that a user considers potentially relevant. Our intent is to provide a tool that is easily configurable according to the preferences of the user; thus we present a set of normalisation actions but do not make any assertions about how they should be applied:

Map For selected elements, we map the concrete markup element to a normalised equivalent. Grammar Markup Language (GML), described in Section 3, provides one means of normalisation and is the method we use in experimentation.

Strip In other cases one may be certain that the tag is not useful in downstream processing, hence we remove the tag but leave its contents in place.

Mask The textual contents of some markup elements are not easily obtainable or have little linguistic meaning. However, their presence still plays a structural role in the sentence (for

```
The name ⌊*Clanfield*⌋ is derived from the ⌊>Old English>⌋ and means
“field clean of weeds”.
Clanfield was historically a small ⌊>farming>⌋ community.
```

Figure 3: Our running example annotated using the Grammar Markup Language (GML), with new line characters to delimit sentences.

example, `code` containing a variable name). We propose replacing such elements with a placeholder (i.e. an empty element) in order to preserve sentence structure.

Purge For some applications one might be confident that certain elements are not useful for downstream processing (tables, for example). For such instances we provide a function to remove the elements and their contents. Note that for efficiency this function is carried out at ♦1A in Figure 2, as it would be wasted effort to attempt to segment content that is only removed later.

Finally we output an account of the segmentation process—a list of sentences with (optionally) normalised markup, each with a series of offsets and lengths that represent deletions and insertions made to the original document.

Note that all the behaviour described above is configurable. For instance, some websites use paragraph elements to force formatting rather than to indicate paragraphs, hence turning off this heuristic for paragraph elements could prevent the generation of spurious sentences.

3 Markup Normalisation

As mentioned previously, the properties of markup can significantly affect subsequent processing. For instance, (Flickinger et al., 2010) observe that italicisation frequently functions like quotation (for example, to indicate use – mention distinctions or foreign language text). To explore the interface between markup and grammar they proposed a Grammar Markup Language (GML) which abstracts away from concrete markup (Flickinger et al., 2010; Solberg, 2012). In this section, we embrace this general idea but propose a refined and extended scheme for this purpose, dubbed GML 2.0.

GML is designed to be a low-impact markup language that promotes compactness and human readability (particularly by those who are unfamiliar with rich mark-up languages such as XML). As such, elements are named with a single character, often using ‘unusual’ characters,[3] with names selected from Unicode’s extended sets in order to promote distinctiveness from the surrounding text. Figure 3 shows our running example with sentences delimited by new line characters, and contains instances of links (>) and bold text (*) as GML annotations. There are only three meta-characters: ⌊ (`LEFT FLOOR, U+230A`), ⌋ (`RIGHT FLOOR, U+230B`) and ¦ (`BROKEN BAR, U+00A6`), respectively serving as left, right and middle delimiters. A GML opening tag is composed of the left delimiter followed by the element name. Conversely the closing tag is the element name followed by the right delimiter. Empty elements are formed of the left delimiter, the element name and the right delimiter (e.g. an image is represented as ⌊✎⌋). Attributes of elements are separated using

[3] Using unusual characters means that GML does not require escaping of common ASCII characters (as is the case in, for example, XML), but good Unicode capabilities are a prerequisite for humans to author GML; however, we anticipate that GML texts will typically be produced automatically.

	Linux			NLP		
	Prec	**Rec**	**F_1**	**Prec**	**Rec**	**F_1**
(1) Markup in place	97.6	91.7	94.6	99.1	91.3	95.1
(2) Markup reduced	98.5	91.4	94.8	98.9	96.3	97.6

Table 1: The performance of two strategies for segmenting marked-up text; (1) segmenting with markup in place and (2), the approach described in this paper, reducing markup to plain text and reintroducing it through character alignment. Note that the gold-standard was derived by manually correcting the segmentation produced by strategy (1), potential biasing the evaluation in its favour.

the middle delimiter and are attached to the closing tag (e.g. second-level headings are specified as ⌊=Title¦2=⌋).

4 Evaluation and Error Analysis

To evaluate our approach we use text from the WeSearch Data Collection, a freely-available corpus of user-generated content (Read et al., 2012b). In this corpus, text was automatically segmented using the sentence splitter from the Stanford CoreNLP Tools, with the HTML-based sentence forcing heuristics described above, but with text being processed by the sentence splitter with HTML in place. Specifically, we use the two mixed source test sections from the blog domain, which are drawn from blogs about Linux (WLB03) and natural language processing (WNB03). The sentence segmentation in these sections was corrected by hand to create a gold-standard resulting in 1,130 sentences in WLB03 and 1,073 sentences in WNB03.

To measure segmentation performance we employ the evaluation script provided by Read et al. (2012a), which calculates precision, recall and F_1 by considering *every character*[4] in the text as a potential sentence boundary. Table 1 shows the performance of two methods when segmenting the raw HTML used to generate WLB03 and WNB03; row (1) gives the results of the original automatic procedure, while row (2) gives the results of the reduction approach described in this paper (when coupled with the Stanford CoreNLP sentence splitter). Note that the gold-standard was derived by manually-correcting the original automatic segmentation; this potentially biases the evaluation in favour of the original procedure.

We find that, for Linux blogs, the new approach results in a small improvement in precision and a minor loss in recall. The effect is quite different for natural language processing blogs. There is a strong improvement in segmentation, with a gain of 5% points in recall, with only a minor loss in precision. We performed an analysis of errors made by our approach, in order to acquire an understanding of how to improve the tool.

False negatives account for the largest proportion of the errors. Within these we note two dominating trends: non-standard lists, and non-standard typography—both of which could be considered issues of robustness when dealing with user-generated content. Over half the false negatives relate to character-based list conventions, where list items are not indicated with HTML but instead delimited with characters as bullets (e.g., '*', '-' or 'o'), or 'Q:' or 'A:'. Another quarter of the false negatives could more generally be considered aspects of informal English or author error. These include the lack of end-of-sentence punctuation, sentence-initial capitals or whitespace between sentences. While the list conventions could potentially be handled heuristically by our tool, more general

[4]Evaluation of sentence segmentation is often carried out using a small set of potential sentence boundaries (e.g. '!', '.' or '?'), but Read et al. (2012a) argue that this is an oversimplification, with these characters only appearing at the end of 92% of sentences in edited text such as The Wall Street Journal portion of the Penn Treebank.

robustness issues arise from the fact that the splitter appears to be optimised towards edited text. This is a well-known issue that is being dealt with in other tools (Foster et al., 2011). Similar techniques could be applied to sentence splitters, but that is not the focus of this tool.

A smaller proportion of the false negative errors do arise from markup-related issues. In particular, we note a few instances where placement of masked elements (e.g. <code/>) from repeated occurrences hide the end of a sentence. While some of these are obvious errors, others point to possible inconsistencies in the gold standard.

Of the smaller number of false positives, we again find that list conventions account for about half the erroneous instances. One regular pattern was enumerated lists where a full-stop is used to delimit the item from its number (e.g. 1.) which incorrectly led to segmentation after the number. Another trend related to the use of full-stops in bibliographic entries, which was specific to the natural language processing dataset and possibly needs specific handling.

Of greater concern from the perspective of our tool was that there were a few instances where the decision to introduce hard sentence breaks at paragraph and list elements led to false positives. While formally an incorrect use of HTML, occasionally authors will use these elements for their layout properties, rather than their abstract meaning. This could suggest that we should introduce a more nuanced handling of these elements, using heuristics to decide when to enforce breaks. However, in these datasets containing just over 2,000 sentences, this type of error only accounts for four false positives, indicating that the improvement this change would bring could well be offset by the false negatives that would likely occur.

5 Conclusions and Future Work

We have produced a simple tool that, by producing a plain text version that is aligned character-by-character, allows users to make use of previous plain text segmentation research and tools, while also using the *advantages* of the markup. Further, by maintaining accountability of any normalisation actions, this tool facilitates the projection of downstream processing back to the original source, enabling inline annotation in a browser, for example. In addition, by including different handling mechanisms for various aspects of the markup, we provide the option of passing on relevant normalised markup to downstream tools, and present Grammar Markup Language (GML) as a way of abstracting from various different concrete markup schemes.

Our main use of this tool to date has been for sentence segmentation of HTML documents, however it was deliberately designed to be suitable for generic text segmentation of multiple markup styles. Most of the markup handling specifics are defined in a simple external configuration file, but the heuristics for expanding segments to include enclosing markup are handled by a HTML-specific module. Ideally, one would learn the markup conventions that influence sentence segmentation. However, the lack of appropriate training data (and the likelihood of needing data for each new website or content management system) makes a statistical approach impractical. As our previous survey (Read et al., 2012a) indicated that rule-based systems perform best at this task, we have opted for that method in our wrapper.

The error analysis has revealed a number of interesting possibilities for improving the tool. Most significantly, almost half of the false negative errors made on the datasets were due to the use of character-based list conventions, without HTML markup. Such errors could perhaps be fixed simply by forcing breaks when observing sequences of newlines, whitespace and a bullet-like characters, potentially greatly improving the recall of sentence boundaries. Similarly, precision could be improved with a heuristic to remove sentence breaks after the item number of enumerated

lists. As mentioned above, there is also potential for improving precision by using soft constraints to decide when to introduce forced breaks at paragraph or list items, however the possibility of missing more sentence breaks in this fashion is a concern that should be tested empirically.

In other future work, we anticipate creating variants of this module for WikiText, and possibly for LaTeX. We intend to run experiments on tokenisation, and possibly other forms of pre-processing where markup could be distracting to off-the-shelf tools, to evaluate the impact of this form of normalisation more generally.

References

Flickinger, D., Oepen, S., and Ytrestøl, G. (2010). Wikiwoods: Syntacto-semantic annotation for English Wikipedia. In *Proceedings of the 7th Conference on International Language Resources and Evaluation*, Valletta, Malta.

Foster, J., Cetinoglu, O., Wagner, J., Le Roux, J., Nivre, J., Hogan, D., and van Genabith, J. (2011). From news to comment: Resources and benchmarks for parsing the language of Web 2.0. In *Proceedings of the 2011 International Joint Conference on Natural Language Processing*, page 893 – 901, Chiang Mai, Thailand.

Gimpel, K., Schneider, N., O'Connor, B., Das, D., Mills, D., Eisenstein, J., Heilman, M., Yogatama, D., Flanigan, J., and Smith, N. A. (2011). Part-of-speech tagging for Twitter: Annotation, features, and experiments. In *Proceedings of the 49th Meeting of the Association for Computational Linguistics*, page 42 – 47, Portland, OR, USA.

Kilgarriff, A. and Grefenstette, G. (2003). Introduction to the special issue on the web as corpus. *Computational Linguistics*, 29(3):333 – 347.

Marcus, M., Santorini, B., and Marcinkiewicz, M. A. (1993). Building a large annotated corpora of English: The Penn Treebank. *Computational Linguistics*, 19:313 – 330.

Read, J., Dridan, R., Oepen, S., and Solberg, L. J. (2012a). Sentence boundary detection: A long solved problem? In *Proceedings of the 24th International Conference on Computational Linguistics*, Mumbai, India.

Read, J., Flickinger, D., Dridan, R., Oepen, S., and Øvrelid, L. (2012b). The WeSearch Corpus, Treebank, and Treecache. A comprehensive sample of user-generated content. In *Proceedings of the 8th International Conference on Language Resources and Evaluation*, Istanbul, Turkey.

Schäfer, U., Kiefer, B., Spurk, C., Steffen, J., and Wang, R. (2011). The ACL Anthology Searchbench. In *Proceedings of the 49th Meeting of the Association for Computational Linguistics System Demonstrations*, page 7 – 13, Portland, OR, USA.

Solberg, L. J. (2012). A corpus builder for Wikipedia. Master's thesis, University of Oslo, Norway.