Saarland University

Department of Computational Linguistics

Bachelor thesis

Automatic Detection of Linguistic Quality Violations

Author:
Jonathan Oberländer
Matriculation: 2539632

Supervisors:
Prof. Manfred Pinkal
Annemarie Friedrich
Alexis Palmer

Date of submission

Erklärung

Hiermit versichere ich, dass ich die vorliegende Arbeit selbstständig verfasst und keine anderen als die angegebenen Quellen und Hilfsmittel benutzt habe, dass alle Stellen der Arbeit, die wörtlich oder sinngemäß aus anderen Quellen übernommen wurden, als solche kenntlich gemacht und dass die Arbeit in gleicher oder ähnlicher Form noch keiner Prüfungsbehörde vorgelegt wurde.

Ort, Datum

Unterschrift

Declaration

I hereby confirm that the thesis presented here is my own work, with all assistance acknowledged.

Place, Date

Signature

Abstract Automatic summarization systems are usually evaluated on content coverage rather than linguistic quality factors, which leads to summaries containing many linguistic quality violations, like ungrammatical sentences. Building on the work of Friedrich et al. (2014), who annotated automatic summaries with a set of violation types, we first find a number of subtypes of ungrammaticality, and then we develop methods for detecting the clause-level violations ungrammaticality, inclusion of datelines and redundancy automatically. We obtain F-Scores of 58.6% for ungrammaticality, 87.8% for redundancy and 37.0% for redundancy.

Acknowledgements I would like to thank my supervisors Prof. Manfred Pinkal, Annemarie Friedrich and Alexis Palmer for their continuous support in the process of creating this thesis, and for providing me with helpful comments and guidance whenever needed. This work couldn't have been done without the work previously done by Marina Valeeva. I also want to thank Lukas Burk for helping me with R (R Core Team, 2013) and ggplot2 (Wickham, 2009).

Contents

1 Introduction

With an ever-growing amount of information, it is becoming increasingly important to reduce that information to a processible amount, by summarizing it. This is useful for people with time constraints who still want to read news, or for quickly skimming the content of a book, a website, a manual or a research paper. The scientific praxis of writing an abstract and an introduction doesn't just come out of nowhere, but is instead an effort to summarize the content of the paper at hand.

Most of the time though, at least when it comes to domains where writing a summary is not essential for the author, it is infeasible to do this manually, and there is thus a need for addressing the process with automated means.

Automatic summarization is the task of taking one or more documents as the input and producing a summary that contains the most important information from these documents.

Automatic summarization systems can be divided in *extractive* and *abstractive*, and in *single-document* and *multi-document* summarization systems.

A *single-document* system takes only one document as an input and produces a summary of that document, while a *multi-document* system summarizes multiple documents on the same topic, while still producing only one summary.

An *abstractive* system is one which infers knowledge from its input into an internal semantic representation and produces a novel text based on that knowledge, using language generation techniques. The task of constructing an abstractive summarization system is very complex, due to the aforementioned semantic representation. This is why most systems used today are *extractive*.

Extractive summarization systems first select relevant sentences from the source document(s). They then reorder them and finally apply sentence fusion (combining multiple sentences into one) and sentence compression (reducing the size of a single sentence by leaving out parts deemed less relevant) to shrink the resulting summary.

We are only concerned with extractive, multi-document summarization, because our data, as described in Section 3, consists of summaries produced by extractive multi-document summarization systems.

Evaluation of automatic summarization systems has largely been done on content coverage. Along with the task being hard in general, this is one of the reasons generated summaries often lack linguistic quality (Nenkova et al., 2011), meaning that they contain errors of various kinds. A new focus on linguistic quality (LQ) evaluation would presumably help decrease these issues and produce overall better summarization systems.

As a first step in this direction, Friedrich et al. (2014) created the *LQVSumm* corpus, which consists of multi-document summaries from the TAC 2011 shared task on Guided Summarization (Owczarzak and Dang, 2011), annotated manually with LQ violations on entity and discourse level.

The corpus also includes annotations on 50 summaries produced by G-Flow (Christensen et al., 2013), which we disregard, because the linguistic quality is much higher than in the rest of the corpus, along with the fact that the size of the annotated G-Flow summaries is much smaller.

Manual annotation is an expensive and time-consuming process, for which an automatic system could be an alternative. Our work deals with creating such a system that manages at least part of the annotation automatically. We develop and evaluate our system on top of the *LQVSumm* corpus. We also limit

ourselves to the clause-level violations listed in Friedrich et al. (2014), except for *no semantic relatedness* and *no discourse relation*.

The rest of this paper is organized as follows:

The next section discusses related work. In Section 3, we take a look at the *LQVSumm* corpus and the challenges it poses for our experiments and methods. We also demonstrate our subtype system in that section, to prepare for the experiments.

In Section 4 we describe and evaluate methods for annotating ungrammaticality (using decision trees with features extracted from parsing and the high-performing detection of the subtype *missing spaces*), datelines (using regular expressions) and redundancy (using n-gram overlap).

We achieve F-Scores of 58.6% for ungrammaticality, 87.8% for datelines and 37.0% for redundancies. We discuss our results in Section 5. Finally, in Section 6, we discuss future work and conclude.

2 Related Work

Pioneer work was done by Atwell (1987), who used the part-of-speech (POS) tagger CLAWS (Leech et al., 1983) alongside learned frequencies/likelihood of POS tag pairs in grammatical sentences to annotate errors using Markov chains.

They then added error tags to words in their tag dictionary — POS tags of words spelled similar to some word were specially marked on that word, when not in the POS tags of the word itself — and used them to make finding likely typing errors easier.

Most work in the field of automatic grammar checking seems to have been done in the area of second language learning. Bender et al. (2004) use the LKB parser and generator Copestake (2002) with the English Resource Grammar (Flickinger, 2000) to build *Arboretum*, a system that, when given an erroneous input sentence, detects that error and outputs a corrected version of the sentence. Their method uses manually selected mal-rules (Schneider and McCoy, 1998) to detect error types specific to second language learners of English.

While this seems to work for their specific domain, it does not transform well into ours: The mistakes language learners make, and the kind of ungrammaticality summarization systems produce are very different. Source texts that are being summarized are usually grammatical, and get their ungrammaticality during summarization from being compressed from two formerly grammatical sentences into one ungrammatical one, for example. Mistakes typical for learners of English as a second language either usually do not occur in automatic summarizations (such as misspelled words), or do happen, but due to a different cause (for example agreement).

Chodorow and Leacock (2000) build *ALEK*, a system to grade *TOEFL* essays. It does so by viewing the task as an extension of Word Sense Disambiguation, and collecting n-grams of POS-tags and function words, and then classifying using primitive statistical measures like *Mutual Information* and the χ^2 test. However, only the whole essay is classified into one of six possible scores, and this decision is only made by looking at the test subject's understanding of the correct usage of the three words *Concentrate*, *Interest* and *Knowledge* in context. Even more than Bender et al. (2004), this approach is not applicable to our task.

Sun et al. (2007), while also citing second language learning as their primary application, use a different approach: Instead of manually writing rules, they employ a supervised statistical learning method. They extract what they call *labeled sequential patterns* (ngrams associated with a class) from both erroneous and correct sentences in their training corpus, enrich them with other automatically computed features, and use them as input for classification models; an SVM (SVMlight, Joachims, 2002) and a Naïve Bayes classification model. The system achieves F-scores of around 80%, evaluated on correct and erroneous corpora of second language learning.

There also has been work in the area of assessing grammaticality of automatically generated summaries, namely Vadlapudi and Katragadda (2010). They used n-grams on POS-tag level and evaluated their system by looking at how well their classification correlates to *ROGUE* scores (Lin, 2004). Vadlapudi and Katragadda achieve a Spearman's ρ correlation of up to 77% to ROGUE scores manually assigned by annotators.

Wagner et al. (2007) compare various methods for detecting ungrammaticality:

Precision Grammar After parsing a sentence with the XLE LFG parser (Maxwell and Kaplan, 1996), a sentence is classified as ungrammatical based on the *-mark (which the parser assigns to any

sentence whose optimal solution uses a constraint marked as ungrammatical), parser exceptions during runtime and on whether there is a parse at all.

POS N-grams Using 10-fold cross validation, part-of-speech n-grams ($2 \leq n \leq 7$) are counted on the training data, and a sentence is classified as ungrammatical if it contains an n-gram that is below a threshold.

Decision Trees on XLE Output Using parser output as features (including whether a sentence is marked with a * (see above), the number of words and the number of parses, amongst others), a J48 decision tree is trained.

Decision Trees on N-grams The same experimental setup as in the previous method, but using the frequencies of the rarest n-gram for each possible n ($2 \leq n \leq 7$).

Decision Trees on Combined Feature Sets Using the features from the two previous methods.

Their error corpus is an artificial one, generated by introducing systematical errors into otherwise grammatical data. Their best method (**Decision Trees on Combined Feature Sets**) achieves an F-Score of 66.9%.

The *BLEU* score (Papineni et al., 2002) is a method for evaluating the effectiveness of a machine translation system, which uses a number of (human) reference translations to score an automatic translation. They showed that their score correlates highly with human evaluation, which has lead to the *BLEU* score having become the standard evaluation metric for machine translation.

Hatzivassiloglou et al. (1999) implement a system for redundancy detection on the level of "text units" (paragraphs or sentences) using machine learning techniques. As primitive features, they use:

Word co-occurence A very similar feature to our **Unigrams** approach in Section 4.3.1.

Matching noun phrases Using *LinkIt* (Wacholder, 1998), they detect simplex noun phrases and match those sharing a head

WordNet synonyms Matching words that occur in the same *WordNet* (Fellbaum, 2005) synset.

Common semantic classes for verbs Matching verbs that share the same semantic class (Levin, 1993)

Shared proper nouns Matching shared proper nouns, identified by *Alembic* (Aberdeen et al., 1995)

These features were then optionally normalized by sentence length or relative frequency of the feature occurence.

In addition to these primitive features, they also used three composite features, which combine pairs of primitive features and are applied as restrictions on their primitive features:

Order Whether an element (word) pair had the same relative order in both sentences/textual units, i.e. word X comes before word Y in both textual units, or vice versa, for some words X and Y

Distance Whether two elements are no farther than some value in both textual units

Primitive Elements are restricted to a specific primitive feature

They then trained a machine learning algorithm (Cohen, 1996) and evaluated on a news text corpus, for which they achieved a precision of 60.5% and a recall of 36.6%.

3 LQVSumm

The LQVSumm corpus consists of annotations on summaries produced by systems participating in the TAC 2011 Guided Summarization Task (Owczarzak and Dang, 2011). It contains 44 "topics", each of which contain 44 summaries (except for the first topic, which, for some reason, contains 43 summaries). Summaries in the same topic are based on the same source documents.

Because the annotation of LQVSumm was done using MMAX2 (Müller and Strube, 2006), the summaries exist in its XML output format. Further annotations done from our side are applied to the existing files, but in seperate tags splitting corpus data from classification or added features.

We split the corpus into three parts, while making sure that topics weren't spread across sets: *dev-1* and *dev-2* are development sets, *test* is our test set. We have two development sets, because for some methods, we developed on one part and trained on another. *dev-1* contains 351 summaries, *dev-2* contains 352, that is 8 topics per development set and about 20% of the corpus each. The test set *test* contains the other 60%, making up 1232 summaries and 28 topics.

The breakdown of the distribution of topics across development and test sets is shown in Table 3.

Set	**Number of summaries**	**Topics**
dev-1	351	D1101 – D1108
dev-2	352	D1109 – D1116
test	1232	D1117 – D1144
Overall	1935	D1101 – D1144

Table 1. Distribution and count of summaries across development and test sets

Our annotations, experiments and the evaluation are all done on sentence basis. We use Stanford CoreNLP (Manning et al., 2014) for sentence splitting. This is necessary to have classifiable entities, but introduces inaccuracies, because LQVSumm is annotated on clause level, clause meaning an arbitrary substring of a summary, which is often a sentence, but also sometimes spanning over sentence bounds.

3.1 Violation types

Some of these annotations were done on the *entity level*, others on the *clause level*. Our work focusses on four clause-level violations.

When a clause is annotated with an *incomplete sentence* violation (INCOMPLSN), the sentence was cut off, most likely due to a summarization system trying to fit as much in the summary as the limit for the task allowed, or due to sentence compression. An example of such a clause would be:

Neighbors and family members saw no

Annotations of the violation type *inclusion of datelines* (INCLDATE) mark timestamps that have ended up in the summary. These fragments are the result of the summaries' domain; news articles. Datelines appear in a couple of different variations, most of the time at the beginning of a sentence:

MADRID, Feb. 15 (Xinhua) Rabei Ousmane Sayed Ahmed,

This example also shows that multiple violations can of course happen in the same sentence, as it is also tagged as containing an *incomplete sentence* violation.

The violation type *other ungrammatical form* is used for any other ungrammaticality happening in a single phrase. Therefore, we find the biggest variety of violations in here:

Valley Veterinary Bismarck, North Dakota, a veterinarian, Carlson, has been worried that pet owners to the Menu Foods on its web site.

A Boeing 737-400 plane with 102 people on board crashed into amountain in Indonesia's West Sulawesi province on Monday, killingat least 90 people.

//www.menufoods.com/recall – early Saturday.

The remaining three violation types apply to multiple phrases. A *redundant information* violation (REDUNINF) means two phrases share a lot or all of their information. There are instances of this violation where the two phrases are exactly the same, and others where the information is rephrased in the second sentence:

Representatives from the commercial fishing industry, environmental groups and government regulators are discussing ways to strengthen information sharing and cooperation among regional organizations to track and manage tuna stocks.

Representatives, including the commercial fishing industry, government regulators and environmental groups, plan to discuss ways to strengthen information sharing and cooperation among regional organizations to better manage tuna stock and adopt an action plan, the Japanese Fisheries Agency said in a statement.

Because the instance counts for the last remaining violation types (*no semantic relatedness* and *no discourse relation*) were so low (only 142 instances of *no semantic relatedness* and 91 instances of *no discourse relation*), they were disregarded and we didn't try to a develop methods to detect them.

Suggestions for possible detection methods for the omitted violation types are made in Section 6.

3.2 Subtype system

As a first step, we inspect the corpus. We find that the type system of the LQVSumm corpus (Friedrich et al., 2014) is not detailed enough for our purposes and has a few shortcomings: Especially under the label of *other ungrammatical form*, we see many different types of errors combined.

Since we assume that dividing the problem into smaller, easier problems could help us detect *other ungrammatical form* violations, we decide to define a number of subtypes for this violation:

missing spaces

This is the most common type of error: The sentence contains a word that does not exist. In almost all cases, this happens when whitespace between two words is missing.

Example: *A strong earthquake measuring 7.8 magnitude struck* ***Wenchuancounty*** *of Sichuan Province on Monday, leaving at least* ***12,000people*** *died and thousands more injured.*

missing words

One or multiple words are clearly missing. These seem to most often be function words such as articles or pronouns, rather than content words. In the example, an underscore marks the position where a word, probably "was" is missing.

Example: *An Israeli woman _ killed and 11 others were wounded in the suicide bombing at a shopping mall in southern Israeli town of Dimona.*

punctuation error

Most of the time, this means: Punctuation is missing, but it can also mean there is something else wrong with punctuation in that sentence which makes it ungrammatical, as can be seen in the example: Unbalanced parantheses.

Example: *China has allocated 200 million yuan (million dollars for disaster relief work after an earthquake rocked the country's killing at least seven people, state reported on Tuesday.*

capitalization error

Words that should be capitalized are not or words that should not be capitalized are.

Example: *earlier on **m**onday **g**erman chancellor **a**ngela **m**erkel and foreign minister **f**rank **w**alter **s**teinmeier offered their condolences to **c**hina over the heavy loss of life in the powerful earthquake that hit **c**hina's southwestern province of **s**ichuan.*

unparsable

This subtype means that the (human) annotator was unable to make sense of the sentence and considers it to be not correctly parsable. Most instances of this subtype seem to happen when sentence fusion occurs; the summarization system combined two or more sentences in an ungrammatical or meaningless way.

Example: *All of those provinces and Chongqing, a special municipality _ deepest condolences to those who lost their loved ones in the devastating natural disaster.*

heading

The sentence contain (usually at the beginning) a sequence of capitalized or otherwise heading-like words that aren't part of a dateline.

Example: *THE CURRENT FIX: Internet applications such as firewalls and spam filters attempt to control security threats.*

incomplete sentence

The type system of the LQVSumm corpus (Friedrich et al., 2014) defines an *incomplete sentence* violation as words being cut off at the end of a sentence. A couple of times though, this also occurs at the beginning of a sentence. We restructure the type system by treating all types of incomplete sentences as this subtype of *other ungrammatical form.*

Example: *A Palestinian suicide bomber detonated an explosive belt at a commercial center in Dimona on.*

should be other type

Sometimes sentences that are tagged as *other ungrammaticality* should (also) be tagged as another type, such as sentences containing datelines. This doesn't necessarily mean that the sentence contains no other ungrammaticalities. This subtype be caused by inconsistencies in the corpus, or from the process of mapping the annotated violations to sentences. While the following example is an *incomplete sentence*, it also contains a dateline.

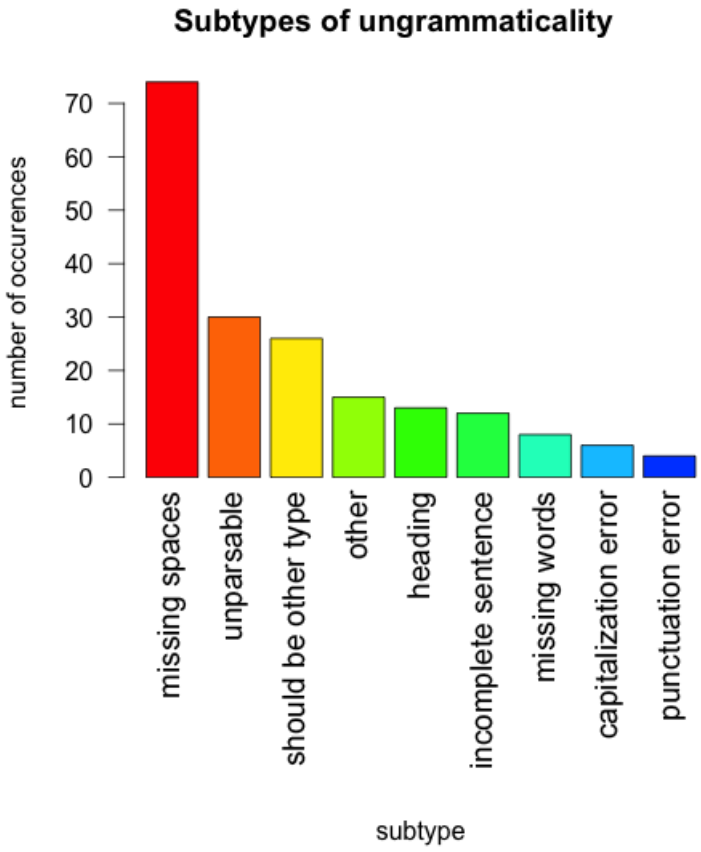

Figure 1. Subtypes of ungrammaticality in *dev-2*. Where the annotator didn't know what subtype to annotate the sentence with, they selected "other".

> Example: *its deepest sympathy to the Israeli people,* ***00 a.m. local time (0800 GMT)*** *as a suicide bomber detonated an explosive belt in the commercial center, according to local media reports.*

After manually counting and investigating these subtypes on *dev-1* we then mark these types in *dev-2* for every sentence that is tagged as *other ungrammatical form* in the corpus. We allow sentences to have multiple subtypes and also annotate *incomplete sentence* violations with additional (sub)types, if applicable. The frequency distribution of the subtypes in *dev-2* can be seen in Figure 1.

Perhaps surprisingly, not all subtypes occur with the same frequency, in fact, one of them makes up almost half of the ungrammaticality violations. We exploit that fact in Section 4.1 by building a detection method specifically for that subtype.

It is to be noted, that from here on, we consider *incomplete sentence* to be a subtype of *other ungrammatical form*.

4 Experiments

We conduct a number of experiments to find reliable ways to detect different kinds of linguistic quality violations. If possible, we would like to eventually be able to reproduce the annotations of our types on the LQVSumm corpus as well as annotate violations in other corpora.

We use the standard metrics of Precision, Recall, F-Score, and Accuracy, which can be seen in Figure 4.

$$\text{Precision} = \frac{\text{true positives}}{\text{true positives} + \text{false positives}}$$
$$\text{Recall} = \frac{\text{true positives}}{\text{true positives} + \text{false negatives}}$$
$$\text{F-Score} = 2 * \frac{\text{precision} * \text{recall}}{\text{precision} + \text{recall}}$$
$$\text{Accuracy} = \frac{\text{true positives} + \text{true negatives}}{\text{all instances}}$$

Figure 2. Standard metrics Precision, Recall, F-Score and Accuracy

4.1 Detecting Ungrammaticality

We start with the violation type *other ungrammatical type* (plus *incomplete sentence*, as explained earlier).

Perhaps the most noticeable result of our corpus study is the large portion of *missing spaces* violations, meaning clauses that include tokens which aren't correct words. Almost all of these cases come from a missing space between two words forming tokens such as *reportsreaching* or *Wenchuancounty*. As there is such a large amount of this type, we believe finding a reliable detection method for this subtype would presumably significantly boost detection of ungrammaticality in general.

4.1.1 The UnknownTokens method

We call our straightforward approach to identifying missing spaces **UnknownTokens**: Tagging a sentence as containing a *missing spaces* ungrammaticality if and only if there is a token in the sentence that isn't a "known" token, i.e. one that doesn't exist in a list of known words. In our setting, we can easily deduce such a list from the source documents of the automatic summarization process. If the source document is available and used as a base for our vocabulary, we call that approach **UnknownTokens-source**.

In case the original documents aren't available or this method should be used in a different setting, we also investigate how to adapt our approach to that. This approach has a few steps, which we will all evaluate individually:

- First, an initial list of words is created from a large text corpus (**UnknownTokens-gw**); we use the first 20% of the AFE part of the gigaword corpus (Graff et al., 2003). If the corpus is large and general enough, this will probably cover a large vocabulary.
- Because this will not include words for named entities yet, we only consider a token to be an unknown token, if it is not tagged as a Named Entity by a Named Entity Recognizer (**UnknownTokens-ner**). We employ the Stanford Named Entity Recognizer (Finkel et al., 2005), which is a widely used, state-of-the-art NER that comes with a model for the English language.

- In order to improve upon the NER and further increase recall, we use two simple heuristics (**UnknownTokens-heur**): Firstly, we assume that words that start with a capital letter and are only followed by lower case letters are named entities and therefore no unknown tokens. Next, we throw out words containing hypens.
- Finally, we automatically check whether an unknown token has an entry in the free online encyclopedia Wikipedia (**UnknownTokens-wiki**).

The combined method **UnknownTokens-gw+heur+ner+wiki** is what we call **UnknownTokens-general**.

We automatically label any tokens that are neither on our list of known tokens from GigaWord, nor tagged as a named entity as "unknown" or caught by either our heuristic or the wikipedia-based addition and tag the sentence as containing unknown tokens.

4.1.2 Evaluation of UnknownTokens

We evaluate how effective **UnknownTokens** is in detecting *missing spaces* violations, both with (**UnknownTokens-source**) and without (**UnknownTokens-general**) access to the source documents, and compare it to a **baseline**, which always assumes the most common case (no *missing spaces* violation). The standard metrics precision, recall and F-score (see Figure 4) are used for evaluation.

Table 2 shows the results of the evaluation of both variations of **UnknownTokens** on *dev-2*. Because the amount of sentences not containing a missing spaces violation is much larger than the ones that do, the weighted metric "Macro-average" is provided, too:

Macro-averages are the arithmetic means of precision and recall of both classes. The new F-score is calculated from the new scores.

Both **UnknownTokens-general** and **UnknownTokens-source** improve vastly over the baseline, but it is clear, that if the source text is available, it should be used to infer known words.

4.1.3 The RandomForest-UT method

To detect more general forms of ungrammaticality, several other methods were investigated. Following Wagner et al. (2007), we train decision trees from outputs of several systems that on their own have not proven sufficient for this task. This experiment was then evaluated on *test*.

We train a language model on parts (the same parts we used as a word source in **UnknownTokens-general** (section 4.1)) of the Gigaword corpus(Graff et al., 2003), extract perplexity scores for the sentences in our test set, and annotate them automatically. Perplexity scores can be described as how surprised the system is when seeing the end of a test sentence, based on the information in the language model.

For the language model generation and the extraction of perplexity scores, we employ SRILM (Stolcke et al., 2002), a language modeling toolkit. The perplexity scores alone had shown not to be effective in detecting ungrammaticality.

We then use a RandomForest (Breiman, 2001) to learn decision trees. RandomForest works by randomly selecting subsets of training instances along with random subsets of features to train a number of decision trees and then classifying test instances with what the majority of trees are classifying the instance with.

	Missing spaces			No missing spaces			Macro-average	Accuracy
	P	**R**	**F**	**P**	**R**	**F**	**F**	
Baseline	0.0	0.0	0.0	94.8	**100**	97.3	48.7	94.8
UT-gw	15.0	**98.7**	26.0	**99.9**	69.1	81.7	68.2	70.7
UTgw+heur	30.5	97.3	46.5	99.8	87.8	93.4	76.5	88.3
UT-gw+heur+ner	35.5	97.3	52.0	99.8	90.3	94.8	78.6	90.6
UT-gw+heur+ner+wiki	70.3	96.0	81.2	99.8	97.8	98.8	90.6	97.7
UT-source	**95.9**	94.6	**95.2**	99.7	99.7	**99.7**	**97.5**	**99.5**

Table 2. Evaluation of **UnknownTokens** on *dev-2*, in percent. **UT-wordnet+heur+ner+wiki** is equivalent to **UnknownTokens-general**. **wn** = Wordnet-based dictionary, **heur** = title case heuristics, **ner** = Stanford NER, **wiki** = Wikipedia entry check

	Precision	Recall	F-Score	Accuracy
Baseline	-	0	-	77.7
RandomForest-UT	65.5	44.2	52.8	82.3
RandomForest-UT+parser	**72.8**	**49.1**	**58.6**	**84.5**

Table 3. Evaluation of **RandomForest-UT** and **RandomForest-UT+parser** on *test*, in percent, based on the class of ungrammatical sentences. As Baseline, we classified every sentence with the most-frequent class, "not ungrammatical".

We first use three features for building our decision trees:

- **language model perplexity** is the perplexity calculated for a sentence by the language model, following Sun et al. (2007). This is a numeric feature. It proved to not be sufficient for detecting ungrammaticality on its own.
- **missing spaces** is the classification of **UnknownTokens-source** for a given sentence, as described in section 4.1. This is a binary feature — either a sentence was classified as containing the violation, or as not containing it.
- **numberofwords** is the number of words in the sentence. We follow Wagner et al. (2007) here, the intuition is that ungrammatical sentences will often be extraordinarily long, or very short:

  Former US President George W Bush, Who Will attend a memorial service on Tuesday, Virginia, and the university of at least 33 people Were killed, in the Worst shooting rampage in modern history, the White House said.

  The report of a

For implementing this task, we use the machine learning toolkit WEKA (Hall et al., 2009).

4.1.4 The RandomForest-UT+parser method

We also investigated whether parser output would improve the results. For this task, we utilized the HPSG parser Answer Constraint Engine (Packart, 2011) with the English Resource Grammar (Copestake, 2002) to parse the entire corpus and annotate it with output statistics of the parser: Following Wagner et al. (2007) loosely, we extracted the number of readings/parses, the amount of memory the parser used for the parse and the success status, which could be either "ok" or "ignored", the latter appearing when the parsing failed entirely for some reason.

We then added these three features to **RandomForest-UT** and call the new method **RandomForest-UT+parser**.

4.1.5 Evaluation of RandomForest

We performed 10-fold cross-validation with both **RandomForest-UT** and **RandomForest-UT+parser** on *test*, the results of which can be seen in Table 3. These evaluations are done on the class of ungrammatical sentences, evaluating it on both classes leads to much higher scores (with a weighted F-score of 83.4% for both "ungrammatical" and "not ungrammatical").

Feature	Decrease in F-Score
Parser Readings	5.7
Number of Words	2.2
Language Model Perplexity	1.4
Parser RAM	1.3
UnknownTokens Output	0.8
Parser Status	0.4

Table 4. Ablation study of **RandomForest-UT+parser**. The numbers denote the decrease of the F-Score if that feature were to be taken out.

As evident from the table, including the parser output statistics as features (**RandomForest-UT+parser**) significantly improves upon the performance of just **RandomForest-UT**. We generally value precision more than recall, because we prefer it if our system misses a case of ungrammaticality to it falsely classifying a grammatical sentence as ungrammatical.

An ablation study can be seen in Table 4. The numbers in the table describe by how much the F-score goes down if we were to remove a specific feature. No number being very high indicates that we have some redundancy of information in the features here; none of the features have a lot of *exclusive* information about ungrammaticality.

In this case, we consider redundancy to be a good thing, as it should make the output more stable.

4.2 Datelines

Source texts from news articles, like the ones in the TAC dataset, often contain datelines, which sometimes make it into the generated summary. While in the context of the article datelines made sense (often displayed smaller or otherwise visually distinctive), they are not needed, distracting and confusing in the summaries.

Datelines take many different forms, with some including a location. There is also a variety of appearing time formats, and some of them don't include timestamps at all:

BLACKSBURG, Virginia 2007-04-16 18:34: 44 UTC *A gunman opened fire in a dorm and classroom at Virginia Tech on Monday, killing at least 30 people in the deadliest shooting rampage in U.S. history.*

BERLIN, May 13(Xinhua) *The German government announced on Tuesday that it is to provide 500, 000 euros(around 770, 000 U.S. dollars) in aid for earthquake victims in Sichuan Province of China.*

00 a.m.*People are panicking.*

4.2.1 Method

The patterns emerging through this violation are so regular, that we use regular expressions to detect them. A few iterations of the expression were created, but so far the highest results were achieved using the regular expression

```
UTC|^\d{4}-\d{2}-\d{2}|^[A-Z]{3,}|^(Jan|Feb|Mar|Apr|May|Jun|Jul|Aug|Sep|Oct
    |Nov|Dec)
```

	Precision	Recall	F-Score	Accuracy
Baseline	-	0	-	89.5
Regex	86.0	89.7	87.8	97.4

Table 5. Evaluation of **Regex** on *test*.

The expression has four parts, of which only one has to match to make the system classify a sentence as containing datelines:

- The first part matches the string "UTC" (Coordinated Universal Time, a common fragment in datelines)
- The second one matches numeric dates like *2014-04-13*
- The third one matches capitalized words that are longer than 4 letters and are therefore expected not to be abbreviations (an example would be *LONDON, Feb. 4 (Xinhua)*)
- And the final part matches abbreviated names of months.

Only "UTC" can occur anywhere, everything else needs to occur at the beginning of a sentence to be counted.

4.2.2 Evaluation

This very simple approach was evaluated on *test* and leads to very promising results that can be seen in Table 5. As a baseline, we again classified all sentences with the most-frequent class: not containing a dateline violation.

When looking into the inaccurately detected instances, the inconsistencies of the LQVSumm corpus make up most of the misses:

IVORY TUSKS

is classified as containing a dateline by the regular expression, but is not annotated as such in the corpus. An example of a false negative would be:

00 a.m. local time (0800 GMT)

which our system doesn't match.

4.3 Redundancy

Automatic summaries constructed from multiple source texts on the same topic often suffer from redundancy, as sentences that contain the same or largely the same information are occasionally included in the summary. As a consequence, reading fluency is impaired.

While — in contrast to many other linguistic quality violations — this also has an impact on traditional evaluation scores for automatic summarization systems (because the content coverage will be lower than it could be), it is also considered to be a problem of our domain.

In the LQVSumm corpus, different degrees of redundant sentence pairs occur. Some sentence pairs annotated as redundant contain similar information, but using completely different phrasing and focus:

According to a survey by the State Food and Drug Administration, 65 percent of the respondents worried about the food safety situation in China.

Food and drug safety has become a major concern of Chinese people.

Other pairs of sentences are apparently at least partly composed of the same source sentence, but are still different. One of the sentences might contain additional information and the two sentences might still be of very different length:

Cyclone Sidr, described as the worst storm in years to hit low-lying and disaster-prone Bangladesh, crashed into the southwestern coast Thursday night before sweeping north over the capital Dhaka.

The cyclone hit the southwestern coast of Bangladesh on Thursday before sweeping north to the capital Dhaka.

Finally, sometimes subsequent sentences have marginal or even no differences: *An unknown number of cats and dogs suffered kidney failure and about 10 died after eating the affected pet food, Menu Foods said in announcing the North American recall.*

An unknown number of cats and dogs suffered kidney failure, and about 10 died after eating the affected pet food, Menu Foods said in announcing the North American recall.

We are seeking to find all of these sentences, but concentrate on the latter two cases where the redundancy is obvious for humans.

4.3.1 Method

Our method (**Unigrams**) consists of a simple unigram overlap score, which we found to outperform more sophisticated approaches:

- First, we remove all non-alphanumeric characters from both sentences and split them into words
- We then treat the words as a set and divide the cardinality of the intersection of each sentence's set by the number of possible overlaps, which will be the number of unique tokens in the smaller sentence.
- A given threshold is then used to classify instances.

A number of other methods are also implemented: **Bigrams** is a variation of the **Unigrams** method which uses bigrams instead of unigrams. **Combined** is using a score that the mean of the **Unigrams** and **Bigrams** scores.

Levenshtein consists of the standard edit distance (Levenshtein, 1966) on word level. A threshold of 2 yielded best results on our development set, meaning that sentences are classified as containing redundancies if their word-level Levenshtein distance is 1 or 0. Prior to computation of the score, non-alphanumeric characters were again pruned from the sentences.

Finally, as a further **Baseline** we classify all sentence pairs as containing redundancies.

We briefly considered using BLEU scores (Papineni et al., 2002) for redundancy detection (with the rationale that if the second sentence is a "good translation" of the first one, it might be redundant), but BLEU scores get their performance from having multiple reference translations available, which is not the case here.

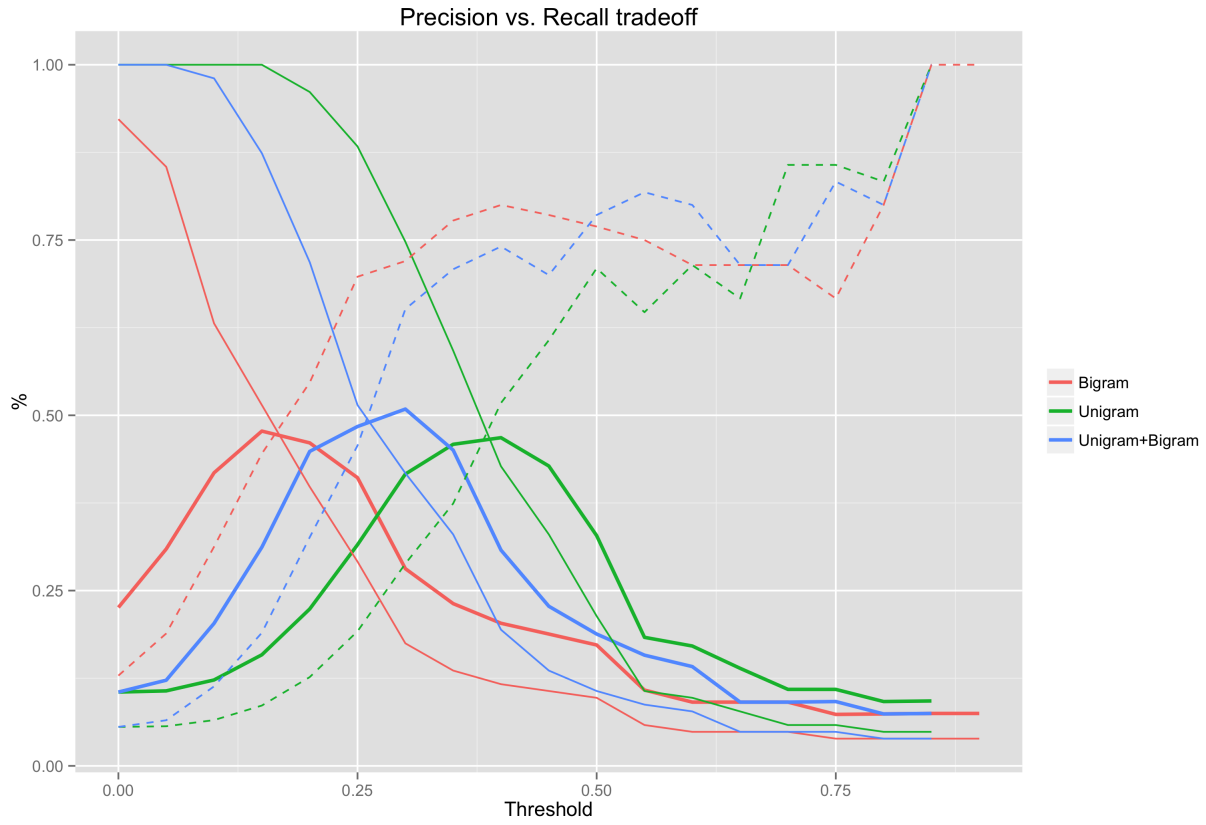

Figure 3. Performance of various thresholds for **Unigrams**, **Bigrams**, and **Combined** in *dev-1* and *dev-2*. Continuous lines are recall, dashed lines are precision and bold lines are F-score.

4.3.2 Evaluation

In the corpus, redundancy isn't annotated on sentence base, but rather on clause level, where "clause" is an arbitrary substring of the summary text. That means that redundancy annotations can span over many sentences, can skip sentences inbetween and can be contained in a single sentence.

For our purposes, we only consider redundancies that occur in two subsequent sentences and our evaluation metrics are based on those.

To obtain a good threshold for **Unigrams**, **Bigrams**, and **Combined**, we compare their performance on the development sets *dev-1* and *dev-2*. The results can be seen in Figure 3.

For recall, **Unigrams** outperforms **Bigrams**, while for precision **Bigrams** is predominant.

As is to be expected, higher thresholds correlate with higher precision and lower recall. And while the F-scores are given in Figure 3 for informational purposes, they are not what we want to optimize; while recall is desirable, we feel that precision is much more important for a system that annotates violations.

As can be seen in Table 6, we therefore choose a threshold of 0.4 for the **Bigrams** and **Combined**, and one of 0.5 for **Unigrams**, because there appear to be local maxima at that point with still some significant amount of recall.

The results can be seen in Table 7. Evidently, the **Unigrams** method outperforms **Levenshtein** and the **Baseline** as well as the other n-gram methods.

Unigrams	**Bigrams**	**Combined**
0.5	0.4	0.4

Table 6. Thresholds for **Unigrams**, **Bigrams**, and **Combined** determined on *dev-1+2*

	Precision	Recall	F-Score
Baseline	4.5%	**100%**	8.7%
Levenshtein	15.8%	17.3%	3.1%
Unigrams	**58.0%**	28.2%	**37.0%**
Bigrams	55.6%	14.5%	22.9%
Combined	56.8%	24.3%	34.0%

Table 7. Evaluation of **Unigrams**, **Bigrams**, and **Combined** on everything but *dev-1* and *dev-2*

We unsuccessfully tried to increase our performance by a few means:

- stemming the words with the Porter Stemming Algorithm (Porter, 1980) from NLTK (Bird et al., 2009)
- part-of-speech-tagging the tokens with NLTK and only counting overlaps that include a verb (with the motivation of ruling out noun-based false positives)
- looking up the first synset of each token in *WordNet* (Fellbaum, 2005), and building the intersections between the sets by considering two tokens to be the same if the first word is included in the second one's synset.

Neither of these methods had any positive effect on our results.

5 Discussion

Detecting linguistic quality violations automatically is not a trivial task. It is not always easy for humans to produce consistent annotations and annotations aren't necessarily highly consistent across humans, as can be seen in the inter-annotator agreement in Friedrich et al. (2014).

Despite that, sometimes the easier methods outperformed the more sophisticated approaches.

In theory, **UnknownTokens-source** should return perfect results, with 100% recall, precision and F-score. So why doesn't it? Some of this can probably be attributed to annotation mistakes during the annotation process of the subtypes. But manual inspection of the false positives showed that as the datelines (Section 4.2) are not a part of the text of the source document (which we took the tokens from), geographical names only present in the datelines were wrongly assumed to be unknown tokens.

UnknownTokens-general's precision is weakened by a few words such as "predisposes" that were neither found in our Gigaword tokens, nor detected to be a named entity by either of our methods. The reader should keep in mind that because we are only evaluating this on *dev-2* (as the subtypes are only annotated there) a couple of words can make a big difference.

Without finding evidence for this happening in our results, in theory cases could occur where a token is a known token (and rightly so), but can also be formed from two other tokens, when missing a space. This could not be correctly classified without contextual knowledge. A fictitious example could be:

To try this taskmaster the other one first.

As taskmaster is an English word, it might occur in the list of known tokens, but will still be annotated as missing a space, because *in this context* it is a contraction of *task* and *master*.

This means that achieving 100% precision *with this method* is theoretically impossible.

We believe our method of detecting datelines is close to the optimum achievable with a regular expression based approach without accidentally detecting times, time ranges or locations that are mentioned in the text as datelines. When looking into the false positives, our decision to match abbreviations of month names leads to some sentences being classified as containing a dateline which actually do not contain one:

Mar*uca-Kovac said he phoned her at about 1 p.m. Wednesday, telling her he had left*

Other false positives could have been annotated as containing datelines in the corpus, but weren't:

Nov*ember 17 itself fired a rocket against the US embassy in 1996*

Our biggest loss in terms of false negatives is that we don't detect datelines of the form:

14(Xinhua)

Out of all of our best methods, our **Unigrams** method for detection of redundancies achieves the lowest scores. Part of that can be attributed to the simplicity of our approach: A token based method can only get so far when a subsequent sentence contains paraphrased content from the previous sentence.

Defense Secretary Robert Gates *named an independent review panel* ***Friday*** *to investigate what he called an unacceptable situation" in outpatient care at* ***Walter Reed Army Medical Center, and he*** *said that some soldiers most directly involved" in the problems have been removed from their positions.*

Defense Secretary Robert *M.* ***Gates*** *said* ***Friday*** *that he would move swiftly to improve conditions for wounded soldiers being treated as outpatients at* ***Walter Reed Army Medical Center, and he*** *warned that senior officials would be held accountable for the poor conditions.*

While this example has some token overlap, the paraphrased content fails to get noticed by our system.

Another cause of a relatively low score is that in the LQVSumm corpus, redundancy annotations can be done on a smaller scale than what is classified with using our method. For example, a repeated phrase within two long sentences could be marked in the corpus, but our or similar approaches will either not find these cases, or classify too much.

The LQVSumm corpus that we used for all of our experiments is, unfortunately, not large enough to have a wide coverage of all violations. This can be seen most prominently when trying to detect redundancy (Section 4.3): While the corpus itself contains 504 *redundant information* violations (see Friedrich et al., 2014), only 173 of them could be used by us, as we needed redundancies that occured in subsequent sentences and only within the bounds of those. However, there's little to no alternatives available.

Apart from its size, the corpus is also sometimes annotated inconsistently or at least challenging for our purposes. In general, annotations are done on the clause level, where clauses are arbitrary substrings of the summary. As we classify and evaluate on sentences produced by the sentence splitter of Stanford CoreNLP (Manning et al., 2014), this introduces noise. For our purposes, a corpus that had been segmented before annotation would be preferable.

As can be seen in Section 3.2, the instances of *other ungrammatical form* could have been divided into several types, which would help the automatic classification.

A legitimate criticism of our work is that our methods, experiments, and results are tailored to the LQVSumm corpus and to the summarization systems that participated in the TAC 2011 Guided Summarization task (Owczarzak and Dang, 2011).

6 Conclusion & Future Work

In this paper, we have presented a method to detect the kind of ungrammaticality that occurs in automatically generated extractive multi-document summarizations. While the results are good, they do rely on a certain type of ungrammaticality occuring often (missing spaces). Future work could concentrate on improving other grammaticality violations, so that not only the accuracy rises, but also the generality. Such a system could potentially detect ungrammaticality in a much broader domain.

We have also shown a simple but high-performing method of detecting datelines. We believe our results are close to what a human could achieve and don't see much room for improvement here, at least not with a variant of our method.

Finally we developed a basic method of detecting redundancy. With the lowest performance of all methods in this paper, redundancy detection is requiring much more work. While we based our approach only on the two subsequent sentences on which redundancy was to be tested, future work could include information available in the context of these sentences or even, as we did in **UnknownTokens-source**, information from the source documents.

One could also try following Hatzivassiloglou et al. (1999) closely and applying some form of machine learning as well, possibly similar to our **RandomForest** method.

	Precision	Recall	F-Score
Ungrammaticality	72.8	49.1	58.6
Datelines	86.0	89.7	87.8
Redundancy	58.0	28.2	37.0

Table 8. The three best methods for detecting Ungrammaticality (*other ungrammatical form + incomplete sentence*), Datelines (*inclusion of datelines*) and Redundancy (*redundant information*)

The best methods for each violation, evaluated on *test*, can be seen in Table 6.

Along with this paper, the authors will publish a toolchain that can be used to annotate given documents with the best methods described in here.

We only looked at a part of the violation types in Friedrich et al. (2014). Methods for detecting the last clause-level violation (no semantic relatedness), as well as the entity- and discourse-level violations have yet to be developed:

For the pronoun violations (*pronoun with missing antecedent* and *pronoun with misleading antecendent*), one could use the output of a coreference resolution tool such as BART (Versley et al., 2008) or the Stanford coreference resolution system (Lee et al., 2011) to infer coreferences between entities in the summaries as well as the source documents and compare the originating coreference chains.

Finding *acronym without explanation* seems to have a straightforward two-step solution: For every sequence of all-uppercase letters, find out whether those are an acronym that is not in a list of well-known acronyms. If it is, look for the expanded form of the acronym near its first occurence.

It is to be seen whether this method works as well as the author believes.

Going after the mention (*first mention without explanation* and *subsequent mention with explanation*) and noun phrase violations (*definite noun phrase without reference to previous mention* and *indefinite*

noun phrase with reference to previous mention) would at least require a named entity recognition system, probably paired with at least some rudimentary form of semantic or discourse parsing.

no semantic relatedness also suggests an approach using semantic parsing, but usable results might even be achieved by some method based on the distance between content words in *WordNet* (Fellbaum, 2005).

Finally, the straightforward solution for detecting the *no discourse relation* violation seems to be parsing the two sentences *without* the discourse connective using a discourse parser, and then comparing the derived discourse relation with the relation commonly linked with the connective.

Future work could also include annotating a corpus after segmenting it, and using the adapted violation type system developed by us for that task.

And finally, since some of the characteristics of the violations our methods were tailored to are specific to the domain and corpus, it is yet to be seen how well our system performs on other corpora, other data sets and other domains. Evaluation on anything other than extractive multi-document summarization systems however would probably lead to disappointing results: The violation types for which we have detection methods should not occur at all in abstractive systems, and some of them will likely not or only rarely occur in single-document summaries, for example *redundant information* or some subtypes of *other ungrammaticality*.

References

Aberdeen, J., Burger, J., Day, D., Hirschman, L., Robinson, P., and Vilain, M. (1995). Mitre: description of the alembic system used for muc-6. In *Proceedings of the 6th conference on Message understanding*, pages 141–155. Association for Computational Linguistics.

Atwell, E. S. (1987). How to detect grammatical errors in a text without parsing it. In *Proceedings of the third conference on European chapter of the Association for Computational Linguistics*, pages 38–45. Association for Computational Linguistics.

Bender, E. M., Flickinger, D., Oepen, S., Walsh, A., and Baldwin, T. (2004). Arboretum: Using a precision grammar for grammar checking in call. In *InSTIL/ICALL Symposium 2004*.

Bird, S., Klein, E., and Loper, E. (2009). *Natural language processing with Python.* " O'Reilly Media, Inc.".

Breiman, L. (2001). Random forests. *Machine learning*, 45(1):5–32.

Chodorow, M. and Leacock, C. (2000). An unsupervised method for detecting grammatical errors. In *Proceedings of the 1st North American chapter of the Association for Computational Linguistics conference*, pages 140–147. Association for Computational Linguistics.

Christensen, J., Mausam, Soderland, S., and Etzioni, O. (2013). Towards coherent multi-document summarization. In *Proceedings of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies (NAACL 2013)*.

Cohen, W. W. (1996). Learning trees and rules with set-valued features.

Copestake, A. (2002). *Implementing typed feature structure grammars*, volume 110. CSLI publications Stanford.

Fellbaum, C. (2005). Wordnet and wordnets.

Finkel, J. R., Grenager, T., and Manning, C. (2005). Incorporating non-local information into information extraction systems by gibbs sampling. In *Proceedings of the 43rd Annual Meeting on Association for Computational Linguistics*, pages 363–370. Association for Computational Linguistics.

Flickinger, D. (2000). On building a more effcient grammar by exploiting types. *Natural Language Engineering*, 6(01):15–28.

Friedrich, A., Valeeva, M., and Palmer, A. (2014). LQVSumm: A corpus of linguistic quality violations in multi-document summarization.

Graff, D., Kong, J., Chen, K., and Maeda, K. (2003). English gigaword. *Linguistic Data Consortium, Philadelphia*.

Hall, M., Frank, E., Holmes, G., Pfahringer, B., Reutemann, P., and Witten, I. H. (2009). The weka data mining software: an update. *ACM SIGKDD explorations newsletter*, 11(1):10–18.

Hatzivassiloglou, V., Klavans, J. L., and Eskin, E. (1999). Detecting text similarity over short passages: Exploring linguistic feature combinations via machine learning. In *Proceedings of the 1999 joint sigdat conference on empirical methods in natural language processing and very large corpora*, pages 203–212. Citeseer.

Joachims, T. (2002). *Learning to Classify Text Using Support Vector Machines – Methods, Theory, and Algorithms*. Kluwer/Springer.

Lee, H., Peirsman, Y., Chang, A., Chambers, N., Surdeanu, M., and Jurafsky, D. (2011). Stanford's multi-pass sieve coreference resolution system at the conll-2011 shared task. In *Proceedings of the Fifteenth Conference on Computational Natural Language Learning: Shared Task*, pages 28–34. Association for Computational Linguistics.

Leech, G., Garside, R., and Atwell, E. (1983). The automatic grammatical tagging of the lob corpus. *ICAME news*, 7(1983):13–33.

Levenshtein, V. I. (1966). Binary codes capable of correcting deletions, insertions and reversals. In *Soviet physics doklady*, volume 10, page 707.

Levin, B. (1993). *English verb classes and alternations: A preliminary investigation*. University of Chicago press.

Lin, C.-Y. (2004). Rouge: A package for automatic evaluation of summaries. In *Text Summarization Branches Out: Proceedings of the ACL-04 Workshop*, pages 74–81.

Manning, C. D., Surdeanu, M., Bauer, J., Finkel, J., Bethard, S. J., and McClosky, D. (2014). The Stanford CoreNLP natural language processing toolkit. In *Proceedings of 52nd Annual Meeting of the Association for Computational Linguistics: System Demonstrations*, pages 55–60.

Maxwell, J. and Kaplan, R. (1996). An efficient parser for LFG. In *Proceedings of LFG*, volume 96, page 131.

Müller, C. and Strube, M. (2006). Multi-level annotation of linguistic data with mmax2. *Corpus technology and language pedagogy: New resources, new tools, new methods*, 3:197–214.

Nenkova, A., McKeown, K., et al. (2011). Foundations and trends® in information retrieval. *Foundations and Trends® in Information Retrieval*, 5(2-3):103–233.

Owczarzak, K. and Dang, H. T. (2011). Overview of the TAC 2011 summarization track: Guided task and AESOP task. In *Proceedings of the Text Analysis Conference (TAC 2011), Gaithersburg, Maryland, USA, November*.

Packart, W. (2011). ACE: the Answer Constraint Engine. http://sweaglesw.org/linguistics/ace/. Accessed: 2014-08-20.

Papineni, K., Roukos, S., Ward, T., and Zhu, W.-J. (2002). Bleu: a method for automatic evaluation of machine translation. In *Proceedings of the 40th annual meeting on association for computational linguistics*, pages 311–318. Association for Computational Linguistics.

Porter, M. F. (1980). An algorithm for suffix stripping. *Program: electronic library and information systems*, 14(3):130–137.

R Core Team (2013). *R: A Language and Environment for Statistical Computing*. R Foundation for Statistical Computing, Vienna, Austria.

Schneider, D. and McCoy, K. F. (1998). Recognizing syntactic errors in the writing of second language learners. In *Proceedings of the 17th international conference on Computational linguistics-Volume 2*, pages 1198–1204. Association for Computational Linguistics.

Stolcke, A. et al. (2002). SRILM-an extensible language modeling toolkit. In *INTERSPEECH*.

Sun, G., Liu, X., Cong, G., Zhou, M., Xiong, Z., Lee, J., and Lin, C.-Y. (2007). Detecting erroneous sentences using automatically mined sequential patterns. In *Annual Meeting-Association for Computational Linguistics*, volume 45, page 81. Citeseer.

Vadlapudi, R. and Katragadda, R. (2010). On automated evaluation of readability of summaries: capturing grammaticality, focus, structure and coherence. In *Proceedings of the NAACL HLT 2010 Student Research Workshop*, pages 7–12. Association for Computational Linguistics.

Versley, Y., Ponzetto, S. P., Poesio, M., Eidelman, V., Jern, A., Smith, J., Yang, X., and Moschitti, A. (2008). Bart: A modular toolkit for coreference resolution. In *Proceedings of the 46th Annual Meeting of the Association for Computational Linguistics on Human Language Technologies: Demo Session*, pages 9–12. Association for Computational Linguistics.

Wacholder, N. (1998). Simplex nps clustered by head: a method for identifying significant topics within a document. In *The Computational Treatment of Nominals: Proceedings of the Workshop*, pages 70–79.

Wagner, J., Foster, J., and van Genabith, J. (2007). A comparative evaluation of deep and shallow approaches to the automatic detection of common grammatical errors. Association for Computational Linguistics.

Wickham, H. (2009). *ggplot2: elegant graphics for data analysis*. Springer New York.