Terminus: An Ada Boosted Sentence Splitter

Anonymous ACL submission

Abstract

In this project, we sought out to create an improved automated sentence boundary detection (SBD) system for English. We reviewed the intuitive, manual-rule based system that detects a sentence boundary based on the presence of a "." symbol. To study such a system, we developed three baseline systems to examine three manual rules for splitting. We developed a system, using the machine learning technique Ada boosting and natural language processing techniques, to improve the system.

1 Introduction

Sentence Boundary Detection (SBD) isn't usually acknowledged as one of NLP's great issues. To add, it has been considered a more or less solved problem since the 1990s as per formal-texts. Despite the fact that there have been very few studies on SBD in recent decades, the evaluation of existing approaches for English is hampered by differences in task specification, evaluation criteria, and test data used. Furthermore, recent developments in NLP pose new challenges for SBD as state-of-the-art systems failing to perform well on social media and more spontaneous language samples, such as Web content. The growing and ever-complicating norms of the English language have made the age-old rules of splitting a sentence based on the presence of '.' neither appropriate nor efficient to the process of SBD. It is important to reiterate the idea that errors caused during SBD formidably affect the subsequent processes of NLP applications such as Information Retrieval, Speech Recognition, Document Summarization, Machine Translation, and Sentiment Analysis. Hence, it becomes important to revisit the manual rule-based system set in place to detect SBD.

Previous research in the area has been confined to formal texts only, and either has not addressed the process of SBD directly (Brill, 1994; Collins, 1996), or not the performance related issues of sentence boundary detection (Cutting et al., 1992).The impact of such textual variance on SBD has been little studied, and off-the-shelf methods may struggle with material that isn't especially newswire-like, i.e. not from the Penn Treebank's legendary Wall Street Journal (WSJ) collection (Marcus et al., 1993). In this paper, we aim to present an up-to-date assessment of the state of the art in SBD with the help of machine learning techniques.

This article begins with a brief explanation of our baseline model which examines whether or not we can build on efficient sentence boundary detection system with simple rules. This also serves as an examination to see how difficult is the problem. Next, with the help Ada boosting we develop a machine learning algorithm capable of adapting to complex rules and perform an analysis of our efficiency on the WSJ data set and the SJVMK data sets (chosen to study sentence boundaries with indexes).

2 Baseline Systems

2.1 Baseline One: splitting on "." symbol

This system predicted the end of a sentence at the position of every "." character observed in the text. This was the most simple and intuitive rule for splitting a sentence in English, since every English sentence ends with a ".". We observed an accuracy of 98%, a precision

of 51%, a recall of 74%, and an f-score of 57%. We determined that the high accuracy was due to the overwhelming amount of true negatives, which commonly occur where there is no "." character and no sentence boundary. We determined that the lower precision, recall, and ultimately f-score, were due to the high amount of false positives, which often occur due to the presence of a "." symbol without a coinciding sentence boundary.

2.2 Baseline Two: splitting on regular expression: '. [A-Z]'

A highly common pattern for a sentence boundary in English is the presence of a period, followed by a space, followed by an upper case letter. We implemented baseline system two with this in mind. Baseline system two predicted a sentence boundary at the position of the regular expression ". [A-Z]". We observed an accuracy of 99%, a precision of 51%, a recall of 38%, and an f-score of 43%. We determined that the relative decrease in f-score compared to baseline system one was caused by an overly restrictive rule that omitted sentence boundaries that did not fit the regular expression's pattern, such as when a quoted sentence ends with '." [A-Z]'.

2.3 Baseline Three: splitting on a ";" symbol

This system predicted the end of a sentence at the position of every ";" character observed in the text. We implemented this system in order to gain more insight on splitting based on a character, especially where we expected poor results. We observed an accuracy of 99%, a precision of 0.9%, a recall of 0.07%, and an f-score of 0.1%. This helped to confirm for us that accuracy is not a metric of evaluation suitable for our task.

2.4 Lessons from Baseline Systems

After observing our baseline systems, we decided that it would be both useful and possible to implement an advanced sentence splitting system that went beyond manual rules. We noted that, while the presence of a period character often denotes the end of a sentence, there is enough variability of sentence boundaries in the English language to warrant the use of machine learning and natural language processing techniques.

For the sake of comparability of results with our final system, we added more manual rules to Baseline System One, since that was the highest performing Baseline. These manual rules are also in the ada boosted system. These manual rules include that if a common abbreviation (such as "Mr.", "Mrs.", "Dr.", "Ms.", "Inc") is present, a sentence boundary is not predicted. Further, if the most recent word is capitalized, a sentence boundary is not predicted. With these features, the results on the WSJ data set was precision of 0.81, recall of 0.90, fscore of 0.85, and accuracy of 0.99.

3 Ada boosted sentence splitter

Tree based algorithms have been shown to work well when it comes to sentence splitting, but previous works often relied on a single decision tree. A single decision tree, however, is prone to many problems, including over fitting, imbalance bias, and inadequate prediction power when many features are present. Ada boosting on the other hand handles the above problems well and often provides very well classification compared to other machine learning models. Therefore, we have decided to use Ada boosting to tackle sentence boundary detection.

3.1 Evaluation

The Ada boosting algorithm creates N decision stumps, (1 depth decision trees) each with their associated positive or negative voting power. To make a decision, each stump votes on whether or not the potential position is a sentence boundary. For each stump, their voting power is summed based on whether they voted yes or no. The decision of the entire ensemble is based on which side has the majority vote.

Each stump can only make a prediction based on one feature, making them weak learners. The stumps and their corresponding voting

power are calculated based on previous mistakes. The ensemble of these weak learners allows the entire system to be a strong learner.

3.2 Training

The core of Ada boosting is how to generate each stump. Suppose we have M training data and we want to create N stumps. Here is the steps of the algorithm:

1. Assign initial weight to each datum. The initial weight should be $1/M$.

2. Calculate the Gini Impurity Index for all threshold of all the features.

3. Pick the threshold with the lowest Gini Impurity Index (the threshold with the least amount of impurity) and make it into a decision stump.

4. Calculate the amount of error and voting power for the stump.

5. If the amount of error reached a predetermined threshold, training is complete. Else, reassign the weight of each datum based on whether or not the datum was correctly classified by the stump and the voting power of the stump.

6. Normalize the weight.

7. Bootstrap the data set where data with higher weights are proportionally more likely to be drawn. This is now our new data set and all subsequent training will be done on this data set.

8. If the number of stump is reached, training is complete. Else repeat from step 1.

Note that these procedures allow stumps to learn from the mistakes of previous stumps by putting more emphasis on wrongly labeled data.

3.3 Gini Impurity Index

The Gini Impurity Index measures the impurity of a node, where higher the index, higher the impurity. The index is calculated by the following formula:

$$Gini = 1 - \sum_{i=1}^{n} (p_i)^2$$

Where n is the number of classes and p_i is the probability samples belonging to class i (the number of data in class i divided the total amount of data).To calculate the Gini Impurity of a threshold, we can calculate the weighted average of each node. The threshold is created using the following formula:

$$Gini_t = \sum_{k=1}^{m} \frac{n_k}{n} Gini(D_k)$$

Where k is the number of nodes the threshold creates, n_k is the number of data each node holds, n is the total number of data, and $Gini(D_k)$ is the Gini Impurity index of the node.

In the binary class case (which is the case for SBD), the index for each node is simplified to:

$$Gini = 1 - (d^2 - e^2)$$

and the index for each threshold simplifies to:

$$Gini = \frac{n_l}{n} Gini(D_l) + \frac{n_h}{n} Gini(D_h)$$

Here, D_l is the node for datum lower than the threshold, and D_h is the node for datum higher than the threshold.

3.4 Error and Voting power

The error is simply the sum of all the mislabeled data. Note that due to the weight, the total error is bounded between 0 and 1 inclusive. The voting power is calculated through the following formula:

$$\alpha_t = \frac{1}{2} \ln \left(\frac{1 - \epsilon_t}{\epsilon_t} \right)$$

3.5 Weight changing

To ensure that next stumps focus more on wrongly labeled data, we must increase the weight of the stumps with wrongly labeled data according to the voting power of the current stump. The change of the weight is given by this formula:

$$correctWeight = w * e^{-a}$$

$$incorrectWeight = w * e^{a}$$

Note that we adjust the weight such that they decrease while increasing the weight of incorrectly labeled data. This allows incorrectly labeled data to be more likely to be drawn in the bootstrap process.

3.6 Bootstrapping and re-normalization

We bootstrap by randomly drawing data with replacement. The probability that each datum getting drawn is proportionally based on their weight such that datum with higher weights will be more likely to be chosen multiple times. This allows the next weak learner to adapt to mistakes from the previous decision stump.

After all the new weights are assigned, it is important to re-normalize the weights to $1/M$ since each datum are back to equal importance.

3.7 Implementation Notes

In order to not approach precision boundary of floating point number, all weights are normalized toward 1000 rather than 1 due to the large data size.

4 Algorithm Features

4.1 Part of Speech Unigram

In order to provide contextual information around potential sentence boundary, we created a unigram distribution of parts of speech based on the specific corpus. While it would be ideal to use a proper POS tagger, most POS tagger require sentence boundary information in the first place. Therefore, an approximation is used. Provided the POS information through a unigram distribution, the system will determine which POS tag a word can be.

Due to large variety of POS tags, including all POS tag would make training our system unrealistically long. Therefore, similar tags are mapped to the same tag. For example: all forms of Adjective and Adverb and grouped together as "modifier." The specific grouping can be configured and differs based on different texts.

4.2 Positions considered

According to Jonathon Read's paper in 2012, sentences end on periods only about 87.7% of the time (Reed 2012). Therefore, it is not enough to only consider periods as potential boundary for a sentence. Instead, all symbols (including escape characters depending on the corpus) are considered possible sentence boundaries. Our system does not tag the start of a sentence, as we will assume that the start of the next sentence is immediately after the end of the current sentence. While this assumption is not perfect, it works well for most corpora with minor inconveniences of leading spaces or escape characters.

4.3 Standard Features

Standard obvious features are given to words around the potential boundary position. These features are length of the word, whether or not the word is capitalized, whether or not the word contains a number, whether or not the word contains a non-alphanumeric character, etc. In addition, the system is capable of recognizing the most common abbreviations in English and uses it as a feature.

4.4 Encoded Feature

Specific features are encoded into to binary vectors - for example, which specific symbol the current potential boundary is. Similarly, POS information of the surrounding words are encoded into a binary vector to provide contextual information. Due to the limit of the computational equipment available to us, this proves to be quite a nuance as an increase in POS information corresponds to an exponential increase in the number of features.

4.5 Manual Rules

A few manual rules are coded in the system to handle commonly miss-classified cases or cases that have no exception. For example: it is impossible for a sentence to end on an opening quote or a comma.

5 Methodology

5.1 Data Set Characteristics

We used the WSJ and SJVMK data sets to test our system. The WSJ data set contains well formed sentences that have already been tokenized while SJVMK contains raw text with the sentence boundaries labeled with indexes. Tokenizing of the SJVMK data set is done through a simple space separation (consecutive spaces that have equal or more than 3 spaces are considered special characters). Furthermore, consecutive newline characters (particularly $\backslash r \backslash n$) are also represented as 1 newline character instead.

5.2 Standard for Data Sets

The data sets are split into development and testing set. The development set is further split into a training set and testing set. The training set is used to train the model and efficiency is calculated on the testing set. For data set with POS information, we derive the POS unigram distribution information from the development training set.

All metrics for efficiency are done on the testing set. The testing set is not used at all during the development process in order to make sure that the system is not in anyway tune towards it.

5.3 Task Definition and Evaluation

Since sentence boundary detection is extremely similar to tagging tasks, we are evaluating it using the same metrics. The results are constructed into confusion matrices. Accuracy, precision, recall, and f-score are used to measure the efficiency of the system.

5.4 Efficiency Analysis

Table 1 shows the performance of our algorithm on different corpora. We can see that the system out performs all the baseline systems in terms of recall, precision, and f-score.

Corpus	WSJ	SJVMK
Recall	0.980	0.706
Precision	0.971	0.814
F-score	0.975	0.756
Accuracy	0.987	0.883

Table 1: This table shows the various score Ada boost algorithm achieved on testing corpora

System	Score
CoreNLP	91.3
LingPipe_1	97.3
LingPipe_2	88.0
MxTerminator	97.4
OpenNLP	99.1
Punkt	98.3
RASP	99.0
Splitta	99.2
tokenizer	97.9

Table 2: This table shows the various f-scores of other systems on the WSJ corpus

5.5 Comparing other systems

Many previous efforts on sentence boundary selection is based on period disambiguation. As shown in Jonathon Read's paper in 2012, this is not enough to detect a significant amount of sentence boundaries (Reed, 2021). In addition, many systems do not explicitly state what they consider to be potential positions for a sentence boundary. Depending on the number of potential sentence boundaries, the various metrics – especially accuracy – are effected by large margins.

Note that while some systems performs better, many are also tuned specifically for the WSJ corpus.

5.6 Implementation problems

Many problems in development came from the data sets rather than the algorithm itself. Specially, the SJVMK data set posed a problem with its unique formatting. Unlike the WSJ corpus, sentence boundaries are placed at spaces after a period, or, in cases where there is no

space, on the symbol itself. This inconsistency is one of the main reasons the system struggles on this particular data set. While a simple solution would be to include spaces as consideration for sentence boundaries, this would cause an exponential increase in the amount of training data. Due to the limited computation equipment available during development, we opted to accept this inconsistency. As a result, we disagree with around 7% of the annotations from a sample gathered.

Another problem was the encoding of the file. While WSJ could be read in ASCII, SJVMK includes Unicode characters. Two types of solutions were tried. One simply replaces all Unicode character with 1 common special ASCII character. This solution could not fully take advantage of the variety of meanings of Unicode characters. Another solution was mapping special Unicode characters to unique features. This proved to give out better performance due to its ability to represent meaning of these characters.

Finally, SJVMK does not assume that the end of a sentence marks the start of the next sentence. This is to deal with formatting issues, for example consecutive newline characters, space characters, and tab characters. Therefore, we had to merge consecutive formatting characters. In cases where this was not possible, all characters between the end of the previous sentence and the start of the next sentence were marked as end of sentences (making them 1 character sentences).

6 Conclusion and Future work

In this project, we sought out to creat an improved automated sentence boundary detection (SBD) system for English. We reviewed the intuitive, manual-rule based system that detects a sentence boundary based on the presence of a "." symbol. To study such a system, we developed three baseline systems to examine three manual rules for splitting. The highest performing baseline system achieved an f-score of roughly 85% at best (which was when used on the WSJ corpus). We developed a system, using the machine learning technique Ada boosting and natural language processing techniques, in order to improve this score. Our system achieved an f-score of approximately 97.5%.

For the future, we wish to add more complex techniques to guessing the POS tag of a token. Furthermore, it has been shown that Ada boosting works even better with deeper weak learners (trees of more than 1 level).

Acknowledgements

We would like to thank all the members of the NYU Computer Science Department and Professor Adam Meyers for making their data sets and tools available. Special thanks to the anonymous reviewers for their comments and suggestions that have helped to improve the paper.

References

Eric Brill. 1994. Some advances in transformation- based part of speech tagging. In Proceedings of the Twelfth National Conference on Artificial Intel- ligence (Vol. 1), AAAI '94, pages 722–727, Menlo Park, CA, USA. American Association for Artificial Intelligence.

Michael John Collins. 1996. A new statistical parser based on bigram lexical dependencies. In Proceed- ings of the 34th Annual Meeting on Association for Computational Linguistics, ACL '96, pages 184– 191, Stroudsburg, PA, USA. Association for Computational Linguistics.

Mitchell P. Marcus, Beatrice Santorini, and Mary Ann Marcinkiewicz. 1993. Building a Large Annotated Corpus of English: The Penn Treebank. Computational Linguistics, 19(2):313–330.

Jonathon Read, Rebecca Dridan, Stephan Oepen, and Lars Jørgen Solberg. 2012. Sentence Boundary Detection: A Long Solved Problem?. In Proceedings of COLING 2012: Posters, pages 985–994, Mumbai, India. The

COLING 2012 Organizing Committee.

George Sanchez. 2019. Sentence Boundary Detection in Legal Text. In Proceedings of the Natural Legal Language Processing Workshop 2019, pages 31–38, Minneapolis, Minnesota. Association for Computational Linguistics.

Dwijen Rudrapal, Anupam Jamatia, Kunal Chakma, Amitava Das, and Björn Gambäck. 2015. Sentence Boundary Detection for Social Media Text. In Proceedings of the 12th International Conference on Natural Language Processing, pages 254–260, Trivandrum, India. NLP Association of India.