Feature-Rich Error Detection in Scientific Writing Using Logistic Regression

Madeline Remse, Mohsen Mesgar and **Michael Strube**
Heidelberg Institute for Theoretical Studies gGmbH
Schloß-Wolfsbrunnenweg 35
69118 Heidelberg, Germany
(madeline.remse|mohsen.mesgar|michael.strube)@h-its.org

Abstract

The goal of the Automatic Evaluation of Scientific Writing (AESW) Shared Task 2016 is to identify sentences in scientific articles which need editing to improve their correctness and readability or to make them better fit within the genre at hand. We encode many different types of errors occurring in the dataset by linguistic features. We use logistic regression to assign a probability indicating whether a sentence needs to be edited. We participate in both tracks at AESW 2016: *binary prediction* and *probabilistic estimation*. In the former track, our model (HITS) gets the fifth place and in the latter one, it ranks first according to the evaluation metric.

1 Introduction

The AESW 2016 Shared Task is about predicting if a given sentence in a scientific article needs language editing. It can therefore be pictured as a binary classification task. Two types of prediction are evaluated: binary prediction (*false* or *true*) and probabilistic estimation (between 0 and 1). These types of prediction form the two tracks of the shared task, both of which we participate in.

We solve both problems by applying a logistic regression model. We design a variety of features based on a thorough analysis of the training data. We choose the set of features that yields the highest performance on training and development sets.

Accounting for the imbalance of numbers of wrong and correct sentences in the training data during feature selection we obtain a model for the probabilistic task that outranks our competitors' systems. However, a detailed analysis of the results shows that the model takes advantage of the evaluation metric and that our less informed system produces results that are, although not yielding a top evaluation score, more meaningful.

In the course of a profound analysis of the training data we encounter both linguistic errors, which likely occur in diverse genres, and such errors that are intrinsic to scientific writing and thus rank among the major challenges of this task. As pointed out on the AESW 2016 webpage[1], correcting problems concerning diction and style is a matter of opinion. It depends on factors that are not necessarily deducible from linguistic properties. Common abbreviations are an example. There are cases where they are accepted by an editor, and there are cases where they are corrected. That is, sometimes *e.g.* is left as is and sometimes it is changed to *for instance* or *for example* without any obvious reason. There are even words that are corrected in opposite directions. For example, the first letter of the name prefix *van* has been corrected to be uppercase in some sentences and also has been corrected to be lowercase in other sentences. Especially abbreviations that are not common within one particular domain, but are used in isolated documents are problematic. This is due to limitations of the dataset, which provides only paragraphs, but not documents as contexts for sentences. For example, we may assume that *R-G* has been introduced as a technical term at some point in a document. But since we do not know which paragraphs belong to this document, we cannot be sure that this is the case.

[1] http://textmining.lt/aesw/index.html

Proceedings of the 11th Workshop on Innovative Use of NLP for Building Educational Applications, pages 162–171,
San Diego, California, June 16, 2016.

Section 2 gives an overview of the types of errors we encountered. In Section 3 we introduce our system design, detail on how we derive features from our data analysis, what kinds of language models we apply, give a short outline on logistic regression and describe the implementation of our system. In Section 4 we describe our training steps, followed by reporting results in Section 5, a discussion of lessons learned in Section 6 and related work in Section 7.

2 Data Analysis[2]

2.1 Simple Errors

SPELLING ERRORS are frequent and many concern using hyphens in compounds. Another common error is the wrong usage of ARTICLES. Definite articles are missing or unnecessarily inserted before generic nouns, (for instance *over the formula _REF_*). Indefinite articles are erroneous with respect to the subsequent phoneme, (e.g. *a open neighborhood*). Some errors concern descriptions of REFERENCES, which are usually capitalized (*table _REF_* or *figures _REF_ and _REF_*). NUMERALS are spelled out when they should not be, and vice-versa (*2* or *seventy-three*). It is correct to spell numerals out if they are smaller than 10, otherwise they are often spelled in digits. CONTRACTIONS, such as *doesn't* and *what's*, are considered too colloquial for scientific writing. Dots behind ABBREVIATIONS are omitted, and also common abbreviations such as *e.g.*, *i.e.* and *vs.* are written wrongly. Other errors include incorrect PLURALIZATION of decades (*1980's*), regular past tense generation of IRREGULAR VERBS (*lighted*) and the modification of words by the wrong PREPOSITION (*very different to the correction*). Words are unnecessarily REPEATED sometimes (*The the*).

2.2 Complex Errors

All errors described above can easily be categorized by means of simple patterns. Other errors are harder to capture, for example wrong word order or missing words. The most common errors that we come across are mistakes in the PUNCTUATION of a sentence, especially unnecessary or missing commas.

[2]All examples in this section have been drawn from the training data.

NUMBER DISAGREEMENT is a common grammatical error. It occurs in passive or active clauses (e.g. *the system are assumed to be the following form* and *the counter variables goes on changing*) and in nominal phrases (e.g. *Three class of boundary conditions* and *these new set of Lyapunov terms*).

WORK-SPECIFIC ABBREVIATIONS such as the insertion of *R-G* for the compound *recombination-generation* are errors that occur in individual situations. Detecting issues with DICTION AND STYLE is probably the most intricate problem in this task.

3 System Design

3.1 Formally Capturing Error Types

Simple errors can mostly be captured by binary features that formalize rules. For example, if a sentence contains an incorrect ABBREVIATION of *id est*, such as *ie.*, then it needs correction. Similar rules can be applied to the spelling mode of cardinal numbers and the CONTRACTION of auxiliary words, such as *'s*, *'ve*, etc. Also, when finding a four digit number starting with 1 and ending with 0, it is likely to denote a decade. If it is directly followed by *'s*, an incorrect PLURALIZATION is detected.

Some rules formulated that way need additional information. To assert that *seventeen* should not be spelled out the system must be aware that it denotes a NUMERAL greater than 10. This information can be made available through appropriate mappings. Lists of wrongly generated past tense forms of IRREGULAR VERBS can be created with managable effort, just like lists of common abbreviations.

SPELLING ERRORS can be detected by looking up words in a dictionary. Whether or not a compound requires being joined by a hyphen cannot be determined that way. Compounds can be created productively and are not necessarily in a dictionary.

NUMBER DISAGREEMENTS are easy to detect by means of dependencies between head and modifiers within phrases and part-of-speech tags, which often carry information about the number of words. However, that means that recognizing these errors heavily depends on the correctness of the dependency trees and the part-of-speech tags.

Other error types are ascertainable by language modeling. PREPOSITIONS often occur in

combination with the same words. Thus an appropriately trained language model learns that the word *different* occurs with *from* much more frequently than with *to*. Classic n-gram models account for unusual sequences of words and faulty word orderings. Language models based on co-occurrences of constituents in syntax trees can reveal grammatical errors and indicate positions where a comma or article is likely to be inserted.

3.2 Language Models

To capture more complex errors we use a variety of language models that we compute on correct sentences in the training data.

The n-gram probability of the i^{th} linguistic unit of a sentence l_i, being a token w or a part-of-speech tag t, given its $n-1$ predecessors is defined as

$$p(l_i \mid l_{i-n+1}^{i-1}) = \frac{c(l_{i-n+1}^{i})}{c(l_{i-n+1}^{i-1})},$$

where $c(x)$ is the number of occurrences of x throughout the dataset (Jurafsky and Martin, 2009, pp. 117–147).

Language modeling is not limited to a language unit and its direct predecessors. The probability of the occurrence of a word or part-of-speech tag can be computed depending on whatever might be appropriate to model a linguistic phenomenon. Therefore we compute the probability of a linguistic unit given the subsequent $n-1$ linguistic units:

$$p(l_i \mid l_{i+1}^{i+n-1}) = \frac{c(l_{i}^{i+n-1})}{c(l_{i+1}^{i+n-1})}.$$

The following formula for the probability of a word w accounts for the relation between part-of-speech tags and lexicals:

$$p(w_i \mid t_i) = \frac{c(w_i, t_i)}{c(t_i)}.$$

In order to identify words that are typically preceded by a particular part-of-speech, we compute

$$p(t_i \mid w_{i+1}) = \frac{c(t_i, w_{i+1})}{c(w_{i+1})}.$$

Given a syntax tree, let $succ(g)$ be the right sibling of a node g, let $pred(g)$ be the left sibling of a node g, and let $child(g)$ be the set of children of a node g. We define:

$$p(pred(h) = g \mid h) = \frac{c(pred(h) = g)}{\sum_{g' \in C} c(pred(h) = g')},$$

$$p(succ(h) = g \mid h) = \frac{c(succ(h) = g)}{\sum_{g' \in C} c(pred(h) = g')},$$

$$p(g \in child(h) \mid h) = \frac{c(g \in child(h))}{\sum_{g' \in C} c(g' \in child(h))},$$

where C is the set of constituents.

Other sets of features address the probability of prepositional phrases as modifiers of words. Let $nmod(v)$ be a preposition that modifies a word v:

$$p(nmod(v) = w) = \frac{c(nmod(v) = w)}{\sum_{w' \in V} c(nmod(v) = w')}.$$

Smoothing: Since the purpose of our language models is to identify unusual combinations and orderings of words, part-of-speech tags, and chunks, we go without strong smoothing measures and leave it to machine learning to reveal the point where a language construct qualifies as unacceptably improbable. Also, we do not prune the vocabulary, because technical terms which are limited to very specific scientific fields or even to only few documents are characteristic for scientific writing. For practical reasons we apply the very basic add-δ smoothing (Jurafsky and Martin, 2009, p. 134), choosing $\delta = 0.1$ in order to prevent zero-division.

3.3 Features

We implement a total of 82 features based on the data analysis described in Section 2. These features can be classified into three sets, depending on their range. Features 1–14 (see Table 1) are integer-valued, features 15–55 (see Table 2) are binary, and features 56–82 (see Table 3) are real-valued.

Most of the integer-valued features originate in readability research and address the coherence of documents, but they may also be helpful to assess sentence quality (Pitler and Nenkova, 2008). It is plausible that long sentences or sentences with a very high parse tree should be shortened or split into more sentences in order to simplify their syntax. Thus they account for those cases where phrases are deleted in favor of conciseness. Many occurrences of constituents such as *VP*, *SBAR* or *NP* are likely to

ID	Definition
1	number of definite articles (token *the* with POS-tag *DT*)
2	number of pronouns (tokens with POS-tag *PRP*)
3	number of SBAR (subtrees of syntax tree with root *SBAR*)
4	number of VP (subtrees of syntax tree with root *VP*)
5	number of NP (subtrees of syntax tree with root *NP*)
6	sentence length (number of tokens)
7	parse tree height (edges on the longest path between the root and a leaf of the syntax tree)
8	number of constituents (subtrees of the syntax tree)
9	number of words not in vocabulary (tokens never seen in training)
10	number of words unknown to WordNet (ignores stop words, compounds with hyphens, tokens with digits)
11	number of words unknown to `pyenchant`-package using `en_US`-dictionary (ignores stop words, compounds with hyphen, tokens with digits)
12	maximal number of verb forms in a row (longest row of POS-tags starting with 'VB')
13	number of dots (ignores period at the end of a sentence)
14	number of abbreviations in paragraph (feature 13, summed over all sentences of a paragraph)

Table 1: Integer-valued features.

occur in too complex sentences. Many pronouns are indicative for ambiguity, since it is more difficult to identify the corresponding antecedents.

The binary features are mostly designed for specific error types, looking for patterns or exact strings found to be frequently corrected in the training data.

Abbreviations sometimes are and sometimes are not accepted (Section 2). In order to capture more information on their usage we added Features 13 and 14. They count the number of abbreviations in the sentence and in the whole paragraph respectively. The general idea is that if an author has a tendency to use abbreviations, an editor does not perceive an individual abbreviation as inconsistent.

Features 47–55 recognize domain-related errors. Although the domain is unlikely to be directly decisive for distinguishing correct from incorrect sentences, some kinds of errors might coincide with individual domains. Our model does not take into account dependencies between features (Jurafsky and Martin, 2009, p. 238). However we examine their impact on the model's performance. They could be beneficial for other machine learning algorithms.

In order to detect spelling errors, some of the binary features check if all words in a sentence are present within specific sets, such as the vocabulary used in the correct training data, an American English dictionary[3], or WordNet[4]. We implement integer-valued counterparts for these features, because an absolute decision might be too restrictive.

Most of the real-valued features consist of probabilities computed in our language models. We compute maximum likelihood estimates of sentences based on different models. We use part-of-speech n-grams and token n-grams for $n \in \{1, 2, 3\}$ and a Hidden Markov Model. We also capture those n-grams in a sentence that yield the lowest probability compared to all other n-grams. Furthermore there are features that detect the position where a comma is most likely to be inserted with respect to the preceding and succeeding tokens and part-of-speech tags as well as the preceding, succeeding and superordinate constituents in the syntax tree. The same is done for inserting and deleting articles and substituting prepositions by other prepositions. Mostly we do not compute an isolated probability, but rather connect it with comparative probabilities. For instance, feature 82 does not only compute the probability of a comma before a pair of words, but returns the factor by which a comma is more likely than the word actually preceding the pair. That way the feature does depend on the subsequent word pair and also on the word to be substituted.

3.4 Machine Learning Approach

We participate in the binary and the probabilistic track using a logistic regression model. Logistic regression is capable of performing both probabilistic estimation and binary classification. Its training

[3] We used the `pyenchant` package: `http://pythonhosted.org/pyenchant/`

[4] `https://wordnet.princeton.edu/`

ID	Definition
15	contains *Van*
16	contains *van*
17	contains *n't*
18	contains is or us contraction (*where's*, *what's*, *that's*, *it's*, *let's*)
19	contains *'ve*
20	contains *'d*
21	first word not capitalized
22	dot after MATH or MATHDISP
23	wrong decade pluralization (*1**0's*)
24	reference description not capitalized (token *table*, *figure*, *lemma*, etc. before token *REF*)
25	abbreviation without dot (token *Tab*, *Fig*, *Figs*, *Eq*, etc. not ended or followed by .)
26	contains word unknown to `pyenchant`-package (binary version of Feature 11)
27	contains word not in vocabulary (binary version of Feature 9)
28	contains word unknown to WordNet (binary version of Feature 10)
29	two nouns connected by hyphen (parts of compound are all in WordNet as nouns)
30	contains small cardinal number in digits (e.g. *2*)
31	contains high cardinal number in letters (e.g. *seventeen*, *thirty*)
32	contains two cardinal numbers in letters, joined by a hyphen (e.g. *seventy-three*)
33	contains two cardinal numbers in letters in a row (e.g. *fifty two*, *one zero*)
34	wrong first letter after indefinite article (to a limited extent recognizes excepions: *an honest*, *a one-dimensional space*, *an SVM*, etc.)
35	contains the same token twice in a row (e.g. *The the*)
36	number mismatch between passive auxiliary verb and subject (number of nsubjpass and number of auxpass of a verb do not match according to the POS-tags.)
37	number mismatch between verb and subject (number of verb and nsubj do not match according to the POS-tags.)
38	number mismatch between article and head of a noun phrase
39	number mismatch between number modifier and head of a noun phrase
40	contains *vs* (not followed by .)
41	contains *vs* (ended with or followed by .)
42	contains *ie* or *ie.*
43	contains *i.e.*
44	contains *eg* or *eg.*
45	contains irregular verb with regular suffix (*lighted*, *builded*, etc.)
46	contains token *based* not followed by *on*
47–55	occurs in domains: Astrophysics, Chemistry, Computer Science, Economics/Management, Engineering, Human Sciences, Mathematics, Physics, Statistics

Table 2: Binary Features

phase is also not very time-consuming, which is beneficial for our feature selection procedure. It derives the probability of an observation x to belong to a particular class y from a linear combination of the observed feature vector f and a weight vector w (Jurafsky and Martin, 2009, pp. 231–239). It applies a logistic function to map the result of this linear combination to lie between 0 and 1. In the training phase the parameters in w are chosen to maximize the probability of the observed y values. During testing unseen samples are classified according to their probability computed by linearly combining their feature vectors with the very weight vector w that was determined in training.

3.5 Implementation

Our system is based on an object-oriented data model that provides information on the different datasets. Sentence objects comprise every piece of information at hand, including the actual tagged data and supplementary information such as lists of tokens and part-of-speech tags, a graph-like structure implementing the syntax tree, and a dictionary mapping tuples of indices in the token list to the

ID	Definition
56	average word length
57	max. gain of changing a preposition ($nmod(w)$ denotes preposition that modifies w): $\max(\{\frac{p_{modify}(w',w_i)}{p_{modify}(nmod(w_i),w_i)} : 1 \leq i \leq \lvert S\rvert \text{ AND } w' \in V \text{ AND } \exists j[1 \leq j \leq \lvert S\rvert \wedge nmod(w_i) = w_j]\})$,
58	max. gain of swapping the case of a first letter ($swap(w)$ is w with case of first letter swapped)): $\max(\{\frac{p_{ngram}(w_{i-2},w_{i-1},swap(w_i))}{p_{ngram}(w_{i-2},w_{i-1},w_i)} : 1 \leq i \leq \lvert S\rvert\})$,
59	maximum likelihood estimate (token unigrams): $\prod_{1\leq i\leq \lvert S\rvert} p(w_i)$
60	maximum likelihood estimate (POS-tag unigrams): $\prod_{1\leq i\leq \lvert S\rvert} p(t_i)$
61	maximum likelihood estimate (token bigrams): $\prod_{1\leq i\leq \lvert S\rvert} p(w_i \mid w_{i-1})$
62	maximum likelihood estimate (POS-tag bigrams): $\prod_{1\leq i\leq \lvert S\rvert} p(t_i \mid t_{i-1})$
63	maximum likelihood estimate (token trigrams): $\prod_{1\leq i\leq \lvert S\rvert} p(w_i \mid w_{i-2}w_{i-1})$
64	maximum likelihood estimate (POS-tag trigrams): $\prod_{1\leq i\leq \lvert S\rvert} p(t_i \mid t_{i-2}t_{i-1})$
65	maximum likelihood estimate (Hidden Markov Model): $\prod_{1\leq i\leq \lvert S\rvert} p(t_i \mid t_{i-1}) \cdot p(w_i \mid t_i)$
66	min. probability of any POS-tag trigram: $\min(\{p(t_i \mid t_{i-2}t_{i-1}) : 1 \leq i \leq \lvert S\rvert\})$
67	min. probability of any POS-tag bigram: $\min(\{p(t_i \mid t_{i-1}) : 1 \leq i \leq \lvert S\rvert\})$
68	min. probability of any POS-tag unigram: $\min(\{p(t_i) : 1 \leq i \leq \lvert S\rvert\})$
69	min. probability of any token trigram: $\min(\{p(w_i \mid w_{i-2}w_{i-1}) : 1 \leq i \leq \lvert S\rvert\})$
70	min. probability of any token bigram: $\min(\{p(w_i \mid w_{i-1}) : 1 \leq i \leq \lvert S\rvert\})$
71	min. probability of any token unigram: $\min(\{p(w_i) : 1 \leq i \leq \lvert S\rvert\})$
72	min. lexical probability of any token: $\min(\{p(w_i \mid t_i) : 1 \leq i \leq \lvert S\rvert\})$
73	fraction of tokens that are commas: $\frac{\lvert\{i:w_i=,:1\leq i\leq \lvert S\rvert\}\rvert}{\lvert S\rvert}$
74	max. gain of inserting comma after chunk: $\max(\{p(succ(g) = \text{,} \mid g) : g \in \textit{Tree}(S) \text{ AND } succ(g) \neq \text{,}\})$
75	max. gain of inserting comma before chunk: $\max(\{p(pred(g) = \text{,} \mid g) : g \in \textit{Tree}(S) \text{ AND } pred(g) \neq \text{,}\})$
76	max. gain of inserting comma within subtree: $\max(\{p(\text{,} \in g \mid g) : g \in \textit{Tree}(S) \text{ AND } \text{,} \notin child(g)\})$
77	max. gain of inserting article: $\max(\{\frac{p(\textbf{DT} \mid w_i)}{p(t_{i-1},w_i)} : 1 \leq i \leq \lvert S\rvert \text{ AND } t_{i-1} \neq \textbf{DT}\})$
78	max. gain of deleting article: $\max(\{\frac{p(t_{i-2},w_i)}{p(\textbf{DT} \mid w_i)} : 1 \leq i \leq \lvert S\rvert \text{ AND } t_{i-1} = \textbf{DT}\})$
79	max. gain of inserting comma after pair of POS-tags: $\max(\{\frac{p(\text{,} \mid t_{i-2}t_{i-1})}{p(t_i \mid t_{i-2}t_{i-1})} : 1 \leq i \leq \lvert S\rvert\})$
80	max. gain of inserting comma after pair of words: $\max(\{\frac{p(\text{,} \mid w_{i-2}w_{i-1})}{p(w_i \mid w_{i-2},w_{i-1})} : 1 \leq i \leq \lvert S\rvert\})$
81	max. gain of inserting comma before pair of POS-tags: $\max(\{\frac{p(\text{,} \mid t_{i+1}t_{i+2})}{p(t_i \mid t_{i+1}t_{i+2})} : 1 \leq i \leq \lvert S\rvert\})$
82	max. gain of inserting comma before pair of words: $\max(\{\frac{p(\text{,} \mid w_{i+1}w_{i+2})}{p(w_i \mid w_{i+1}w_{i+2})} : 1 \leq i \leq \lvert S\rvert\})$

Table 3: Real-valued features

dependency relation between the corresponding tokens. The object can hold both its correct and its incorrect versions. The Sentence class also implements all features and methods needed for data analysis. The purpose of the Corpus class is to gather and manipulate sentence information and transfer it to convenient output formats. It also holds a static object that encapsulates all functionality regarding language modeling.

Each step on the way to the final system is then implemented in a seperate script that accesses the data model described above. These steps can be combined to form a closed system or be extended to do further data analysis or to use machine learning approaches other than logistic regression.

For machine learning we used the `scikit-learn`[5] implementation of logistic regression.

4 Training

All sentences in the training set are used for training, that is, a sentence that needs correction enters the training set with both its original and its corrected version and thus introduces two samples with different labels to the training data, namely -1 for the correct version and $+1$ for the wrong version. Sentences that do not need modification have the label -1. To prevent single features from being predominant we scale all feature vectors using the `scikit-learn MaxAbsScaler`. It maps all our

[5] `http://scikit-learn.org/stable/`

values to lie between 0 and 1 by dividing by the largest absolute value that occurs in each feature during training. That way binary features and 0 values remain unaffected. Note that test data samples can still end up with feature values greater than 1, but all features will still be cut to reasonable sizes.

4.1 Feature Selection

In order to determine which of the features are helpful in an actual system, we first extract a small subset of binary features that all yield a high precision when classifying sentences of the development set solely based on their value. Seven of the features yield a precision of more than 90%, namely 17, 18, 19, 22, 23, 32, and 42. We train a logistic regression model using only these features. We evaluate the predictions of the model using the F1-scores for both tracks of the shared task, as defined in (Daudaravičius, 2015). Then we add each of the remaining features and keep the one that improves the F1-score most. We repeat that process until none of the features improves the score anymore. We perform this process on both training and development data seperately. Note that we do not include the features which encode the domain of a sentence. Instead, their combined impact is tested at the end of the procedure. If and only if adding them all yields an improvement, they are kept in the final model.

After having determined the most informative features, we account for distributional properties of our training set by adjusting some parameters. The training set is heavily biased towards correct sentences, because for each sentence (even with error) there is a correct version, but there is not necessarily a wrong version for each sentence. In order to make up for this imbalance we set the class weights inversely proportional to their respective proportions in the training data, as suggested by the `scikit-learn`-documentation[6]. Applying L1 regularization instead of L2 regularization gives us a minor performance boost, too. Table 4 shows the feature sets determined by the feature selection process along with the performances of the models on the different datasets with weighted classes and using L1 regularization.

[6] `http://scikit-learn.org/stable/modules/generated/sklearn.linear_model.LogisticRegression.html`

Model	precision	recall	F1
prob.u.L2	0.6655	0.7889	0.722
prob.w.L1	0.9333	0.7491	0.8311
bool.w.L1	0.3765	0.9480	0.5389

Table 6: Results on test data

Seeing how setting the right parameters can improve the performance of logistic regression, we do another feature selection on the development data. This time we weight the classes as described above and apply L1 regularization from the outset. That way we obtain the feature sets reported in Table 5.

5 Results

Since the results in Table 5 yield very promising results on the development data, we apply the two models to the test data, which yields comparable results (see models **bool.w.L1** and **prob.w.L1** in Table 6). Taking a closer look at the individual outcomes, however, reveals that they are by no means expressive. In the binary task our system almost always assigns *true* and thereby ensures the high recall. The precision on the other hand is relatively low and roughly matches the proportion of spurious sentences in the data. Hence our system would be outperformed by one that assigns *true* to all samples.

Our results on the probabilistic track look similar. Apart from a few instances to which our model assigns a probability around 95%, the estimations are always very close to 50%.

In order to examine the effects of a larger set of features we also apply the model resulting from feature selection on the training data to the test data for the probabilistic task. We expect that thanks to the multitude of features this model (**prob.u.L2**) will eventuate in a more diverse result. Despite the fact that as reported in Table 6 the F1-score drops by 11 points compared to our other system, the individual outcomes in fact seem to be much more expressive. The results still have a tendency to range around 50% but there are considerably more outliers and a lot more probabilities greater than 95%.

6 Discussion

6.1 Lessons learned

It is noticeable that we end up with very few features when performing the feature selection process

	features	F1-score
bool: training data	1, 7, 8, 9, 13, 14, 17, 18, 19, 20, 21, 22, 23, 24, 25, 29, 30, 31, 32, 34, 40, 42, 43, 44, 45, 46, 57, 58, 73, 77, 78, 79, 81, 82	0.4251
prob: training data	7, 9, 11, 14, 17, 18, 19, 22, 23, 32, 42, 47, 48, 49, 50, 51, 52, 53, 54, 55, 56, 60, 66, 67, 69, 70, 73, 74, 75	0.7500
bool: development data	1, 3, 8, 9, 13, 14, 17, 18, 19, 22, 23, 24, 25, 30, 31, 32, 34, 35, 41, 42 43, 45, 46, 57, 58, 77, 82	0.5264
prob: development data	11, 14, 17, 18, 19, 22, 23, 26, 30, 32, 39, 42, 56, 66, 67, 69, 70, 74, 75	0.7547

Table 4: F1-scores resulting from feature selection, weighting classes and applying L1 regularization afterwards

	features	F1-score
bool: development data	17, 18, 19, 22, 23, 32, 42, 69, 79	0.5701
prob: development data	17, 18, 19, 22, 23, 32, 42, 44, 72	0.8477

Table 5: F1-scores resulting from feature selection with classes weighted and L1 regularization applied from the outset

weighting the classes beforehand. By weighting the classes inversely proportional to their proportions in the training data, the system is immediately biased towards high probabilities for *true* labels, trying to compensate the superior number of *false* labels in the training data. Starting the feature selection process with high-precision features, the probability spikes whenever these features are 1. So both the model for the boolean track and the one for the probabilistic track start out with very high precisions. Due to the strong *true*-bias, all other probabilities are close to but still smaller than 50%, yielding a relatively high recall in the probabilistic system, which results in a very good performance according to the provided evaluation metric. The boolean system, on the other hand, has a very low recall, so in order to increase its F1-score, the precision is sacrificed during feature selection in favor of a better recall.

The feature selection processes show which features are more useful than others. We see that most of the integer-valued features that are valuable for readability assessment are never chosen for any model. A possible reason is that readability ease in scientific writing is not as important as in other domains, since the target readers are highly educated. A high linguistic complexity is rather characteristic for scientific writing and is possibly not perceived as a deficiency as much.

Interestingly the WordNet features (10, 28) do not work well, in contrast to the features using the `pyenchant`-package (11, 26).

The number of abbreviations in a paragraph (14) is chosen by every model so it is possible that an author's writing style throughout the rest of a document affects the editor's acceptance of individual sentences. It is worth considering to design more features that account for consistency in a paragraph.

Binary features often manage to improve the models, except for features 15 and 16, which is not surprising, given the fact that they denote exactly opposite properties and the model is not able to account for dependencies between features. Features 36–39 try to detect number disagreements and seem to perform poorly. Being based on both dependency trees and part-of-speech tags, these features rely on the correctness of the supplementary data, which in this case has been generated automatically, and hence cannot be guaranteed to be correct.

Our results also show that the domain-related features are not very helpful in combination with logistic regression. We can report that they only make a minor difference in the one model they entered.

Especially the models for probabilistic estimation are improved by features 66, 67 and 69, which are supposed to detect the most unlikely n-grams in a sentence. They are better in detecting local discrepancies in a sentence than the maximum likelihood estimation features 59–65, because an unlikely n-gram does not have much impact on the likelihood estimation of a sentence, so even a major error reflected in a very low n-gram probability can possibly go unnoticed. That cannot happen in the features 66–72.

The remaining features, dealing with the effects of insertion, deletion, and substitution of commas, articles, and prepositions, have positive impact on

some of the models, which is why we are confident that language modeling is the key to other helpful features yet to be found.

6.2 Evaluation Metric

The evaluation score works well for a system whose only purpose is the identification of erroneous sentences, so for the binary classification task the F1-score is perfectly suitable. However, it may be worth considering whether the information that a sentence is fine could be valuable, too. That might be the case whenever sentences must be further processed. In that case the accuracy metric might be the better choice, because it takes all correct classifications into account, whereas the F1-score does not reward instances correctly classified as *false*.

As for the probabilistic task, our results show that the evaluation score is not strict enough, and that it is prone to misjudge the expressiveness of the results. In fact, correctly assigning 1.0 to only one faulty sentence and 0.5 to all other sentences yields a score of 0.8571. The result is not as extreme if precision and recall are computed based on the mean absolute error, which results in 0.6667. This, still, clearly overestimates the quality of the results.

7 Related Work

As Daudaravičius (2015) states, a lot of scientists authoring scientific papers are nonnative English speakers. This insight suggests a relation of automatic evaluation of scientific writing to the field of language learner systems. Gamon (2010) mainly addresses article and preposition errors, which have shown to be frequent errors in the dataset provided for the AESW 2016 Shared Task, too. He uses language models on both a lexical and a syntactical level to find more likely alternatives for prepositions and articles with respect to the linguistic environment they occur in. He also bases some features on ratios of language model outcomes, rather than on individual probabilities, which is an approach that underlies many of our real-valued features.

Tetreault et al. (2010) examine how helpful parser output features are when modeling preposition usage. They present several phrase structure and dependency-based features, including left and right contexts of constituents in parse trees and the lexicals modified by a prepositional phrase.

For our features we extract those ideas from these works that seem the most promising for the challenge we encounter. But they both hold inspiration for even more features than those we implement in the course of our participation in the shared task and will be reconsidered in future work.

8 Conclusions

To detect spurious sentences in scientific writing we trained a logistic regression model. After a thorough data analysis, which gave us some profound insight into the types of errors occurring in scientific writing, we designed a number of features to detect these errors. We identified the most meaningful features by performing an incremental feature selection. Some of the resulting features show that corrections which seemed arbitrary might be justified by means of consistency of a text. We also used the probabilities of sentences according to language models as features, which our feature selection process determined to be helpful. Using the selected features our regression model achieved respectable results compared to our competitors' systems. Weighting our classes during the feature selection procedure, we accomplished a score to rank highest according to the evaluation metric in the probabilistic track of the task. However, we discovered that these results are very homogeneous and thus not expressive enough for a real life system. For future improvements of our system, we plan on developing an evaluation metric that takes the diversity of result data into account.

Acknowledgements

This work has been funded by the Klaus Tschira Foundation, Heidelberg, Germany. The second author has been supported by a HITS Ph.D. scholarship. We would like to thank Mark-Christoph Müller for giving feedback on earlier drafts of this report.

References

Vidas Daudaravičius. 2015. Automated evaluation of scientific writing: AESW shared task proposal. In *Proceedings of the Tenth Workshop on Innovative Use of NLP for Building Educational Applications,* Denver, Col., 4 June 2015, pages 56–63.

Michael Gamon. 2010. Using mostly native data to correct errors in learners' writing: A meta-classifier approach. In *Proceedings of Human Language Technologies 2010: The Conference of the North American Chapter of the Association for Computational Linguistics,* Los Angeles, Cal., 2–4 June 2010, pages 163–171.

Daniel Jurafsky and James H. Martin. 2009. *Speech and Language Processing*. Pearson Education, Upper Saddle River, N.J., second edition.

Emily Pitler and Ani Nenkova. 2008. Revisiting readability: A unified framework for predicting text quality,. In *Proceedings of the 2008 Conference on Empirical Methods in Natural Language Processing,* Waikiki, Honolulu, Hawaii, 25–27 October 2008, pages 186–195.

Joel Tetreault, Jennifer Foster, and Martin Chodorow. 2010. Using parse features for preposition selection and error detection. In *Proceedings of the ACL 2010 Conference Short Papers,* Uppsala, Sweden, 11–16 July 2010, pages 353–358.