Combining Off-the-shelf Grammar and Spelling Tools for the Automatic Evaluation of Scientific Writing (AESW) Shared Task 2016

René Witte and Bahar Sateli Knowlet Networks Inc. Montréal, QC, Canada http://www.knowlet.net

Abstract

We applied two standard, open source tools for detecting spelling and grammar errors to the AESW 2016 shared task: *After the Deadline* and *LanguageTool*. The tools' output was combined with a Maximum Entropy machine learning model to classify each input sentence as requiring or not requiring any edits. This approach yielded the second-highest precision of 64.41% in the binary estimation task at AESW 2016, but also the lowest recall of 36.85%, resulting in an F-Measure of 46.34%.

1 Introduction

The Automated Evaluation of Scientific Writing Shared Task (AESW 2016) targeted the analysis of individual sentences, in order to assess whether or not a sentence as a whole requires editing or not (Daudaravicius et al., 2016). The long-term vision behind this task is to "promote the development of automated writing evaluation tools that can assist authors in writing scientific papers".¹

Our work in this area is similarly motivated by the idea of providing an interactive "virtual research assistant" that supports researchers in their daily tasks. Writing support includes providing interactive feedback on the quality of textual artifacts to academic authors. Such a support generally requires achieving high precision over recall, as tools with too many false positives tend to be ignored by authors. Additionally, providing salient feedback on detected mistakes – ideally together with suggestions for improvements – to the authors is an important feature. Standard, open source grammar and spelling tools have addressed these questions for quite some time, but are generally not focused on academic texts. Hence, we were interested in how well existing, general-purpose tools perform when applied to academic writing. In our experiments, we applied two well-known tools, *After the Deadline* and *LanguageTool* (described in Section 2.3), to the AESW data sets. Our hypothesis was that these two tools are (*i*) complementary to some degree, so that their combination can increase precision and/or recall; and (*ii*) a machine learning approach can combine the tools' output with additional syntactical context information, thereby attuning them to academic writing.

2 Methods

In this section, we discuss the setup of our AESW 2016 experiments.

2.1 Data and Task Description

Here, we briefly describe the datasets and tasks; for the full details, please refer to (Daudaravicius et al., 2016).

The input sentences were randomly selected from more than 9,000 journal articles across different domains (Computer Science, Physics, Human Sciences, etc.). The *training* and *development* data sets contain 256,389 and 31,732 sentences, respectively. Some sentences contain changes that were applied by professional editors, who are native English speakers. These sentences have inline <ins> and tags that mark inserted and removed content, respectively. An example sentence is shown in Fig. 1.

The test data set (31,085 sentences) "retain texts

¹AESW 2016, http://textmining.lt/aesw/index.html

```
<sentence sid="3570.3">
The enterprise aims to increase <del>in</del>
<ins>the</ins> output (at the same time to
reduce expenses) _MATH_ and to decrease
<del>in</del><ins>the</ins> consumed
<del>efforts</del><ins>effort</ins> _MATH_.
```

Figure 1: Example sentence in XML format from the AESW 2016 development set

tagged with tags and the tags are dropped. Texts between <ins> tags are removed."²

The goal of the AESW tasks was to predict whether a given sentence as a whole requires editing – that is, individual insertions or deletions did not have to be annotated. Thus, the output for the two tasks was either a boolean feature (True meaning a sentence requires editing) or a probabilistic feature with a value in [0,1] (where "1" indicates that an edit is required).

2.2 Preprocessing

To facilitate cross-fold evaluations, we split all AESW data sets (development, training, and testing) into individual XML files containing 1000 sentences each.

For machine learning, each sentence from the development and training sets received an *edit* feature of True if it contained at least one <ins> or tag, otherwise the *edit* feature was set to False. For training, content marked as 'inserted' (between <ins> tags as shown in Fig. 1) was removed from the texts. The tags were likewise removed, but the content was retained, thereby showing a sentence's content before any changes performed by an editor. Note that this conforms to the format of the test set, as described above.

2.3 Writing Error Detection Tools

We experimented with two open source tools for writing error detection:

After the Deadline (AtD) detects spelling, grammar, and style errors (Mudge, 2010).³ We used version atd-081310 in its default configuration for our experiments.

LanguageTool (**LT**) is another popular open source spelling and grammar tool,⁴ which also supports multiple languages. We used version 3.2-SNAPSHOT from the LT GitHub repository⁵ for our experiments.

2.4 Experimental Setup

To facilitate the combination of the individual results, we integrated both tools into a pipeline through the *General Architecture for Text Engineering* (GATE) (Cunningham et al., 2011). Each error reported by one of the tools is added to the input text in the form of an *annotation*, which holds a start- and end-offset, as well as a number of *features*, such as the type of the error, the internal rule that generated the error, and possibly suggestions for improvements, as shown in Figure 2.

Additionally, we added a number of standard GATE plugins from the ANNIE pipeline (Cunning-ham et al., 2002) to perform tokenization, part-of-speech tagging, and lemmatization on the input texts. Finally, annotations spanning placeholder texts in the sentences, such as _MATH_, were filtered out, as these were particular to the AESW data.

2.5 Machine Learning

In addition to applying the AtD and LT tools individually, we experimented with their combination through machine learning. Essentially, we follow a stacking approach (Witten and Frank, 2011) by treating the AtD and LT tools as individual classifiers and use them to train a model for assigning the output 'edit' feature to a sentence.

ML Features. Table 1 lists all features we derive from the input sentences. We experimented with different token root and category n-grams, including unigrams, bigrams and trigrams.

ML Algorithms. Training and evaluation were performed using the Weka⁶ (Witten and Frank, 2011) and Mallet⁷ (McCallum, 2002) toolkits. These were executed from within GATE using the *Learning*

²AESW 2016 Data Set, see http://textmining.lt/aesw/index. html#data

³After the Deadline, http://www.afterthedeadline.com/

⁴LanguageTool, https://languagetool.org/

⁵LanguageTool GitHub repository, https://github.com/ languagetool-org/languagetool

⁶Weka, https://sourceforge.net/projects/weka/

⁷Mallet, http://mallet.cs.umass.edu/

* 🚓 🗶 ***							
🎽 🏘 🖗 🔮 🌘							
GATE	🕴 Messages 🕻 🎆 Knowlet AEWS201 💰 AESW Developmen 💰 xad_aesw2016(v1						
* Applications	Annotation Sets Annotations List Annotations Stack Co-reference Editor Text	5					
— 🎆 Knowlet AEWS2016-Processing	This paper is an exemption of the sugged in on the sugginized as follows.						
- 🎆 Knowlet AESW2016-Learning	This article deals with _MATH_ equipped with its standard cone _MATH_, and all the matrices are real square matrices of order _MATH_ unless + 🖌 🔽 AND						
🙀 Language Resources	The set of all _MATH_ matrices over the real numbers _MATH_ is denoted by _MATH						
- 📀 xad_aesw2016(v1.2)_dev.finf_0	We denote the transpose, the null space and the range space of _MATH_by_MATH_ IMATH_ and _MATH_ respectively. _MATH_ is said to be nonnegative (i.e., _MATH_) if all the entries of _MATH_are nonnegative, and _MATH_for matrices _MATH_and _MATH_ if						
S AESW Development	MATH Similarly, a vector MATH means that MATH for all MATH , and A Y A P						
2 Processing Resources	Let_MATH_ be complementary subspaces of _MATH_						
Datastores	Then _MATH_ denotes the projection of _MATH_ onto _MATH_ ald LanguageTool del del						
Dutastores	The spectral radius of a matrix _MATH_ is denoted by _MATH_, ar C ITSIssueType v misspelling v X						
	eventually occur. Message v Possible spelling mistake found v X						
	In these situations, the operator may want to just try again the faile Rule V MORFOLOGIK_RULE_EN_US V X						
	However, in the initial fine-grained parallel BC algorithm, there and Context and the state of t						
	focus position 5. C SuggestedReplacements 🗸 non negative 🗸 🗙						
	A single psd_MATH_was constructed for each time interval.						
	were averaged together to produce _MATH_ with lower noise.						
	The nearest 96-minute full disk low-resolution magnetogram y						
	Type Set Start End Id						
limeTvpe 💌 text/xml	LanguageTool 33181 33192 101987 (ITSIssueType=misspelling, Message=Possible spelling mistake found, Rule=MORFOLOGIK_RUL						
	AtD 33395 33404 103380 (ITSIssueType=misspelling, Rule=spelling, ShortMessage=Spelling, SuggestedReplacements=[s] LanguageTool 33395 33404 101988 (ITSIssueType=misspelling, Rule=spelling, ShortMessage=Spelling, SuggestedReplacements=[s]						
ate.SourceURL ▼ file:/home/rene/R∉	Language tool 3339 33404 TU1988 (IT Sissue Type=misspeiling, Message=rossine speiling misstake toulina, Rueim-Mork-OLOum-Kita-Ruo AtD 334347 33454 (1015) 13382 (TSIssue Type=grammar, Message=rolssible a strong verb for <mp>rojection</mp>						
~	AtD 33462 33466 103381 (TSIssueType=style, Message= <h3>Try: ***<h3>You should avoid redundant expressions w</h3></h3>						
	AtD 33597 33607 103384 (ITSIssueType=grammar, Message= <h>3>Revise is denoted/em> with active voice<h>3>is denoted/em> with active voice</h></h>						
	LanguageTool 33654 33660 101989 (ITSIssueType=misspelling, Message=Possible spelling mistake found, Rule=MORFOLOGIK_RUL						
		_					
	2140 Annotations (1 selected) Select:	Nev					

Figure 2: Combination of the writing analysis tools After the Deadline and LanguageTool in the GATE Developer GUI

Feature	Description	
Token.root	Morphological root of the token	
Token.category	Part-of-speech tag for the token	
LT.rule	Rule name as reported by LT	
LT.string	Reported text (surface form)	
AtD.rule	Rule name as reported by AtD	
AtD.string	Reported text (surface form)	

Table 1: Machine Learning Features

*Framework Plugin.*⁸ We experimented with a number of classification algorithms, including Decision Trees, Winnow, Naïve Bayes, KNN, PAUM, and Maximum Entropy. The latter generally performed best for the dataset and features, hence in this paper we only report the results from the MaxEnt model.

3 Results

In this report, we provide a summary of our system's results – for a complete description of all AESW 2016 results, please refer to (Daudaravicius et al., 2016).

Baseline experiments. To establish a baseline, we ran the AtD and LT tools on the development set. Here, every sentence that had at least one error annotation received an edit feature of True. Table 3 shows the results as reported by the *Codalab* site⁹ used in the competition.

Tool	Precision	Recall	F-Measure
AtD	0.4318	0.7448	0.5467
LT	0.4719	0.4739	0.4729

Table 3: Baseline experiments: Evaluation of the individual tools

 on the development set

Feature analysis. We measured the impact of the various features shown in Table 1 on the classification performance. A selected set of results is shown in Table 2. Accuracy was calculated using Mallet with a three-fold cross-validation on the training data set. Generally, adding more features increased precision, but did not improve recall.

Submitted run. For the submitted run, we retrained the MaxEnt classifier using the full fea-

⁸GATE Learning Framework Plugin, https://github.com/ GateNLP/gateplugin-LearningFramework

⁹Codalab, http://codalab.org/

Feature Set	Accuracy
AtD.rule, AtD.string, LT.rule, LT.string	0.6261
AtD.rule, AtD.string, LT.rule, LT.string, Token.root unigrams, Token.category unigrams	0.6584
AtD.rule, AtD.string, LT.rule, LT.string, Token.root bigrams, Token.category bigrams	0.7300
AtD.rule, AtD.string, LT.rule, LT.string, Token.root trigrams, Token.category trigrams	0.8525

Table 2: Three-fold cross-validation of the MaxEnt classifier on the training data with different feature sets

ture set (using trigrams for both Token.root and Token.category) on both development and training set. The exact same configuration was used for the probabilistic task submission, using the classifier's confidence as the prediction value (with 1-confidence for sentences classified as not requiring edits). The results are summarized in Table 4.

ΤοοΙ	Precision	Recall	F-Measure
binary	0.6241 (1)	0.3685 (8)	0.4634 (7)
probabilistic	0.7294 (2)	0.6591 (6)	0.6925 (5)

 Table 4: Submitted runs for the AESW 2016 task on the test set

 (as reported by Codalab)

4 Conclusions

Based on our experiments, standard spell and grammar checking tools can help in assessing academic writing, but do not cover all different types of edits observed in the training data. In future work, we plan to categorize the false negatives and develop additional features to capture specific writing errors.

As the AESW 2016 task was performed on individual sentences, the results do not accurately reflect the interactive use within a tool: False positive errors, such as spelling mistakes reported for an unknown acronym, are counted for every sentence, rather than once for the entire document, thereby decreasing precision significantly when an entity appears multiple times. Also, document-level writing errors, such as discourse-level mistakes, cannot be captured with this setup - for example, use of acronyms before they are defined or inconsistent use of American vs. English spelling. Finally, while the sentence-level decision can be helpful in directing the attention of an editor to a possibly problematic sentence, by itself it does not explain why a given sentence was flagged or how it could be improved, which are important information for academic writers.

References

- Hamish Cunningham, Diana Maynard, Kalina Bontcheva, and Valentin Tablan. 2002. GATE: A Framework and Graphical Development Environment for Robust NLP Tools and Applications. In *Proceedings of the 40th Anniversary Meeting of the Association for Computational Linguistics (ACL'02)*.
- Hamish Cunningham, Diana Maynard, Kalina Bontcheva, Valentin Tablan, Niraj Aswani, Ian Roberts, Genevieve Gorrell, Adam Funk, Angus Roberts, Danica Damljanovic, Thomas Heitz, Mark A. Greenwood, Horacio Saggion, Johann Petrak, Yaoyong Li, and Wim Peters. 2011. *Text Processing with GATE (Version 6)*. http://tinyurl.com/gatebook.
- Vidas Daudaravicius, Rafael E. Banchs, Elena Volodina, and Courtney Napoles. 2016. A report on the automatic evaluation of scientific writing shared task. In *Proceedings of the Eleventh Workshop on Innovative Use of NLP for Building Educational Applications*, San Diego, CA, USA, June. Association for Computational Linguistics.
- Andrew Kachites McCallum. 2002. MALLET: A Machine Learning for Language Toolkit. http://mallet.cs. umass.edu/.
- Raphael Mudge. 2010. The Design of a Proofreading Software Service. In Workshop on Computational Linguistics and Writing: Writing Processes and Authoring Aids (CL&W 2010).
- Ian H. Witten and Eibe Frank. 2011. *Data Mining: Practical Machine Learning Tools and Techniques*. Morgan Kaufmann, 2nd edition.