HITS-SBD at the FinSBD Task: Machine Learning vs. Rule-based Sentence Boundary Detection

Mehwish Fatima[1] and **Mark-Christoph Müller**[2]
[1,2]HITS gGmbH, Heidelberg, Germany
{mehwish.fatima, mark-christoph.mueller}@h-its.org

Abstract

This paper presents two different approaches towards Sentence Boundary Detection (SBD) that were submitted to the FinSBD-2019 shared task. The first is a supervised machine learning approach which tackled the SBD task as a combination of binary classifications based on TF-IDF representations of context windows. The second approach is unsupervised and rule-based and applies manually created heuristics to automatically annotated input. Since the latter approach yielded better results on the Dev set, we submitted it to evaluation for English and reached an F score of 0.80 and 0.86 for detecting begin of sentences and end of sentences, respectively.

1 Introduction

Sentences are the fundamental units of text, consisting of words and punctuation, and constructing phrases and paragraphs [Reynar and Ratnaparkhi, 1997]. Sentence Boundary Detection (SBD), or finding the start and end of sentences, is an essential prerequisite in the Natural Language Processing (NLP) pipeline for various applications, such as Discourse Parsing [Polanyi *et al.*, 2004], Machine Translation, Document Summarization, Alignment of Parallel Text, Sentiment Analysis, and Information Retrieval [Jonathon *et al.*, 2012]. SBD can have a strong impact on the performance of these applications due to error propagation in the NLP pipeline.

Probably the most important factors in SBD are 1) ambiguous expressions and 2) source of text. In disambiguating sentence boundaries, the most misleading expression is the period (.), which is not only used as end of sentence (*ES*) marker, but also in abbreviations or numerical expressions (e.g. ordinals and dates). Some other examples of problematic expressions are question mark (?), exclamation mark (!), and colon (:). The other key factor is the genre and / or quality of the text, which can impact the performance of an SBD system. Most of the existing SBD systems are highly accurate on formal and high quality text, but their performance often degrades when the input text is noisy or informal.

The research work for SBD mostly focuses on disambiguating the ends of sentences in formal, noise-free text [Agarwal *et al.*, 2005; Kiss and Strunk, 2006; Akita *et al.*, 2006; Gillick, 2009; Rudrapal *et al.*, 2015]. The FinSBD-2019 Shared Task, in contrast, tackles SBD in noisy text extracted from PDFs from the financial domain.

We applied two different approaches to solve the SBD problem. The first approach treats the problem as an supervised classification task on TF-IDF based representations of context windows, thus taking advantage of the annotated training data set. The second approach is unsupervised, rule-based and applies manually created heuristics to automatically annotated input.

The rest of the paper is structured as follows: Section 2 covers some existing work on SBD. Section 3 explains the general configurations for experiments, including details about the data set and evaluation measures. Sections 4 and 5 explain the two approaches in more detail, and Section 6 concludes the paper.

2 Related Work

This section covers some studies that used rule- or machine learning-based approaches for SBD.

[Agarwal *et al.*, 2005] applied a combination of rule-based features and a probability based classifier (MaxEnt) to predict sentence ending boundaries. They considered three different data sets: 1) Wall Street Journal (WSJ) ($43,948$ sentences), 2) Penn Treebank ($24,243$ sentences) and 3) POS-labelled GENIA ($20,544$ sentences). The feature extraction was executed on each context trigram of data set where each context trigram contained its own label along with a label that either center token is an end of sentence or not. The best results were obtained on WSJ with an F score of 97.8%.

[Kiss and Strunk, 2006] worked on SBD by using different heuristics (ratios, length, etc.) and collocation information for finding abbreviations, initials, and ordinal numbers in an unsupervised manner. The system named 'Punkt' was evaluated on news data for eleven different languages and produced a high F score of 91.50% and 97.22% on Wall Street Journal (WSJ) and Lacio Web data sets respectively.

[Akita *et al.*, 2006] experimented with SBD on Japanese spoken transcripts extracted from Corpus of Spontaneous Japanese (CSJ). The total corpus consisted of 200 selected conversations out of which 30 conversations were used as test set. The corpus was annotated both manually and automatically. Two different approaches were applied, a statistical language model (SLM) and Support Vector Machine

(SVM) model. For SLM, they extracted three different feature categories: i) *Linguistic*, which consisted of word surface, reading, POS tags, conjugation based features, and so on, for each context based window, ii) *Pause*, which was calculated on normalized duration (if available), and iii) *Dynamic*, which was extracted by estimating the results of preceding words. For SVM, [Akita *et al.*, 2006] considered the task as text chunking and adopted three categories: i) I (inside), ii) O (outside), and iii) E (end) of chunk, respectively. They applied the YamCha text chunker which is based on SVM with polynomial kernel functions, and achieved an F score for manual annotations of 0.854 and 0.818 for SLM and SVM respectively.

[Gillick, 2009] combined rule-based features and a machine learning approach to SBD. They extracted the rule-based features from training data and fed them into the Support Vector Machine (SVM). For training, WSJ data was used and then the trained model was applied to New York Times data from the AQUAINT corpus to automatically annotate 100 million words. [Gillick, 2009] reported the error score (lowest 0.25%) instead of F score for presenting the results of their system.

[Orosz *et al.*, 2013] presented a hybrid system based on a rule-based approach and unsupervised machine learning for clinical data in Hungarian language. The data set consisted of $1,320$ lines in Dev set and $1,310$ lines in Test set respectively, without having a train set. They achieved an F score of 91.89% on the test set.

[Rudrapal *et al.*, 2015] collected social media text from Facebook, Twitter and manually annotated the data for sentence breaking utterances. The final corpus was composed of $6,444$ sentences in total. [Rudrapal *et al.*, 2015] applied two different approaches, rule-based and machine learning based, to solve the SBD task. Before applying any of approaches, the sentences were tokenized using CMU tokenizer to remove the ambiguities of end of sentence markers. [Rudrapal *et al.*, 2015] obtained an F score of 78.72% and 87.0% with the rule-based approach and SMO classifier respectively.

[Kreuzthaler and Schulz, 2015] worked on abbreviation and SBD with a supervised approach on medical narratives in German language. The authors collected patient discharge summaries from the Graz University Hospital for a period of more than 6 years. The final data set consisted of $1,696$ documents which were split into half for generating training and testing set. Different rule-based features were extracted from the data based on length, word information, and contextual and language information. These features were passed to an SVM classifier in the training phase for abbreviations and SBD individually, and achieved the F score of 0.95 and 0.94, respectively.

More recently, [Wiechetek *et al.*, 2019] conducted research on a North Sámi data set consisting of the Uralic language, which has a complex morphological structure and is spoken in Norway, Sweden and Finland. [Wiechetek *et al.*, 2019] applied a context based approach for feature extraction by using constraint grammar and some other structural based information. They report promising results for detecting sentence boundaries with 97% accuracy and 99.99% recall.

	English		**French**	
Data set	**Tokens**	**Sentences**	**Tokens**	**Sentences**
Train	904,057	22,342	827,825	22,636
Dev	49,859	1384	119,008	3141
Test	56,952	1265	106,577	2981

Table 1: Statistics of the FinSBD-2019 Data Set

3 Experimental Setup

This section covers the data set and configuration used for our experiments.

3.1 Data set

The Fin-SBD data set [Ait Azzi *et al.*, 2019] was provided in JSON format along with original PDF files. The data set was comprised of three different parts: i) Train set, ii) Dev set, and iii) Test set, for English and French. Table 1 shows the basic data set statistics. Each token was annotated as either *ES* (End of Sentence), *BS* (Begin of Sentence), or *O* (Ordinary token).

3.2 Evaluation Measures

The evaluation metrics include Precision, Recall and F score, which can be automatically computed by the supplied evaluation script for all three labels. Due to the overwhelming number of tokens that are labeled as *O*, the official system performance was only computed as the average F score for *BS* and *ES* label prediction. Also, the standard result format has only two decimal places, i.e. F score ranges from 0.00 to 1.00. While this might be sufficient for overall ranking of shared task participants, we found it too coarse-grained, especially in the detailed analysis of the rule-based system (cf. Section 5), for which we changed the result precision to five decimal places.

4 ML based SBD with Contextual Windows

This section describes our machine learning based SBD approach. It exploits the advantages of supervised machine learning along with context based information for a token passed as a window.

4.1 Generation of Contextual Windows

We utilize the surrounding information of a token as a context window to determine whether this token is a boundary token (either begin or end) or not. A contextual window (CW) for a center token at position i consists of some preceding tokens ($i-1$, $i-2$, ..., $i-n$), the token itself (i), and some succeeding tokens ($i+1$, $i+2$, ..., $i+n$) depending on the size of the window (n). A context window (CW) of size n can be generalized for a center token T_i as follows:

$$CW_i = T_{i-n} + ... + T_{i-1} + T_i + T_{i+1} + ... + T_{i+n}$$

The size of CW depends on how much context we want to take into account. For example, a context of size 1 will result in a CW size of 3, consisting of one preceding token, the center token, and one succeeding token. We calculated different sizes of CW, i.e. 3, 5, 7 and 9. Each CW is labeled for center token only, therefore, depending on that token, a CW can have one of the three labels *BS*, *ES*, or*O*. A CW is generated for each token that is present in the data set, with majority of the CWs falling into the *O* category.

4.2 Reduction of Contextual Windows

Generating a CW for each token would result in a huge list of windows that might include duplicates as well. As mentioned earlier, the majority of CWs are labeled *O*, and it is undesirable to have a highly imbalanced data set for the classification task as it can strongly impact the performance of classifier. Therefore, we opted for two strategies to remove the duplicates and to mitigate the effect of the inherently imbalanced nature of the data set.

Handling Majority Class

To optimize the approach by reducing the majority class *O*, we applied a selection criterion for each center token to decide whether to add the corresponding CW into the list or not. For application of this criterion, we generated two dictionaries based on the Train set consisting of all unique tokens labeled as *BS* and *ES*. We utilized the given *BS* and *ES* indices to get all *BS* and *ES* tokens from the Train set and then finding unique lists for both. The total number of unique *BS* and *ES* tokens are 921 and 216 for English, 790 and 575 for French Train data, respectively. These dictionaries served as selection criteria that if a center token is present in the dictionary, then the corresponding CW will be added in the final CW list, hence resulting in a smaller number of *O* labeled CWs than before.

Removing Duplicates

To remove the duplicates from the final CW list, at the time of inserting a new CW, a search was made on entire list to check if current CW is already present in the list or not. If current CW is already present, then it will not be added again to handle the redundancy. Finally, each input document was converted into a list of context based windows (CWs) which can be passed to a machine learning classifier for predicting labels for each center token of CW. To preserve the mapping of the original tokens with their indices, we stored the index of each center token for each CW.

4.3 Feature Representation

For applying a machine learning algorithm on textual data, we have to transform the textual data into some numeric or vector representation. There are different methods available for this, such as occurrence based, term-frequency based, TF-IDF (Term Frequency-Inverse Document Frequency), word co-occurrence matrix, and so on. TF-IDF is an occurrence based vector representation of text where TF (Term Frequency) represents the normalized score of word occurrence by the size of the document [Joachims, 1997]. TF of a word w can be expressed as below where D denotes to Document [Joachims, 1997].

$$TF(w) = \frac{Count(w)\ in\ D}{Total\ w\ in\ D}$$

The result of TF is the assignment of the same weights for each word in the vector representation, which is undesirable. This is because discriminating information of text is mainly contained in words *other than* stop words, articles, prepositions, etc which occured a lot in a document. Therefore, such unwanted words should have reduced weights, and the process of suppressing TF scores is done with IDF (Inverse Document Frequency). IDF of a word w can be calculated as follows where D denotes to Document.

$$IDF(w) = Log(\frac{Total\ \#\ of\ D}{\#\ of\ D\ having\ w})$$

TF-IDF is the product of both scores which converts the textual data into a vector representation where discriminative words have a higher score than others. We applied TF-IDF vector representation to CWs for generating character N-grams. This resulted in variable length of character N-grams where the minimum length of N is 3 and the maximum length of N is 10. The TF-IDF vector representation usually generates a huge vector space which is computationally expensive. Therefore, we selected the top $5,000$ features only.

4.4 Classification

We performed supervised machine learning to exploit the benefit of given annotated data set. To simplify the task, we split the task into two binary classifications: *BS* vs. *O* and *ES* vs. *O*. For classification, we opted for Random Forest ensemble classifier (decision trees = 100) due to its unbiased and stable nature and Naive Bayes classifier as a baseline. We provided TF-IDF based features on different CW sizes as an input to the classifiers. The Train and Dev data sets were used for training and evaluation to find the optimized configuration. We found the best results from both classifiers with CW size = 5 among all sizes. After that, Train and Test data sets were used for getting final predictions for Test set. Each predicted label was stored against the original index number (cf. Section 4.2), so it could be mapped back into the Test data. The given data set is the full list of tokens and their indices. The indices for which no prediction is made are marked as *O* by the given evaluation script. The final step is to merge the predictions of both classification tasks to maintain the structure and format of the given data set. As we treated *BS* and *ES* detection as separate tasks, so there are two predictions for each token. For each token, we require only one prediction, so two cases arise here for selecting a single prediction for each token, *BS/ES* vs. *O*, and *BS* vs. *ES*. We resolved this according to the evaluation script where all tokens are initialized with *O*, then *BS* markers are placed, and finally *ES* markers are written.

4.5 Results

Table 2 presents the results of the machine learning based approach with contextual windows CWs of size 5. The first column presents the data set for which the results are reported in the corresponding row. The second column denotes to the classifier used for the classification. The next columns are grouped together to present the results of *BS* and *ES* detection in terms of Precision (P), Recall (R) and F score (F). For the English data set, the highest results are obtained on the Dev set for *ES* (F score = 0.88) and *BS* (F score = 0.70) detection respectively with the Random Forest classifier, while

		BS			ES		
		P	R	F	P	R	F
Set	Clf	EN					
Dev	RF	0.65	0.77	**0.70**	0.83	0.93	**0.88**
Test	RF	0.60	0.72	0.65	0.74	0.94	0.83
Dev	NB	0.29	0.61	0.40	0.62	0.77	0.69
Test	NB	0.34	0.60	0.44	0.56	0.76	0.65
		FR					
Dev	RF	0.73	0.75	0.74	0.77	0.91	0.84
Test	RF	0.76	0.79	**<u>0.77</u>**	0.81	0.91	**<u>0.85</u>**
Dev	NB	0.38	0.57	0.46	0.52	0.78	0.63
Test	NB	0.38	0.56	0.45	0.54	0.77	0.63

Table 2: Results of ML based Approach with CW Size = 5

performance of the classifier dropped on the Test set for *ES* (F score = 0.83) and *BS* (F score = 0.65) detection respectively. The same behavior can be observed with the Naive Bayes classifie, i.e. better results are produced on Dev set for *ES* detection (F score = 0.69), however, for *BS* detection (F score = 0.44), the results of Test set are more satisfactory.

Overall, Recall is greater than Precision for Random Forest, which shows that the classifier learned very well despite the imbalanced nature of the data set. For Naive Bayes, Recall is also higher than Precision, however, low Precision results in low F score. For the French data set, the best results are obtained on Test set for *ES* (F score = 0.85) and *BS* (F score = 0.77) detection respectively with Random Forest classifier, and the performance of the classifier was a bit less on Dev set for *ES* (F score = 0.84) and *BS* (F score = 0.74) detection respectively. Interestingly, the performance of the Naive Bayes classifier on Dev and Test sets is very similar for *ES* (F score = 0.63) and *BS* detection (F score = 0.46 for Dev and F score = 0.45 for Test set). A similar pattern is identified for Recall and Precision where Recall is better than Precision for Random Forest, which also confirms the robust nature of the classifier. For Naive Bayes, Recall is also higher than Precision, however, low Precision resulted in low F score.
Comparing the results of the *BS* and *ES* detection task, overall the results of *ES* detection are higher than *BS* detection which depicts that *BS* marker are harder to find than the *ES* markers. The other reason for better *ES* detection result can be correlated with the total number of unique marks. As mentioned in Section 4.2, the total number of unique *ES* markers is much smaller than that of *BS* markers. Probably, the smaller number of unique markers resulted in better learning of classifier. Regarding the languages, the classifiers showed good performance for French on Test set considering both *ES* and *BS* detection.

5 Rule-based SBD

The general strategy underlying our alternative, rule-based approach can be characterized as follows: In an initial step, for easier access at the following stages, the input document is represented as a sequence of numbered tokens which are read from the original JSON data set. We do this by extracting the content of the 'text' element from the JSON file and splitting it on the basis of white space, thus maintaining the original tokenization and the gold data token indices, which are stored as the first item in a 4-tuple (cf. Figure 1). Then, we strip any trailing newlines from each token, but store in the tuple for each token whether it originally ended in a newline. Note that all period (.) tokens in the data have a trailing newline, apparently added at tokenization time, while other newlines are merely for layout purposes (e.g. headlines, cf. token 11778 in Figure 1). Finally, each tuple contains an (initially empty) list to which annotation labels are added. Figure 1 contains a short excerpt from the Dev data set.

```
...

(11765, 'calculated', False, []), (11766, 'using'       False, []),
(11767, 'the',        False, []), (11768, 'Relative',   False, []),
(11769, 'Value,'      False, []), (11770, 'at',         False, []),
(11771, 'Risk',       False, []), (11772, 'Approach',   False, []),
(11773, '.',           True, []), (11774, 'Portfolio',  False, []),
(11775, 'management', False, []), (11776, 'and',        False, []),
(11777, 'investment', False, []), (11778, 'advising',    True, []),
(11779, 'UBS',        False, []), (11780, 'Third',      False, []),
(11781, 'Party',      False, []), (11782, 'Management', False, []),
(11783, 'Company',    False, []), (11784, 'S',          False, []),
(11785, '.',           True, []), (11786, 'A',          False, []),
(11787, '.',           True, []), (11788, ',',          False, []),

...
```

Figure 1: Enriched Document Tokens before Automatic Annotation

Then, we perform an automatic, pattern-based annotation of the input document, which assigns descriptive labels (e.g. `URL`, `ABBREVIATION`, `ENUMERATION`, and `SECTION`) to sequences of tokens. Apart from providing shallow hints regarding the document structure (e.g. start / end of an `ENUMERATION`, which normally occur in groups), these labels also effectively disambiguate potential *ES* markers (mainly periods (.)) by binding them to the larger units. Since neither an annotation scheme nor an operationalizable definition of *BS* and *ES* for the domain of the Fin-SBD documents were available, rules were created inductively by analyzing the annotated documents (based on an HTML-based visualization). Finally, we perform the actual *ES* and *BS* detection, which employs hand-crafted, heuristic rules partly operating on the plain tokens, and partly on the token annotations. While the heuristic rule application happens *after* the pattern-based annotation, we describe it first (Section 5.1), and the pattern-based annotation afterwards (Section 5.2).

5.1 ES and BS Detection

ES and *BS* detection work by going through the list of 4-tuples and checking each tuple against a short list of rules. These rules take into account the string and pertaining annotations of both the current token T_i and its immediately preceding and following tokens T_{i-n} and T_{i+n}. *ES* detection is done first, because it provides some information that is used by *BS* detection later. The full list of heuristic rules for *ES* detection in their actual order is given in Figure 2. At most *one* rule is applied to every token T_i. We use the expression STRING(T_i) to represent the actual token, ANNO(T_i) to represent the annotations assigned to T_i, and NEWLINE(T_i) to represent whether the token originally ended in a newline.
Many of the rules in Figure 2 are more or less self-explanatory. Rules 1 and 2 directly handle potential *ES*-signalling tokens, with the additional condition in rule 1 ensuring that a previously disambiguated period character (.) is prevented from causing a sentence break. Rule 3 exploits an observed structural property of the annotation, i.e. that items in enumerations (like '(a) ...', '(b) ...', '(c) ...') are treated as sentences. Similarly, rule 4 exploits the fact that token se-

Given a token T_i, label it as *ES* if

1. STRING(T_i) = '.' **and**
   ANNO(T_i) = [ ];
2. STRING(T_i) = '?' **or** '!';
3. ANNO(T_{i+1}) = 'B-ENUM';
4. STRING(T_i) = ';' **or** 'and' **or** 'or' **and**
   STRING(T_{i+1}) = '-';
5. STRING(T_i) = ':' **or** ',' **or** ';' **or** 'and' **or** 'or' **and**
   ANNO(T_{i+1}) = 'B-ENUM' **and**
   ANNO(T_{i-1}) != 'E-ENUM';

Figure 2: Heuristic ES Detection Rules

quences starting with a dash (-) are also treated as sentences (cf. also rule 4 in Figure 3), and that the tokens ';', 'and', and 'or' also mark sentence breaks if they immediately precede such a dash. Rule 5, finally, handles sequences of enumerations. For *BS* detection, the rules are given in Figure 3. Note that tokens T_i for which ANNO(T_i) = 'B-HEADLINE' or 'B-BULLET' are regarded as non-*BS*, and are not submitted to the rules. The same is true for tokens T_i for which STRING(T_{i-1}) = '-' and NEWLINE(T_{i-2}) = True.

Given a token T_i, label it as BS if

1. ANNO(T_i) = 'B-ENUM';
2. ANNO(T_{i-1}) = 'E-HEADLINE';
3. ANNO(T_{i-1}) = 'E-BULLET' **and**
   ANNO(T_i) = [ ];
4. STRING(T_i) = '-' **and**
   ANNO(T_{i-1}) = '*ES*';
5. STRING(T_i) starts_with_uppercase **and**
   ANNO(T_{i-1}) = '*ES*';

Figure 3: Heuristic BS Detection Rules

The rules in Figure 3 are less complex than those in Figure 2. At least in part, this is due to the fact that they make use of the results of previous rules (cf. below). The first two rules in Figure 3 are very straightforward. One striking difference between the *BS* and the *ES* detection rules is that the former makes use of the output of the latter: Rule 4 is the complement to rule 4 in Figure 2, which assigns the *BS* label to dash characters (-) which immediately follow a previously assigned *ES* label. Similarly, rule 5 is a kind of default which assigns the *BS* label to all tokens with an uppercase initial character directly following a previously assigned *ES* label.

5.2 Pattern-based Automatic Annotation

In this step, the document is matched against a group of simple to averagely complex patterns. As mentioned earlier, this step happens *before* rule application, because the patterns are used to enrich the input for the rules. Some patterns operate strictly locally, while others depend on earlier patterns. The patterns, their sequence of application, and their respective *cumulative* effects on rule application for Train, Dev, and Test can be found in Table 3.

Note that the result precision was set to five decimal places because, as will become apparent, the effects that some patterns have on *BS* and *ES* detection are rather subtle. Row 0 contains the results when the rules described in Section 5.1 are applied without any previous annotation. Performance differences between Train, Dev, and Test on this level are indicative of inherent differences between these data sets. Actually, we can observe the following: i) On Train, F score for the *BS* task is higher than for the *ES* task, while on Dev and Test, F score for the *ES* task is higher. ii) Looking only at the *BS* task, P and R are *roughly* on a par for all three data sets, while for the *ES* task, R is generally much higher than P, and F scores for Train and Test are very similar (.67758 and .66194), while F score for Dev is much higher (.74945). iii) Looking only at the *ES* task, F score for Dev and Test are reasonably similar (.82627 and .80643), while F score for Train is much lower (.62658).

The `URLS` and `DATES` patterns (rows 1 and 2) detect simple expressions like 'www . ubs . com' and '12 . 10 . 2011', respectively. These are mainly targeted at period (.) disambiguation, which should be visible in improvement of P for *ES* detection. And in fact, *BS* performance on neither Train, Dev, nor Test is affected by these patterns, and *ES* R is also constant. We can see the expected improvements in P for *ES* detection, however, these are extremely small only.

The `ABBREVS_LU` and `ABBREVS_PT` patterns (rows 3 and 4) detect the abbreviations including period (.) characters. The first pattern does a simple look-up in a predefined list of abbreviations, while the second detects the abbreviations by matching sequences of single upper case letters and period characters. These two patterns are also mainly targeted at period disambiguation, which is indeed visible in considerable jumps in *ES* detection P (.49848 to .56340 for Train, .75073 to .89815 for Dev, and .73839 to .77070 for Test). R is consistently drops a little, but overall F score for *ES* detection increases. However, we also see a positive effect of these two patterns on P of the *BS* detection task, because with more correct *ES* being detected, the coverage of the *BS* detection rules (e.g. rule 5 in Figure 3) also improves.

The `SECTION` pattern (row 5) is similar to the `ABBREVS_PT` pattern, but detects sequences of numbers and period characters. This is another pattern targeting period disambiguation which should have an impact on P for *ES* task. However, this is the case for Train only, where P and consequently F score is considerable improved from .67546 to .73884, while there is hardly any impact on Dev and Test.

The `AMOUNTS` pattern (row 6) is a variant of the `SECTION` pattern which also handles leading and trailing zeros, to detect expressions such as '0 . 025'. This pattern has the overall expected positive effect on P (and F score) for the *ES* detection task, but again, we observe huge differences in improvement between Train on the one hand and Dev and Test on the other. For Train, application of this pattern boosts P of *ES* from .66324 to .83619 with only a marginal drop in R, resulting in an improvement in F score from .73884 to .83473. For Dev and Test, the final improvement in F score is only about .02.

`ENUMS` and `ENUMS-MERGE` (rows $7a$ and $7b$) is a two-step pattern which does not address period disambiguation, but the detection of expressions like '(a', '(a)', 'I b)' etc., which the Fin-SBD annotation treats as sentences. Accordingly, this pattern has a more complex effect, as it mainly addresses R for both *BS* and *ES*, while accepting some drops in P. In general, however, application of this pattern(s) results

	Train						Dev						Test					
	BS			ES			BS			ES			BS			ES		
Pattern	P	R	F	P	R	F	P	R	F	P	R	F	P	R	F	P	R	F
0 -	.68976	.66583	.67758	.49488	.85382	.62658	.76330	.73627	.74954	.74810	.92269	.82627	.65909	.66482	.66194	.73550	.89249	.80643
1 URLS	.68976	.66583	.67758	.49843	.85382	.62943	.76330	.73627	.74954	.74897	.92269	.82680	.65909	.66482	.66194	.73839	.89249	.80816
2 DATES	.68976	.66583	.67758	.49848	.85382	.62947	.76330	.73627	.74954	.75073	.92269	.82788	.65909	.66482	.66194	.73839	.89249	.80816
3 ABBREVS_LU	.72387	.66059	.69079	.54893	.84585	.66579	.76646	.72327	.74424	.80608	.91908	.85888	.65986	.66403	.66194	.75300	.89170	.81650
4 ABBREVS_PT	.74823	.65912	.70085	.56340	.84317	.67546	.83207	.71604	.76971	.89815	.91113	.90459	.66933	.66245	.66587	.77070	.89012	.82612
5 SECTIONS	.76025	.65075	.70125	.66324	.83390	.73884	.83207	.71604	.76971	.90394	.91113	.90752	.66933	.66245	.66587	.77495	.89012	.82855
6 AMOUNTS	.76025	.65075	.70125	.83619	.83327	.83473	.83404	.71532	.77013	.93824	.91113	.92449	.66933	.66245	.66587	.80300	.88933	.84396
7a ENUMS	.71812	.73673	.72730	.79389	.89925	.84329	.78777	.79118	.78947	.89708	.97616	.93495	.62608	.74387	.67991	.76662	.94783	.84765
7b ENUMS-MERGE	.73191	.72961	.73076	.81434	.89893	.85455	.81111	.79118	.80102	.92156	.97616	.94807	.64408	.74387	.69039	.78830	.94783	.86073
8 HEADLINES	.80346	.85413	.82802	.81327	.88443	.84736	.89393	.88295	.88840	.92156	.97616	.94807	.74251	.86166	.79766	.79024	.94704	.86156
9 BULLETS	.80199	.84245	.82172	.81126	.86939	.83932	.90973	.93208	.92077	.92324	.97327	.94759	.74251	.86166	.79766	.79024	.94704	.86156

Table 3: Results of BS and ES Detection Rules with Cumulative Effect of Different Annotation Patterns

in an increase in F for both *BS* and *ES* for all data sets.

The HEADLINES pattern (row 8) uses, among other things, the NEWLINE feature extracted from the raw data (cf. Figure 1 above) to distinguish actual sentences from sentence-like headlines, which are not annotated as sentences in the Fin-SBD data. Headline information is explicitly used for *BS* detection (cf. rule 2 in Figure 3). Accordingly, this pattern mainly affects the *BS* results, while *ES* results are mostly unaffected. It greatly improves both P and R for *BS* on all three data sets, with increases of up to .1 in F: F for Train increases from .73076 to .82802, for Dev from .80102 to .88840, and for Test from .69039 to .79776.

The final pattern is `BULLETS` (row 9), which looks for itemized text. The effect of this pattern, however, is mixed, with some small increases in R for some data sets, and some decreases in others, but no effect at all on the Test set.

Of the two final results that we submitted, the first one (HITS-SBD1) is the final result for Test in Table 3, which was rounded and averaged by the organizers to an F of **.83**.

5.3 Optional PDF Re-Processing

The patterns and rules described and analyzed above represent the core of our rule-based system. We created one alternative result (submitted as HITS-SBD2) in which we addressed an apparent problem with the original tokenization, which we expected to improve the results considerably. During the data set inspection, we often observed cases where tokens in the original data set were incorrectly merged, like in the following examples:

```
(31288, 'the', [], False),
(31289, 'Prospectus', [], False),
(31290, '.', [], True),
(31291, '_______________________Investor', [], False),
(31292, 'profile', [], True),
(31293, 'The', [], False)
...
(38074, 'duties', [], False),
(38075, 'or', [], False),
(38076, 'other', [], False),
(38077, 'chargesD', [], False),
(38078, '=', [], True),
(38079, 'net', [], False)
```

We applied Apache tika[1] to the provided PDF files and created an improved tokenization. We automatically aligned it with the original tokenization and detected cases of incorrect token merges. These were then split, where care was taken to retain the original token indices. As a result, the improved tokenization looked like the following:

```
(31288, 'the', [], False),
(31289, 'Prospectus', [], False),
(31290, '.', [], True),
(31291, '_______________________', [], False),
(31291, 'Investor', [], False),
(31292, 'profile', [], True),
(31293, 'The', [], False)
...
(38074, 'duties', [], False),
(38075, 'or', [], False),
(38076, 'other', [], False),
(38077, 'charges', [], False),
(38077, 'D', [], False),
(38078, '=', [], True),
(38079, 'net', [], False)
```

While we were expecting this improved pre-processing to have a huge effect on our results, the actual improvements were minimal: On Test, we obtained an F of .80162 for *BS* and .86280 for *ES*, respectively.

6 Summary and Conclusion

We presented two competing systems for SBD that were developed for the Fin-SBD shared task 2019. The ML-based system was based on a simple and elegant approach which was inspired by the prior work on SBD in more classical, less noisy genres, where more locally oriented features (like the context windows applied in this work) are known to work better. However, in the domain of the Fin-SBD shared task, where the notion of *sentence* is less well-defined, this approach failed to reach an acceptable result. The other approach was based on a combination of simple, surface-based patterns, and heuristic rules for solving the task in an unsupervised manner. Both patterns and rules were created manually on the basis of the introspection of the Train and Dev data sets. While this approach allowed for the creation of some high-precision rules, including ones that are strongly tailored towards the sometimes idiosyncratic annotations in the original data set, it failed to produce a complete and sufficiently robust solution. However, the results, while more in the bottom range, are still acceptable, especially given the fact that the results from all competing parties are rather close together.

Acknowledgements

This work has been supported by the Klaus Tschira Foundation, Heidelberg, Germany. In addition, the first author has been supported by the German Academic Exchange Service (DAAD), Germany, and Higher Education Commission (HEC), Pakistan.

[1]https://tika.apache.org/

References

[Agarwal *et al.*, 2005] Neha Agarwal, Kelley Herndon Ford, and Max Shneider. Sentence boundary detection using a maxEnt classifier, 2005.

[Ait Azzi *et al.*, 2019] Abderrahim Ait Azzi, Houda Bouamor, and Sira Ferradans. The FinSBD-2019 Shared Task: Sentence boundary detection in PDF Noisy text in the Financial Domain. In *The First Workshop on Financial Technology and Natural Language Processing (FinNLP 2019)*, Macao, China, 2019.

[Akita *et al.*, 2006] Yuya Akita, Masahiro Saikou, Hiroaki Nanjo, and Tatsuya Kawahara. Sentence boundary detection of spontaneous Japanese using statistical language model and support vector machines. In *Ninth International Conference on Spoken Language Processing (ICSLP)*, pages 1033–1036, Pittsburgh, Pennsylvania, 2006. International Speech Communication Association.

[Gillick, 2009] Dan Gillick. Sentence Boundary Detection and the Problem with the U.S. In *Proceedings of Human Language Technologies: The 2009 Annual Conference of the North American Chapter of the Association for Computational Linguistics, Companion Volume: Short Papers*, pages 241–244, Boulder, Colorado, 2009. Association for Computational Linguistics.

[Joachims, 1997] Thorsten Joachims. A Probabilistic Analysis of the Rocchio Algorithm with TFIDF for Text Categorization. In *Proceedings of the Fourteenth International Conference on Machine Learning (ICML '97)*, pages 143–151, Nashville, TN, USA, 1997. Morgan Kaufmann Publishers Inc.

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

[Kiss and Strunk, 2006] Tibor Kiss and Jan Strunk. Unsupervised multilingual sentence boundary detection. *Computational Linguistics*, 32(4):485–525, 2006.

[Kreuzthaler and Schulz, 2015] Markus Kreuzthaler and Stefan Schulz. Detection of sentence boundaries and abbreviations in clinical narratives. *BMC Medical Informatics and Decision Making*, 15(2):S2–S4, 2015.

[Orosz *et al.*, 2013] György Orosz, Attila Novák, and Gábor Prószéky. Hybrid text segmentation for Hungarian clinical records. In *12th Mexican International Conference on Artificial Intelligence*, pages 306–317, Mexico City, Mexico, 2013. Springer.

[Polanyi *et al.*, 2004] Livia Polanyi, Chris Culy, Martin Van Den Berg, Gian Lorenzo Thione, and David Ahn. A rule based approach to discourse parsing. In *Proceedings of the 5th SIGdial Workshop on Discourse and Dialogue at HLT-NAACL 2004*, pages 108–117, Cambridge, Massachusetts, USA, 2004. Association for Computational Linguistics.

[Reynar and Ratnaparkhi, 1997] Jeffrey C. Reynar and Adwait Ratnaparkhi. A maximum entropy approach to identifying sentence boundaries. In *ANLC '97: Proceedings of the Fifth Conference on Applied Natural Language Processing*, pages 16–19, Washington, DC, 1997. Association for Computational Linguistics.

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

[Wiechetek *et al.*, 2019] Linda Wiechetek, Sjur Nørstebø Moshagen, and Thomas Omma. Is this the end? Two-step tokenization of sentence boundaries. In *Proceedings of the Fifth International Workshop on Computational Linguistics for Uralic Languages*, pages 141–153, Tartu, Estonia, 2019. Association for Computational Linguistics.