

Modeling Abbreviations and Punctuation for Sentence Segmentation

Phillip Potamites

December 16, 2007

The potential complexity of language processing demands that texts be divided into manageable sub-problems. Furthermore, understanding language arguably demands the identification of intended sub-units. Though ill-defined *theoretically*, sentences are *intuitively* privileged units of meaning. We might approximately characterize them as the largest units of syntax and the smallest units of discourse¹. Thus, accurate sentence boundary detection (SBD) is a prerequisite for proper interpretation, theoretical analysis, etc.

This notion of ‘prerequisite’, however, does not imply that SBD might not be fruitfully incorporated with other tasks. For instance, Mikheev (2000) incorporates SBD with overall part-of-speech tagging. Such techniques are probably important avenues of research, but the methods described here focus strictly on identifying abbreviations, punctuation, and boundaries.

Kiss and Strunk (2006) describe unsupervised methods for identifying abbreviations and then sentence boundaries. They base these decisions on 3 characteristics: a final period, shortness, and internal periods.

The system described here also uses general assumptions to identify abbreviation candidates, and additionally divides them into name or non-name abbreviations. It builds a tagging model of the data within an EM framework, which allows for automatic improvement of the match between the model and the data. This tagging model is constrained by definitions and generalizations regarding types of punctuation and abbreviations.

Most importantly, no training data of any kind is required to utilize this system. Only a source text of sufficient size is needed.

1 spaces

Instead of associating sentence boundaries with a label, ‘sb’, on items of punctuation, this system assumes that ‘sb’ is a property of spaces. By considering spaces, rather than punctuation, it is possible for the system to capture titles and other line-formatted breaks. These kinds of breaks, for chapters, etc., are rarely ended in punctuation, and thus, missed by punctuation-oriented ‘sb’-detectors.

Every computational system requires assumptions about the format of acceptable input. This system assumes that words and punctuation are line-delineated. Given standard conventions of hanging punctuation immediately onto words, while typically leaving a space behind, it is necessary

¹If either of those terms was well-defined.

to separate punctuation from its preceding word, but also essential (given the adopted space-orientation) to retain the following space. This preprocessing seeks to immediately take advantage of the fact that those punctuations which are *not* immediately followed by a space are *almost* never sentence boundaries.²

”	(1)
<i>My</i>	(2)
<i>God</i>	(3)
!	(4)
	(5)
<i>Mr</i>	(6)
.	(7)
	(8)
<i>Chace</i>	(9)
,	(10)
	(11)
<i>what</i>	(12)
<i>is</i>	(13)
<i>the</i>	(14)
<i>matter</i>	(15)
?	(16)
”	(17)
	(18)

While the system still needs to make decisions for the spaces at (5,8,11,18), the lack of sentence boundaries at (1,16) is taken for granted. Thus, some challenges are immediately eliminated by assumed punctuation conventions, the preprocessing treatment of spaces, and the space-oriented focus of this system.

2 model

The overall probability of a corpus (\vec{W}) can be defined (under independence assumptions) as the product of its components (\vec{w}):

²What to do with the exceptions remains an outstanding problem: Below are just a few examples of how (Gutenberg’s) Melville uses ‘-’ in a sentence- and word-level sense, as well as for citations, all without a trailing space:

- (1)
 - a. Here ye strike but splintered hearts together–there, ye shall strike unsplinterable glasses!
 - b. “Ay Ay, sir! There she blows! there–there–THAR she blows–bowes–bo-o-os!”
 - c. “Very like a whale.” –HAMLET.

The proper segmentation of the citation (c) is debatable, perhaps even unimportant, but the others appear to me clear-cut and important.

$$p(\vec{W}) = \prod_{\vec{w}} p(\vec{w}) \quad (19)$$

Assuming that \vec{w} are only the observable manifestation of a complex containing hidden information requires that we define $p(\vec{w})$ as a sum over any of these potential hidden properties (\vec{t}):

$$p(\vec{w}) = \sum_{\vec{t}} p(\vec{w}, \vec{t}) \quad (20)$$

Even with the help of dynamic methods, like the forward-backward algorithm, modeling all possible tags for an entire corpus can be time-consuming.³ Regarding sentence segmentation, without a sophisticated syntactic model, much of this modeling can also be considered irrelevant. In so far as this system restricts itself to tagging words as either normal or some kind of abbreviation, it seems reasonable to assume that only the 2 preceding items (perhaps ‘Mr’ and ‘.’), and the following item (if capitalized, etc.) are relevant to the sentence boundary decision. The extraneous material can be assumed to be identical under all models. However, some dangers and complications of *partially* modeling the data will be addressed below.

Thus, here, all relevant \vec{w} are 4-grams where the 3rd member is a blank space: $w_{i-2}, w_{i-1}, _i, w_{i+1}$. Possible tags are also categorized into sets which deterministically limit their application to different items:

- spaces:
 - ‘sb’: sentence breaks
 - ‘wb’: word breaks
- following possibilities:
 - ‘cap’: capitalized⁴
 - ‘ncap’: not capitalized
 - ‘spc’: another space
- preceding punctuation:
 - ‘cp’: a comma
 - ‘pp’: a sentence break period
 - ‘ap’: an abbreviation period
 - ‘op’: other punctuation
- preceding words:
 - ‘ww’: a normal word
 - ‘na’: name abbreviations

³My original implementations with trigram models took several days to run on the complete *Moby Dick*.

⁴This category includes space-following punctuation.

– ‘nna’: non-name abbreviations

For instance:

$$\begin{array}{ccccccc} \left\{ \begin{array}{c} ww \\ na \\ nna \end{array} \right\} & \left\{ \begin{array}{c} cp \\ pp \\ ap \\ op \end{array} \right\} & \left\{ \begin{array}{c} sb \\ wb \end{array} \right\} & \left\{ \begin{array}{c} cap \\ ncap \\ spc \end{array} \right\} & & & (21) \\ Mr & . & & Chace & & & \end{array}$$

To capture the relation between tag and word sequences, we can adopt a noisy channel model, where the crucial tag is conditioned on its potential neighbors, and the words are conditioned by the tags. By assumption, both ‘sb’ and ‘wb’ always result in blank spaces (i.e. $p=1$), and the properties of the space-following item can also be determined, with absolute certainty, via regular expression libraries. Although this determination does not give us the exact value of $p(w_{i+1}|t_{i+1})$, this value is fixed across our tag sequence candidates, and therefore can also be disregarded. Therefore, our modified noisy channel model can define the joint probability of the relevant word and tag sequences as follows:

$$p(\vec{w}, \vec{t}) = p(t_i|t_{i-2}, t_{i-1}, t_{i+1})p(w_{i-2}|t_{i-2})p(w_{i-1}|t_{i-1}) \quad (22)$$

Initially, we do not know any of these probabilities, but we can initialize them according to the constraints discussed in the next section.

According to the definition of conditional probabilities (cf. Bayes), they can also be defined, respectively, as follows:

$$p(t_i|t_{i-2}, t_{i-1}, t_{i+1}) = \frac{c(t_{i-2}, t_{i-1}, t_i, t_{i+1})}{c(t_{i-2}, t_{i-1}, t_{i+1})} \quad (23)$$

$$p(w_j|t_j) = \frac{c(w_j, t_j)}{c(t_j)} \quad (24)$$

Initially, we do not know these counts. However, according to the EM method of fractional counting, we can collect them by weighting our confidence in the proposed tag sequence. For each \vec{w} , we consider all possible \vec{t} , and for every relevant event included in (\vec{w}, \vec{t}) , we increment the count of that event by $p(\vec{t}|\vec{w})$:

$$+ = p(\vec{t}|\vec{w}) = \frac{p(\vec{w}, \vec{t})}{p(\vec{w})} = \frac{p(\vec{w}, \vec{t})}{\sum_{\vec{t}} p(\vec{w}, \vec{t})} \quad (25)$$

Thus, we can pass back and forth, from recalculating counts or probabilities as many times as is useful or necessary. The method of fractional counting is guaranteed to increase, if possible, the probability of the data according to the model.

3 initially

The initial probabilities lock in characteristics which are fundamental to sentence boundary decisions and the intended interpretations of various labels.

- commas are uniquely labeled ‘cp’:

$$p(x|cp) = \begin{cases} 1 & \text{if } x = ',' \\ 0 & \text{otherwise} \end{cases} \quad (26)$$

- pp (sb period) and ap (abbreviation period) only label periods⁵:

$$p(‘.’ |pp) = 1, p(‘.’ |ap) = 1 \quad (27)$$

- spaces followed by other spaces are definitely ‘sb’⁶:

$$p(wb_i | spc_{i+1}) = 0 \quad (28)$$

- Only words ‘always’ followed by a period (with a sufficient count of occurrences) can be abbreviations:

$$p(w_i | na \vee nna) = \begin{cases} p(w_i | \cdot_{i+1}) & \text{if } p(\cdot_{i+1} | w_i) > .99 \text{ and } c(w_i, \cdot_{i+1}) > 2 \\ 0 & \text{otherwise} \end{cases} \quad (29)$$

To avoid over-generating, the bigram count of word and period also have to be greater than 2. In fact, the name abbreviation candidates require a subsequent capital, as well as the period. These simple techniques eliminate abbreviation candidates with $p(\cdot_{i+1} | w_i) > .99$ just because they only appeared in the text once or twice at the end of a sentence. More sophisticated statistical methods, such as χ^2 -tests, or binomial tests (see Brent (1991)), could also be used to achieve the same goal.

Notice that these restraints need to be evaluated over the entire corpus, and not just on the 4-grams considered relevant to boundary decisions. The question of period co-occurrence must make use of the entire text, and not just space contexts. However, given that these are word rather than tag bigrams (plus the capital feature for the name abbreviation candidates), they can be evaluated on the initial read of the data, and do not require evaluation of tagging hypotheses. On the other hand, this is also why the abbreviation of ‘Danish’, ‘Dan.’, fails to be considered as an abbreviation candidate, since ‘Dan’ also appears as a name.

- name abbreviations are always followed by periods, spaces and capitals⁷:

$$p(na_i | \neg ap_{i+1} \vee \neg wb_{i+2} \vee \neg cap_{i+3}) = 0 \quad (30)$$

- non-name abbreviations are always followed by periods:

$$p(na_i | \neg ap_{i+1}) = 0 \quad (31)$$

⁵Notice that it is still possible for ‘op’ to label periods; such labeling actually becomes essential where there are non-abbreviations, periods, and non-caps.

⁶Recollect that the preprocessing only leaves a single punctuation-following space, and line break spaces. Therefore, a space sequence indicates a line break.

⁷Presumably, texts will not use sentences like “Excuse me, Mr.”

- non-name abbreviations with caps are ‘sb’:

$$p(na_i, ap_{i+1}, wb_{i+2}, cap_{i+3}) = 0 \tag{32}$$

- non-name abbreviations with non-caps are ‘wb’:

$$p(na_i, ap_{i+1}, sb_{i+2}, ncap_{i+3}) = 0 \tag{33}$$

- other punctuation with caps are ‘sb’:

$$p(op_i, wb_{i+1}, cap_{i+2}) = 0 \tag{34}$$

- other punctuation with non-caps are ‘wb’:

$$p(op_i, sb_{i+1}, ncap_{i+2}) = 0 \tag{35}$$

- for other words and labels: the probability of the word is assumed to be independent of the tag:

$$p(w|t) = p(w) \tag{36}$$

- for other tag sequences: the maximum possible tag sequence candidates, for 2 punctuations and with the final label determined equals $4 \times 4 \times 2 \times 1 = 32$.

$$p(xb|*, *, *) = 1/32 \tag{37}$$

For instance, from our example above, the emission constraints reduce the label candidates as follows:

$$\begin{matrix} \left\{ \begin{matrix} ww \\ na \\ nna \end{matrix} \right\} & \left\{ \begin{matrix} \cancel{pp} \\ pp \\ ap \\ op \end{matrix} \right\} & \left\{ \begin{matrix} sb \\ wb \end{matrix} \right\} & \left\{ \begin{matrix} cap \\ \cancel{ncap} \\ \cancel{ncap} \\ \cancel{ncap} \end{matrix} \right\} \\ Mr & . & & Chace \end{matrix} \tag{38}$$

Tag sequence constraints further reduce the potential tag sequence candidates. Then given that, for any potential abbreviation, w_i , $p(w_i|_{\cdot i+1})$ is greater than $p(w_i)$, the abbreviation interpretations will have a general advantage.

	first iteration		third iteration
ww pp sb cap	6.52e-06	ww pp sb cap	5.63e-08
ww op sb cap	1.63e-07	ww op sb cap	9.90e-10
na ap wb cap	0.00026	na ap wb cap	0.5943
nna ap sb cap	0.00026	nna ap sb cap	0.2345
total	0.000526	total	0.8288

Comparing the above 2 tables demonstrates how the label of ‘na’ for ‘Mr’, and the decision ‘wb’, in this case, gain probability over iterations of the EM algorithm.

4 accuracy

To evaluate the system's performance, we can compare it to both a simple rule-based procedure, and a state-of-the art segmenter such as MXTerminator.

The rule-based procedure can be summarized by the following lines:

```
Punc=['.',',','?', '!','"']
For each line in text:
    If line is blank (but not already preceded by a blank):
        Insert Sentence Boundary.
        Continue.
    For each character in line:
        If character is '"' and the next character is a letter: Continue.
        If the next character is in Punc: Continue.
        But if we made it here, and character is in Punc:
            Insert Sentence Boundary.
```

The above psuedo-code insures sentence breaks at line breaks and at all the members of Punc unless they are immediately followed by another Punc, or if it is an open-'"', rather than a close-'"'.

MXTerminator is a maximum entropy based segmenter designed by Jeffrey C. Reynar and Adwait Ratnaparkhi (see Reynar and Ratnaparkhi (1997)). The comparison, here, is a little unfair, in so far as it uses the original training settings of their system, while they stress the ease with which it can be trained for new domains. However, all systems are thus compared without the use of any target specific training.

In tables 1-3, the performance of each system is compared across 3 excerpts from *Moby Dick*. The first excerpt is the first 100 lines of chapter 1. This excerpt contains no abbreviations, and is primarily presented to demonstrate their best behavior, and their ability to just identify simple sentences. However, this section does contain Melville's rather wild use of '–', without spaces, sometimes immediately after appropriate sentence breaks, and also a non-capitalized sentence break like 'look! here come ...' The rule-based strict focus on punctuation isn't confused by these oddities, but both the other systems miss them.

The second excerpt, 'prologue', contains the very first 100 lines in *Moby Dick*, including the title, and a collection of quotations, with ellipses, name and language abbreviations, etc. (but not 'etc.'). MXTerminator stumbles over the non-punctuated breaks, as well as the prolific capitalization (of language names), which suggests breaks where they don't belong. The prolific capitalization also gets to the proposed tagging model, though it fairs better on the line breaks. The basic rule system, in its ignorant bliss, handles the line breaks and avoids being mislead by apparent cues.

The last comparison, 'lbs', is across a short section of text including a list of supplies, and the abbreviations 'Dr' and 'lbs'. Here, the massive number of abbreviations finally drags the knee-jerk method down, while both MXTerminator and this system do better, though capitalized proper product names following non-name abbreviations, like '60,000 lbs. Friesland pork', again mislead them.

While the system developed here does slightly worse, on the first test, it pulls ahead of (the pre-packaged training of) MXTerminator on the second test, and beats both alternatives on the third.

Table 1: Hand-Written

	ch.1(511-610)	prologue(1-100)	lbs(17934-81)
gold sentences	61	51	21
proposed sentences	60	61	37
correctly segmented	60	46	9
precision	1.0	.75	.24
recall	.98	.90	.43
f1-measure	.99	.82	.31

Table 2: MXTerminator

	ch.1(511-610)	prologue(1-100)	lbs(17934-81)
gold sentences	61	51	21
proposed sentences	59	53	26
correctly segmented	55	35	11
precision	.93	.66	.42
recall	.90	.69	.52
f1-measure	.92	.67	.47

Table 3: Tags

	ch.1(511-610)	prologue(1-100)	lbs(17934-81)
gold sentences	61	51	21
proposed sentences	59	62	23
correctly segmented	54	43	17
precision	.92	.69	.74
recall	.89	.84	.81
f1-measure	.90	.76	.77

5 abbreviations

The method adopted here also makes it possible to collect proposed types of abbreviations. Tables 4-5 show, respectively, the words which have been assigned either a name abbreviation label or a non-name abbreviation label, along with the counts of that assignment.

Melville also makes use of the name abbreviation ‘Rev’, but it only appears once in the entire text, so it is excluded from the abbreviation candidates as being insufficiently represented.

Items like ‘1839’ and ‘afresh’ show that ‘nna’ is still over-generating; however, the over-generation is not really damaging, in so far as subsequent capitals still elicit sentence boundary decisions.

Table 4: ‘na’ tag counts

Mr	63
St	24
Mrs	13
Dr	6

Table 5: ‘nna’ tag counts

II	14
III	11
lbs	7
V	4
E	4
IV	3
1839	3
IBID	3
afresh	3
etc	2

6 conclusion

The system described here may be summarized as follows. The system is informed of generalizations such as ‘abbreviations are always followed by periods’ or ‘name abbreviations are always followed by capitals’, as well as ‘never sentence break after a name abbreviation’ or ‘only sentence break after a non-name abbreviation, if the following word is capitalized’. Then the system analyzes the data and decides what are abbreviations and where sentence boundaries belong. The system has achieved almost state-of-the-art accuracy without any specially annotated training data; however, this success did require both pre-processing assumptions and numerous, effectively deterministic rules.

References

- Brent, Michael R. (1991) “Automatic Acquisition of Subcategorization Frames from Untagged Text,” in *Meeting of the Association for Computational Linguistics*, pp. 209–214.
- Kiss, T. and Strunk, J. (2006) “Unsupervised Multilingual Sentence Boundary Detection,” *Computational Linguistics*, 32(4), 485–525.
- Mikheev, Andrei (2000) “Tagging Sentence Boundaries,” in *ANLP*, pp. 264–271.
- Mikheev, Andrei (2002) “Periods, Capitalized Words, etc.” *Computational Linguistics*, 28(3), 289–318.

- Reynar, J. and Ratnaparkhi, A. (1997) “A Maximum Entropy Approach to Identifying Sentence Boundaries,” .
- Xu, J., Zens, R., and Ney, H. (2005) “Sentence Segmentation Using IBM Word Alignment Model 1,” in *Proceedings of the European Association for Machine Translation, 10th Annual Conference (EAMT)*, Budapest, Hungary.
- Zimmerman, M., Hakkani-Tur, D., Fung, J., Mirghafori, N., Gottlieb, L., Shriberg, E., and Liu, Y. (2006) “The ICSI+ Multilingual Sentence Segmentation System,” in *ICSLP*.