Sentence Boundary Detection and the Problem with the U.S.

Dan Gillick
Computer Science Division
University of California, Berkeley
dgillick@cs.berkeley.edu

Abstract

Sentence Boundary Detection is widely used but often with outdated tools. We discuss what makes it difficult, which features are relevant, and present a fully statistical system, now publicly available, that gives the best known error rate on a standard news corpus: Of some 27,000 examples, our system makes 67 errors, 23 involving the word "U.S."

1 Introduction

Many natural language processing tasks begin by identifying sentences, but due to the semantic ambiguity of the period, the sentence boundary detection (SBD) problem is non-trivial. While reported error rates are low, significant improvement is possible and potentially valuable. For example, since a single error can ruin an automatically generated summary, reducing the error rate from 1% to 0.25% reduces the rate of damaged 10-sentence summaries from 1 in 10 to 1 in 40. Better SBD may improve language models and sentence alignment as well.

SBD has been addressed only a few times in the literature, and each result points to the importance of developing lists of common abbreviations and sentence starters. Further, most practical implementations are not readily available (with one notable exception). Here, we present a fully statistical system that we argue benefits from avoiding manually constructed or tuned lists. We provide a detailed analysis of features, training variations, and errors, all of which are under-explicated in the literature, and discuss the possibility of a more structured classification approach. Our implementation gives the best performance, to our knowledge, reported on a standard Wall Street Journal task; it is open-source and available to the public.

2 Previous Work

We briefly outline the most important existing methods and cite error rates on a standard English data set, sections 03-06 of the Wall Street Journal (WSJ) corpus (Marcus et al., 1993), containing nearly 27,000 examples. Error rates are computed as $(\text{number incorrect}/\text{total ambiguous periods})$. Ambiguous periods are assumed to be those followed by white space or punctuation. Guessing the majority class gives a 26% baseline error rate.

A variety of systems use lists of hand-crafted regular expressions and abbreviations, notably Alembic (Aberdeen et al., 1995), which gives a 0.9% error rate. Such systems are highly specialized to language and genre.

The Satz system (Palmer and Hearst, 1997) achieves a 1.0% error rate using part-of-speech (POS) features as input to a neural net classifier (a decision tree gives similar results), trained on held-out WSJ data. Features were generated using a 5000-word lexicon and a list of 206 abbreviations. Another statistical system, mxTerminator (Reynar and Ratnaparkhi, 1997) employs simpler lexical features of the words to the left and right of the candidate period. Using a maximum entropy classifier trained on nearly 1 million words of additional WSJ data, they report a 1.2% error rate with an automatically generated abbreviation list and special corpus-specific abbreviation features.

There are two notable unsupervised systems. Punkt (Kiss and Strunk, 2006) uses a set of log-likelihood-based heuristics to infer abbreviations and common sentence starters from a large text corpus. Deriving these lists from the WSJ test data gives an error rate of 1.65%. Punkt is easily adaptable but requires a large (unlabeled) in-domain corpus for assembling statistics. An implementation is bundled with NLTK (Loper and Bird, 2002). (Mikheev, 2002) describes a "document-

Proceedings of NAACL HLT 2009: Short Papers, pages 241–244,
Boulder, Colorado, June 2009. ©2009 Association for Computational Linguistics

centered" approach to SBD, using a set of heuristics to guess which words correspond to abbreviations and names. Adding carefully tuned lists from an extra news corpus gives an error rate of 0.45%, though this increases to 1.41% without the abbreviation list. Combining with a supervised POS-based system gives the best reported error rate on this task: 0.31%.

Our system is closest in spirit to mxTerminator, and we use the same training and test data in our experiments to aid comparison.

3 Our Approach

Each example takes the general form "$L.\ R$", where L is the context on the left side of the period in question, and R is the context on the right (we use only one word token of context on each side). We are interested in the probability of the binary sentence boundary class s, conditional on its context: $P(s|$"$L.\ R$"$)$. We take a supervised learning approach, extracting features from "$L.\ R$".

Table 1 lists our features and their performance, using a Support Vector Machine (SVM) with a linear kernel[1]. Feature 1 by itself, the token ending with the candidate period, gives surprisingly good performance, and the combination of 1 and 2 outperforms nearly all documented systems. While no published result uses an SVM, we note that a simple Naive Bayes classifier gives an error rate of 1.05% (also considerably better than mxTerminator), suggesting that the choice of classifier alone does not explain the performance gap.

There are a few possible explanations. First, proper tokenization is key. While there is not room to catalog our tokenizer rules, we note that both untokenized text and mismatched train-test tokenization can increase the error rate by a factor of 2.

Second, poor feature choices can hurt classification. In particular, adding a feature that matches a list of abbreviations can increase the error rate; using the list ("Mr.", "Co.") increases the number of errors by up to 25% in our experiments. This is because some abbreviations end sentences often, and others do not. In the test data, 0 of 1866 instances of "Mr." end a sentence, compared to 24 of 86 instances of "Calif." (see Table 2). While there may be meaningful abbreviation subclasses, a feature indicating mere presence is too coarse.

#	Feature Description	Error
1	$L = w_i$	1.88%
2	$R = w_j$	9.36%
3	$len(L) = l$	9.12%
4	$is\_cap(R)$	12.56%
5	$int(log(count(L; \text{no period}))) = c_i$	12.14%
6	$int(log(count(R; \text{is lower})) = c_j$	18.79%
7	$(L = w_i, R = w_j)$	10.01%
8	$(L = w_i, is\_cap(R))$	7.54%
1+2		0.77%
1+2+3+4		0.36%
1+2+3+4+5+6		0.32%
1+2+3+4+5+6+7+8		0.25%

Table 1: All features are binary. SVM classification results shown; Naive Bayes gives 0.35% error rate with all features.

Abbr.	Ends Sentence	Total	Ratio
Inc.	109	683	0.16
Co.	80	566	0.14
Corp.	67	699	0.10
U.S.	45	800	0.06
Calif.	24	86	0.28
Ltd.	23	112	0.21

Table 2: The abbreviations appearing most often as sentence boundaries. These top 6 account for 80% of sentence-ending abbreviations in the test set, though only 5% of all abbreviations.

Adding features 3 and 4 better than cuts the remaining errors in half. These can be seen as a kind of smoothing for sparser token features 1 and 2. Feature 3, the length of the left token, is a reasonable proxy for the abbreviation class (mean abbreviation length is 2.6, compared to 6.1 for non-abbreviation sentence enders). The capitalization of the right token, feature 4, is a proxy for a sentence starter. Every new sentence that starts with a word (as opposed to a number or punctuation) is capitalized, but 70% of words following abbreviations are also, so this feature is mostly valuable in combination.

While we train on nearly 1 million words, most of these are ignored because our features are extracted only near possible sentence boundaries. Consider the fragment "... the U.S. Apparently some ...",

[1] We use SVM Light, with $c = 1$ (Joachims, 1999). Non-linear kernels did not improve performance in our experiments.

which our system fails to split after "U.S." The word "Apparently" starts only 8 sentences in the training data, but since it usually appears lowercased (89 times in training), its capitalization here is meaningful. Feature 6 encodes this idea, indicating the log count of lowercased appearances of the word right of the candidate period. Similarly, feature 5 gives the log count of occurrences of the token left of the candidate appearing without a final period.

Another way to incorporate all of the training data is to build a model of $P(s|\text{“}L\ R\text{”})$, as is often used in sentence segmentation for speech recognition. Without a period in the conditional, many more negative examples are included. The resulting SVM model is very good at placing periods given input text without them (0.31% error rate), but when limiting the input to examples with ambiguous periods, the error rate is not competitive with our original model (1.45%).

Features 7 and 8 are added to model the nuances of abbreviations at sentence boundaries, helping to reduce errors involving the examples in Table 2.

4 Two Classes or Three?

SBD has always been treated as a binary classification problem, but there are really three classes: sentence boundary only (S); abbreviation only (A); abbreviation at sentence boundary ($A + S$). The label space of the test data, which has all periods annotated, is shown in Figure 1.

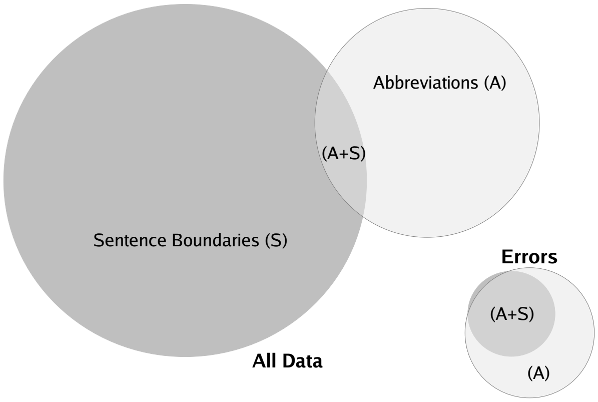

Figure 1: The overlapping label space of the test data: sentence boundaries 74%; abbreviations 26%; intersection 2%. The distribution of errors given by our classifier is shown as well (not to scale with all data).

Relative to the size of the classes, $A + S$ examples are responsible for a disproportionate number of errors, pointing towards the problem with a binary classifier: In the absence of $A + S$ examples, the left context L and the right context R both help distinguish S from A. But $A + S$ cases have L resembling the A class and R resembling the S class.

One possibility is to add a third class, but this does not improve results, probably because we have so few $A + S$ examples. We also tried taking a more structured approach, depicted in Figure 2, but this too fails to improve performance, mostly because the first step, identifying abbreviations without the right context, is too hard. Certainly, the $A+S$ cases are more difficult to identify, but perhaps some better structured approach could reduce the error rate further.

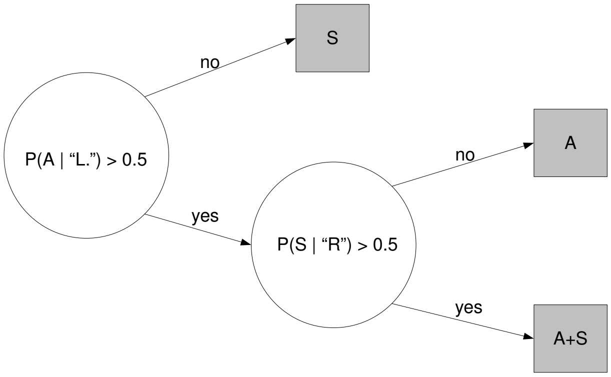

Figure 2: A structured classification approach. The left context is used to separate S examples first, then those remaining are classified as either A or $A + S$ using the right context.

5 Training Data

One common objection to supervised SBD systems is an observation in (Reynar and Ratnaparkhi, 1997), that training data and test data must be a good match, limiting the applicability of a model trained from a specific genre. Table 3 shows respectable error rates for two quite different test sets: The Brown corpus includes 500 documents, distributed across 15 genres roughly representative of all published English; The Complete Works of Edgar Allen Poe includes an introduction, prose, and poetry.

A second issue is a lack of labeled data, especially in languages besides English. Table 4 shows that results can be quite good without extensive labeled resources, and they are likely to continue to improve if additional resources were available. At the least, (Kiss and Strunk, 2006) have labeled over

Corpus	Examples	in S	SVM Err	NB Err
WSJ	26977	74%	0.25%	0.35%
Brown	53688	91%	0.36%	0.45%
Poe	11249	95%	0.52%	0.44%

Table 3: SVM and Naive Bayes classification error rates on different corpora using a model trained from a disjoint WSJ data set.

10000 sentences in each of 11 languages, though we have not experimented with this data.

Corpus	5	50	500	5000	42317
WSJ	7.26%	3.57%	1.36%	0.52%	0.25%
Brown	5.65%	4.46%	1.65%	0.74%	0.36%
Poe	4.01%	2.68%	2.22%	0.98%	0.52%

Table 4: SVM error rates on the test corpora, using models built from different numbers of training sentences.

We also tried to improve results using a standard bootstrapping method. Our WSJ-trained model was used to annotate 100 million words of New York Times data from the AQUAINT corpus, and we included high-confidence examples in a new training set. This did not degrade test error, nor did it improve it.

6 Errors

Our system makes 67 errors out of 26977 examples on the WSJ test set; a representative few are shown in Table 5. 34% of the errors involve the word "U.S." which distinguishes itself as the most difficult of tokens to classify: Not only does it appear frequently as a sentence boundary, but even when it does not, the next word is often capitalized ("U.S. Government"; "U.S. Commission"), further confusing the classifier. In fact, abbreviations for places, including "U.K.", "N.Y.", "Pa." constitute 46% of all errors for the same reason. Most of the remaining errors involve abbreviations like those in Table 2, and all are quite difficult for a human to resolve without more context. Designing features to exploit additional context might help, but could require parsing.

Context	Label	$P(S)$
... the U.S. Amoco already ...	$A+S$	0.45
... the U.K. Panel on ...	A	0.57
... the U.S. Prudential Insurance ...	$A+S$	0.44
... Telephone Corp. President Haruo ...	A	0.73
... Wright Jr. Room ...	A	0.67
... 6 p.m. Travelers who ...	$A+S$	0.44

Table 5: Sample errors with the probability of being in the S class assigned by the SVM.

7 Conclusion

We have described a simple yet powerful method for SBD. While we have not tested models in languages other than English, we are providing the code and our models, complete with tokenization, available at *http://code.google.com/p/splitta*. Future work includes further experiments with structured classification to treat the three classes appropriately.

Acknowledgments

Thanks to Benoit Favre, Dilek Hakkani-Tür, Kofi Boakye, Marcel Paret, James Jacobus, and Larry Gillick for helpful discussions.

References

J. Aberdeen, J. Burger, D. Day, L. Hirschman, P. Robinson, and M. Vilain. 1995. MITRE: description of the Alembic system used for MUC-6. In *Proceedings of the 6th conference on Message understanding*, pages 141–155. Association for Computational Linguistics Morristown, NJ, USA.

T. Joachims. 1999. Making large-scale support vector machine learning practical, Advances in kernel methods: support vector learning.

T. Kiss and J. Strunk. 2006. Unsupervised Multilingual Sentence Boundary Detection. *Computational Linguistics*, 32(4):485–525.

E. Loper and S. Bird. 2002. NLTK: The Natural Language Toolkit. In *Proceedings of the ACL Workshop on Effective Tools and Methodologies for Teaching Natural Language Processing and Computational Linguistics*, pages 62–69.

M.P. Marcus, M.A. Marcinkiewicz, and B. Santorini. 1993. Building a large annotated corpus of English: the penn treebank. *Computational Linguistics*, 19(2):313–330.

A. Mikheev. 2002. Periods, Capitalized Words, etc. *Computational Linguistics*, 28(3):289–318.

D.D. Palmer and M.A. Hearst. 1997. Adaptive Multilingual Sentence Boundary Disambiguation. *Computational Linguistics*, 23(2):241–267.

J.C. Reynar and A. Ratnaparkhi. 1997. A maximum entropy approach to identifying sentence boundaries. In *Proceedings of the Fifth Conference on Applied Natural Language Processing*, pages 16–19.