Bauhaus Universität Weimar
Faculty of Media
Media Systems

Using Language Models to Detect Errors in Second-Language Learner Writing

Bachelor Thesis

Nils Rethmeier Student number 40106

Born at Rostock, Germany, on June 12, 1983

1. Supervisor: Prof. Dr. Benno Maria Stein

Tutor: Martin Potthast

Date of submission: April 7, 2011

Declaration

Unless otherwise indicated in the text or references, this thesis is entirely the product of my own scholarly work.

Weimar, April 7, 2011

.....

Nils Rethmeier

Abstract

This thesis explores the use of different language modelling approaches to error detection in English Language Learner writing and investigates detection performance measures at different levels of detection granularity. We compare the error detection performance of word language models and a class-based extension to them at sentence and world level using language learner corpora and a corpus of automatically generated errors. Furthermore, we explore the issues that arise from ambiguities in both human and algorithmic error annotation and the impact they have on error detection results. We find that our class-based extensions to word language models improve error detection performance and that error detection quality at word level may require more advanced measures than the standard precision and recall as they seem to neither adequately assess human nor algorithm error detection quality. We conclude that the latter finding requires further investigation as the learner corpora we draw our results from are too small in size to produce fully conclusive results.

Contents

List of Figures

Chapter 1

Introduction

This thesis is about finding errors in second language (L2) writing using algorithmic error detection. Writing errors, such as wrong wording, flawed word order, misspelling, made-up words and faulty construction of tense, degrade the quality of a text and can make it hard, if not impossible, to understand the content.

Errors are easily made during writing, but usually we neither know that we made them nor where we made them. As a result, finding one's errors requires time, effort, and sometimes even money if professional help is being used. This is especially true when writing in a foreign language because, rather than merely translating individual words, one has to adapt to foreign idiomatic expressions.

To detect errors in second language writing, we examine existing error detection methods and focus on approaches that use language models, i.e. statistical models that can be used to determine how likely each expression contained in a presumably erroneous sentence is. Furthermore, we study evaluation measures commonly used as detection quality criteria and explore the impact of word class-based language models on detection quality. Finally, we compare the performance of algorithmic detection methods and a crowdsourcing approach that uses native speakers to detect L2 errors.

Chapter 2

Language Models

A language model uses a statistical representation of a language, called an *n-gram corpus*, to determine how likely it is for a portion of text (e.g. a phrase) to appear in a language [1]. First, this chapter describes how an n-gram corpus is built from a large collection of text: how n-grams are extracted from a text collection and then counted to be stored as a statistical representation of language. Second, it overviews how n-gram counts from the corpus can be used to determine if a given n-gram exists and how likely the n-gram is to appear in the language. Additionally, the chapter summarizes how language models are smoothed and how class-based language models work. The formulas and concepts described in this chapter are based on the book "Speech and Language Processing" by Daniel Jurafsky and James H. Martin [1], which provides one of the most concise and sound overviews of the subject matter. Since a large part of our language model implementations use Google's web n-gram collection released in 2006 [2], we also include a brief examination of their collection.

2.1 N-Gram Corpus Construction

This section overviews how an n-gram corpus is constructed by collecting n-grams frequency information from a text collection. It describes what an n-gram is, followed by an illustration of how n-grams are extracted from a large collection of written language and how n-gram occurrence counts are collected to form an n-gram corpus.

A *word n-gram* is a sequence of n words $w_1 w_2 \dots w_n$. The phrase "a sequence of words", for example, a 4-gram which is denoted as $w_1 w_2 \dots w_4$.

An n-gram corpus contains information about how often each distinct n-gram occurred in a given collection of written language (a text corpus). Figure 2.1 illustrates

the details of corpus construction using an example paragraph[1].

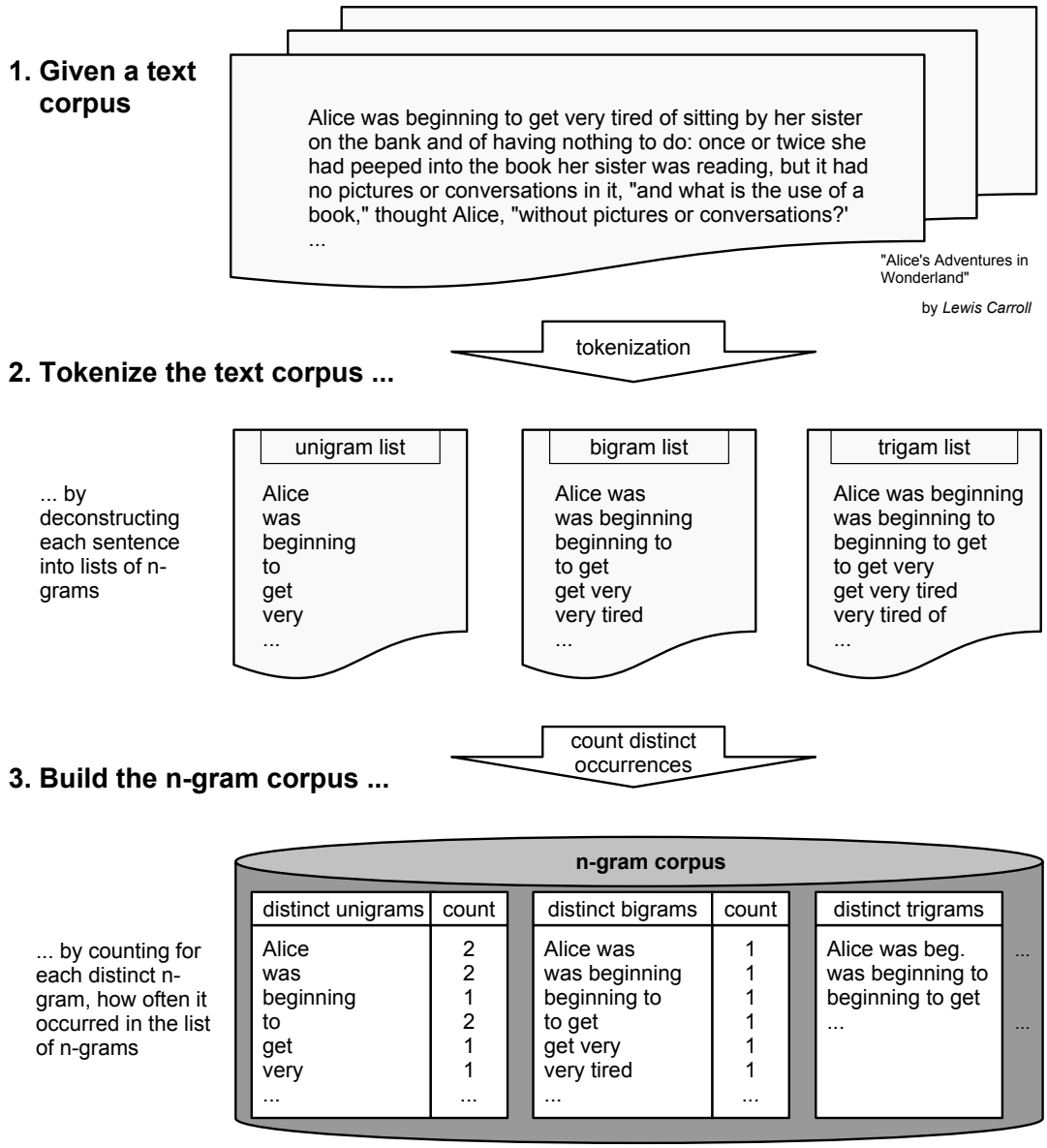

Figure 2.1: A text corpus **1.** is tokenized into n-grams **2.** of different length to obtain n-gram lists for uni- bi- trigrams and so forth. In a third step n-gram occurrences are counted to from the n-gram corpus **3.**

[1] The paragraph was taken form the book "Alice's Adventures in Wonderland" by Lewis Carroll [3].

2.2 Language Models

This section describes three approaches to how n-gram counts from n-gram corpora are used to model a language. The first approach uses *raw n-gram counts* to determine whether an n-gram exists within a language. The second approach determines how likely it is for an n-gram's last word to follow its preceding words. And finally, the third approach describes how likely a whole phrase (n-gram) appears in the modelled language.

N-gram Existence

The most basic approach to determining whether a certain phrase is a part of a language or not is to check if the phrase n-gram can be found in the n-gram corpus. For example, if we wanted to determine whether the bigrams "beginning to" and "starting to" exist within the language described by the n-gram corpus built in figure 2.1, we would simply retrieve their counts form the corpus. To get the counts a function count($w_1 w_2 \dots w_n$) is used, which takes an n-gram $w_1 w_2 \dots w_n$ as query and retrieves the associated count from the corpus. Using the counts for the two bigrams we would determine that the bigram "beginning to" exists within the corpus and therefore in the modelled language, whereas the bigram "starting to" does not (see figure 2.2).

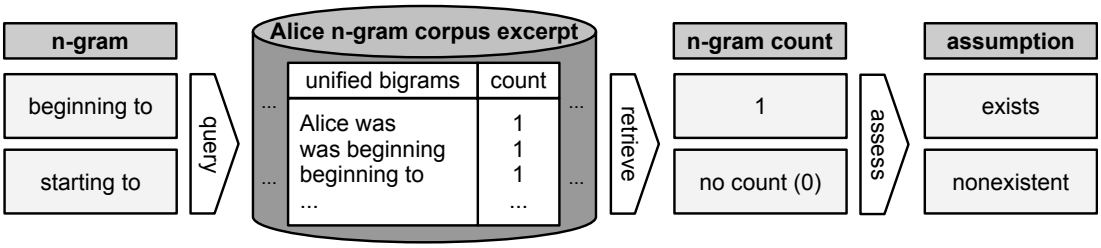

Figure 2.2: An n-gram can be queried against the n-gram corpus to retrieve an n-gram count and make an assumption whether or not an n-gram exits in the langauge.

Conditional probability of an n-gram

N-gram counts can also be used to determine the probability with which an n-gram's last word w_n follows the $n-1$ preceding words. Using the previously mentioned count function an n-gram's conditional probability $P(w_n|w_1 w_2 \dots w_{n-1})$ can be calculated from n-gram counts as follows [1].

$$P(w_n|w_1w_2\dots w_{n-1}) = \frac{\text{count}(w_1w_2\dots w_{n-1}w_n)}{\text{count}(w_1w_2\dots w_{n-1})} \tag{2.1}$$

In this equation $\text{count}(w_1w_2\dots w_n)$ describes the corpus count of the *whole* n-gram, including the last word w_n, and $\text{count}(w_1w_2\dots w_{n-1})$ describes the corpus count of the n-gram without its last word. For example, the conditional probability of the bigram "was beginning", $P(\text{"beginning"}|\text{"was"})$, would be calculated like so:

$$P(\text{"beginning"}|\text{"was"}) = \frac{\text{count}(\text{"was beginning"})}{\text{count}(\text{"was"})}$$

The probability is determined at $\frac{1}{2}$ because the unigram "was" appears two times in the corpus and the bigram "was beginning" appears exactly once as can be seen in figure 2.3.

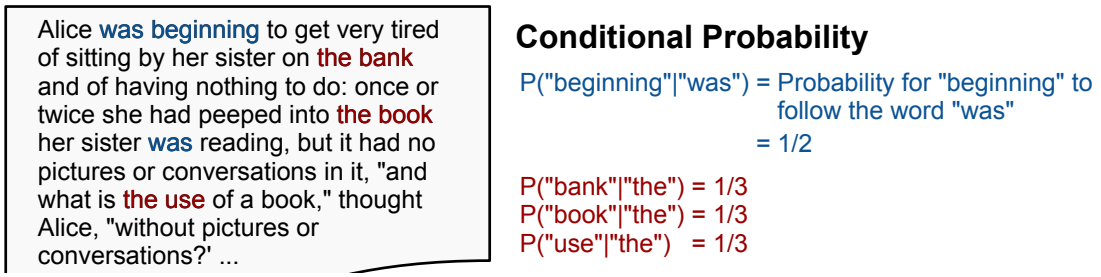

Figure 2.3: Conditional probabilities from n-gram counts.

Joint probability of an n-gram

To determine the likelihood of an *entire n-gram* (or phrase) an n-gram's joint probability can be calculated using either the total number of n-grams in the n-gram corpus or the chaining rule for conditional probabilities.

To calculate the joint probability using the total number T_n of all corpus n-grams equation 2.2 is used.

$$P(w_1w_2\dots w_n) = \frac{\text{Count}(w_1w_2\dots w_n)}{T_n}\ , \tag{2.2}$$

Here $w_1w_2\dots w_n$ is a word n-gram, and T_n is the number of corpus n-grams that have the same length as the word n-gram. This means that, for example, the joint probability of the 2-gram "was beginning" is calculated by dividing the 2-gram's corpus count (1)

by the total number of all corpus 2-grams T_2, which is 56 as seen in figure 2.4. According to equation 2.2 this would yield the following joint probability:

$$P(\text{“was beginning”}) = \frac{\text{Count}(\text{“was beginning”})}{T_2} = \frac{1}{56}$$

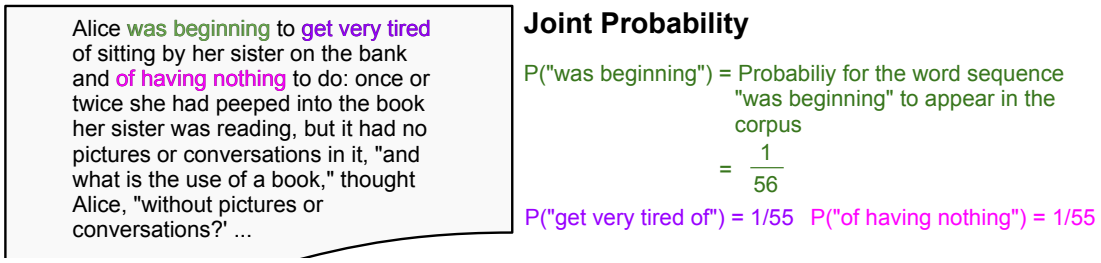

Figure 2.4: From Counts to Joint Probabilities:

Calculating the joint probability using the corpus T_n is usually very fast, because T_n can be recalculated for every n. However, recalculation is not always applicable as the following two cases will demonstrate. First, the n-gram can never be longer (have more words) then the longest n-grams in the corpus, because a corpus that contains 1, 2, and 3-grams can not directly be used to get the count of a 4-gram. And second, when an online source such as search engine queries are used get n-gram counts T_n is unknown.

In such cases, joint probabilities have to be calculated from conditional probabilities. Recalling that, in probability theory, a joint probability can be calculated from conditional probabilities (see equation 2.1) using the *chaining rule*, an n-gram's joint probability is calculated as follows [1]:

$$\begin{aligned} P(w_1 w_2 \dots w_n) &= P(w_1)P(w_2|w_1)P(w_3|w_1 w_2)\dots P(w_n|w_1 \dots w_{n-1}) \\ &= \prod_{i=1}^{n} P(w_i|w_1 \dots w_{i-1}) \end{aligned} \tag{2.3}$$

Since equation 2.3 requires $2{\cdot}n$ times the n-gram count lookups that equation 2.2 requires, it computes phrase probabilities $2 \cdot n$ times slower. Where n is the number of words in an n-gram.

In conclusion, there is a general trade-off between the two methods of calculation. While equation 2.2 needs information about the total number of corpus n-grams, but computes rapidly, equation 2.3 does not require such information, but is slower to compute for long n-grams. However, both equations produce a zero probability if an n-gram

does not exist in the n-gram corpus. Some n-grams are naturally missing from the n-gram corpus because language is flexible and evolves over time. To counteract this issue called *model sparseness*, two improvements exist, and are explained in the next section.

2.3 Model Sparseness

One of the main issues of language models is that the data they are built on is *sparse*, meaning that they do not contain n-grams to represent every portion of a language. As a result, n-grams that are not in the language model's n-gram corpus, are assigned zero probability.

To alleviate this limitation there are two improvements available.

- *More n-grams.* Scaling up the amount of data the language model uses, decreases the amount of a language it does not "know".
- *Language model smoothing.* This technique makes it possible to *estimate* language model probabilities for n-grams that are missing from the n-gram model.

Both techniques are more closely described in the following two sections.

Web-Scale Corpora

In late 2006 Google Research released a collection of n-grams that was taken from public webpages and scaled up to be a thousand times larger than existing corpora at that time, hence the expression *web-scale* [2]. The goal behind this effort was to provide the research community with a larger data set for their language modelling efforts, since Google's own research had proven that, "*there is no data like more data*". A hypothesis that has been reaffirmed by other researchers when they built their second language (L2) error detection on both, the offline Google n-gram collection, and counts attained online using Bing search queries [4]. However, larger n-gram collections can only reduce sparseness to some extent, not eliminate it.

Language Model Smoothing

Another approach to the problem of model sparseness and hence avoiding zero probabilities, is *language model smoothing*. This means shifting some probability mass from highly probable n-grams towards zero probability n-grams, thereby assigning n-grams that do not occur in the n-gram corpus with a low, but non-zero probability. N-gram

back-off is an efficient solution to language model smoothing [5]. A recently developed technique is described below.

Google Stupid Backoff [6].

This smoothing technique was devised by Google Research in 2007 and optimized for large scale models. It needs less calculations than other smoothing techniques while achieving similar performance, given a sufficiently large n-gram corpus [6]. While other language model smoothing techniques guarantee that the smoothed model is still a proper probability distribution, meaning that its n-gram probabilities sum up to exactly one, the Stupid Backoff technique sacrifices this property to significantly decrease calculation complexity, and thereby increase processing speed.

Stupid Backoff is used to determine an n-gram's backed off conditional probability $P_{SB}(w_n|w_1 \dots w_{n-1})$ from n-gram counts. To recall section 2.2, an n-gram's conditional probability determines with what probability a word w_n follows its $n-1$ preceding words $w_1 \dots w_{n-1}$. The preceding $n-1$ words are often referred to as w_n's history to describe the back off mechanism. In case a *zero probability* is attained, the technique backs off one step by shortening w_n's history and applying a back-off factor α. This is formalized by the following equation:

$$P_{SB}(w_n|w_1 \dots w_{n-1}) = \begin{cases} \dfrac{\text{count}(w_1 \dots w_{n-1} w_n)}{\text{count}(w_1 \dots w_{n-1})} & \text{if count}(w_1 \dots w_{n-1} w_n) > 0, \\ \alpha P_{SB}(w_n|w_2 \dots w_{n-1}) & \text{else, shorten history and repeat} \end{cases} \tag{2.4}$$

Here $\text{count}(w_1 \dots w_{n-1} w_n)$ is the corpus count of the whole n-gram, (i.e. the history $w_1 \dots w_{n-1}$ combined with the last word w_n.) The history is shortened by one word from the left, which is described by $w_2 \dots w_{n-1}$. In case no non-zero probability can be determined even for the backed off n-gram, the process is repeated recursively until a probability is found. Contrary to Google's original implementation of Stupid Backoff we stop recursion at bigram level because at unigram level it would not be possible to condition the occurrence of a word on the context of the preceding word. To give an example, Figure 2.5 illustrates how equation 2.4 can be used to calculate the conditional probability $P(\text{“book”}|\text{“Alice was reading a”})$.
The initial conditional probability $P_{SB}(\text{“book”}|\text{“Alice was reading a”})$ has a value of 1, but it still has to be corrected by multiplying the empirically determined back-off

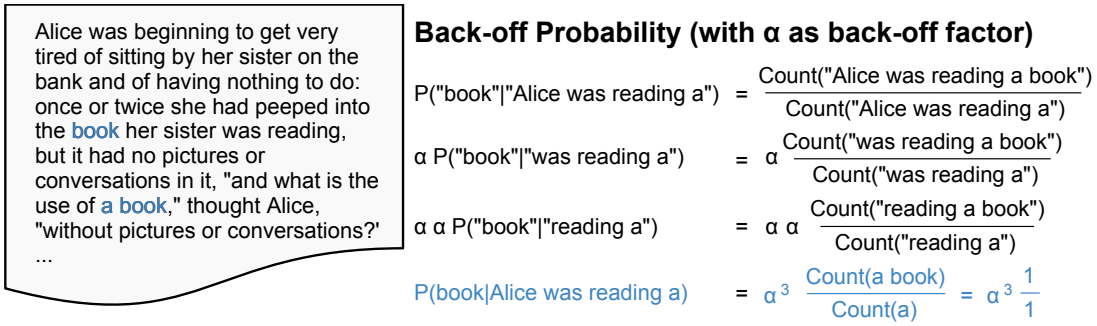

Figure 2.5: From Counts to Back-off Probability:

factor $\alpha = 0.4$ at each back-off level. Since back-off occurred three times the resulting probability $P_{SB}(\text{“book”}|\text{“Alice was reading a”})$ has a value of $1 \cdot (0.4)^3 = 0.064$.

In conclusion, smoothing improves a language model by making it possible to estimate probabilities for any given n-gram despite of model sparseness. Another extension would be to use multiple different language models and combine their information (see next section).

2.4 Class-based N-Gram Models

Another method to model a language is to use n-grams of word classes instead of word n-grams. Each word in a language belongs to a certain word class such as verb, noun, preposition and so on. Instead of tokenizing a sentence into its word n-grams it is also possible to tokenize the sentence into its word class n-grams by using parts-of-speech tagging which assigns every word in a sentence to a word class [1]. Hence, word class n-grams, like word n-grams, can be used to build an n-gram corpus from a collection of written language. Such a word class n-gram corpus can then be used by a language model to determine how likely it is for a certain word class n-gram to appear in a language. Which means that, class-based models are language models that use word class n-grams instead of word n-grams. Additionally, class n-grams are known to perform better at small model sizes than word n-gram models do, because there are significantly fewer distinct word classes than there are distinct words. This means class-based models exhibit less model sparseness than word language models. On the other hand, it also means that word language models built on large n-gram collections can determine n-gram probabilities more finely grained than class-based models which generalize words into

classes, as Samuelsson and Reichl point out [7].They use class-based language models to combine word n-gram probabilities with word class probabilities. Below, we summarize two approaches Samuelsson et al. propose [7]:

- *Interpolation between word and word class.* Using $w_1 w_2 \dots w_n$ to describe a word n-gram and $c_1 c_2 \dots c_n$ to describe the respective word class n-gram, word n-gram probabilities can be interpolated with word class n-gram probabilities using a weighting factor λ like so:

$$\begin{aligned} P_{IWC}(w_n|w_1 \dots w_{n-1}) &= \lambda_w \cdot P_w(w_n|w_1 \dots w_{n-1}) \\ &+ (1 - \lambda_w) \cdot P_c(c_n|c_1 \dots c_{n-1}) \end{aligned} \tag{2.5}$$

  In this equation $P_w(w_n|w_1 \dots w_{n-1})$ is the conditional probability of a word w_n as described by equation 2.1, λ_w is a standard interpolation weight used for the word probability, and $P_c(c_n|c_1 \dots c_{n-1})$ is the word class probability along with the remaining weight $1 - \lambda_w$. Interpolation has the advantage that, due to the interpolation weights, different models can be flexibly combined at any point as the different models are not dependent on each other's calculation. Another approach is to use a back-off mechanism.

- *Back-off from word to word class.* Here the model backs off from word n-grams to class n-grams if a word n-gram's probability falls below a predefined probability value. Thus, as Samuelsson et al. [7]. noted, this model has the advantage of only using class-based information when word n-grams failed, whereas interpolation mixes in information from class n-grams, even when this is is not necessary.

Chapter 3

Error Detection Algorithms

This chapter discusses the algorithms we use to detect second language (L2) writing errors. It starts with a standard definition for writing errors, overviews three algorithms, how we annotate errors, and what types of errors the algorithms are able to find.

3.1 Definition Writing Error

There is no single definition for errors in writing. Writing errors are usually defined by the category they belong to. The various categorizations found in the literature can be boiled down to the following four types..

- *Grammar errors.* A grammar error is a deviation from the grammar of a language. A grammar is defined as a set of rules that defines the "*correct formation of words (morphology) and sentences (syntax). A grammar is not concerned with either meaning (semantics) nor sound (phonology).* " [8]

- *Spelling errors.* A spelling error is a deviation from the "conventionally accepted way of forming words from letters" [9]. Fossati et al. [10] identified two types of spelling errors. The first type, *non-word errors*, are errors that lead to nonexistent words like "Wikypedia". The second type, *real-word errors*, are errors that result in an existing word, which is misplaced considering the context of the surrounding words. An example of a real-word error is the phrase "The companies expenditures increased...", that should correctly spell "The company's..." instead.

- *Semantic errors.* Finding errors in the meaning (semantics) of a text requires an understanding of its subject matter. Therefore, provided that a subject matter is

not understood, semantic errors can not be found. For example, if an anthropologist reads a scientific article about quantum physics he or she might not be able to fully grasp the article's content, much less find errors in its semantics.

- *Style.* Unlike the aforementioned error types, style focuses on making the content of a text more easily accessible for the reader. According to the Plain Language campaign [11] good style "avoids obscurity, inflated vocabulary and convoluted sentence construction." Thus, style errors are errors that make a text harder to understand. However, whether or not a certain style is appropriate depends on the domain and genre of the writing. Inflated vocabulary and convoluted sentence construction might be deemed acceptable style for philosophical works, but in a scientific publication of biology or physics they have no place.

3.2 Error detection methods

To detect the errors in a text we tokenize the text into n-grams and categorize each n-gram as correct or erroneous using n-gram occurrence counts and n-gram probabilities provided by our language models. We categorize an n-gram as erroneous if its probability lies below an empirically determined probability threshold τ. If a model uses occurrence counts a count threshold is used instead. To define the value of τ a language model is trained on a large collection of errors so that categorization into correct and erroneous n-grams is as accurate as possible. As the threshold globally divides the modelled language into correct and incorrect language, the collection of errors used for the training of τ has to be large enough to be statistically representative for the whole language.

We use the following three methods to detect erroneous n-grams which can then be joined to form error annotations for text.

N-Gram Existence

Our most basic method detects errors using occurrence counts from n-gram corpora and a count threshold to determine whether an n-gram exists in a language or not. First we tokenize a sentence into its word n-grams and store their respective positions in the text. We then look up each n-gram's count in the n-gram corpus and finally, use n-gram positions and counts to determine error positions. Figure 3.1 illustrates how this works using a snapshot of the Google n-gram corpus [2].

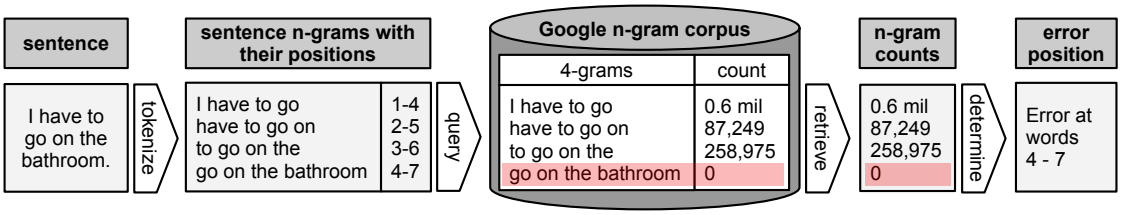

Figure 3.1: Error positions are determined through examination of each n-gram's occurrence count and text position.

N-Gram Backoff Probabilities

Our second approach detects erroneous n-grams using a probability threshold τ along with n-gram Stupid Backoff probabilities calculated from n-gram counts (see section 2.4) [6] . As before, the text is first tokenized into n-grams and their respective positions in the text so that each n-gram's probability can be calculated using n-gram count lookups on the n-gram corpus. Once a text's erroneous n-grams have been determined using τ their respective positions can be used to obtain the final error positions within the text. Figure 3.2 illustrates in how far the process differs from the existence method.

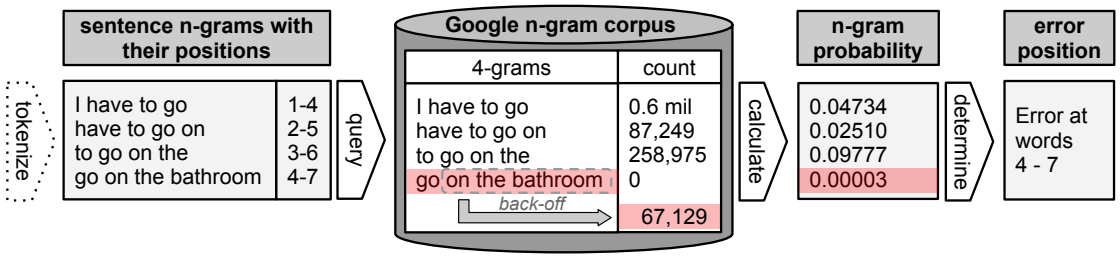

Figure 3.2: Using the counts from the Google corpus we calculate a Stupid Backoff probability for each 4-gram. Since the 4-gram "go on the bathroom" does not exist among the corpus 4-grams, the algorithm backs off to the 3-gram "on the bathroom" to calculate a probability. Since the resulting 3-gram probability of 0.00003 is smaller than the predetermined τ, an error is detected at word positions $4 - 7$.

Class-Based N-Gram Probabilities

In the third approach we combine language models of words with language models of word classes to detect errors. At first, a given sentence is tokenized into its word n-grams, word class n-grams and their corresponding positions within a sentence. Then,

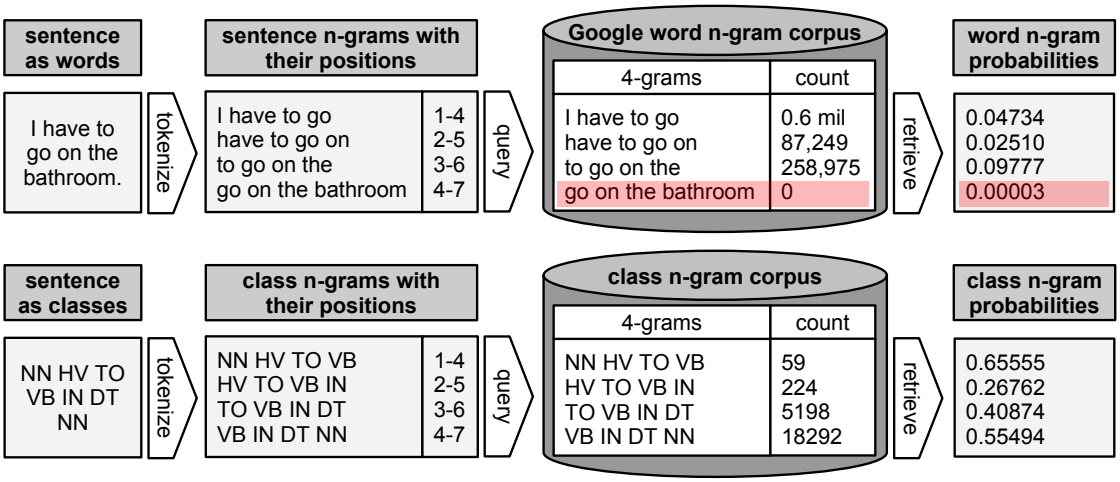

Figure 3.3: Process for retrieving erroneous word and word-class n-grams from a sentence. Both word and word class n-gram probabilities are obtained using Stupid Backoff.

word n-grams are queried against a word n-gram corpus to determine erroneous word n-grams. Additionally, word-class n-grams are queried on a word-class corpus to determine erroneous word class n-grams (see Figure 3.3).

Erroneous n-grams of both words and classes are combined either using one of the methods described in section 2.4, or by normalizing word n-gram probabilities with class n-gram probabilities. Given P_w as word probability and P_c as word class probability we normalize words with classes as follows:

$$P_N(w_n|w_1 \dots w_{n-1}) \quad = \quad P_w(w_n|w_1 \dots w_{n-1}) \cdot P_c(c_n|c_1 \dots w_{n-1}) \tag{3.1}$$

Here, both P_w and P_c are independently calculated, each using Google's back-off method described on page 11. P_w and P_c may be of different back-off levels, which means P_w might be attained on a word bigram while P_c resulted from a class 4-gram. An example of how this method can be used to detect erroneous n-grams is illustrated in Figure 3.3. While the Stupid Backoff probability for the word 4-gram "go on the bathroom" is very low the probability of the respective word-class 4-gram "VB IN DT NN" is very high. However, combing both probabilities using the normalization method we just describe results in a probability of 0.000016 which means that the n-grams at position $4-7$ are still classified as erroneous.

3.3 Algorithmic Error Annotation

We demonstrated three methods that classify the n-grams of a sentence as correct and erroneous, but the erroneous n-grams still need to be assembled to form *error annotations* that represent the errors detected within a sentence. The Figure 3.4 demonstrates how error annotations are drawn from a set of erroneous n-grams and their positions. While other methods to create error annotations may come to mind we choose a merging approach because we found it to be the most reliable way of creating algorithmic error annotation. A more detailed review of other annotation methods we tested is given in the next chapter (section 4.2).

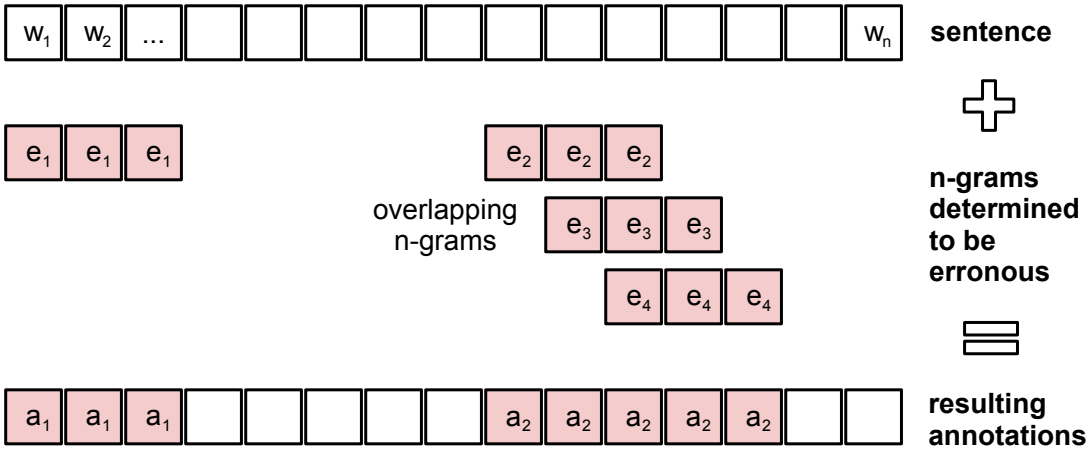

Figure 3.4: Using one of the previously mentioned error detection algorithms, erroneous n-grams e_i are now assembled to form error annotations a_i. For example, the n-grams of e_2, e_3 and e_4 have overlapping positions within the sentence. Hence, we merge them to form one single annotation a_2.

3.4 Error Detection Prospects

While the beginning of this chapter describes types of writing errors, we now summarize what types of errors are expected to be found using language models. Numerical examples are based on Google Stupid Backoff probabilities and n-gram counts [6] obtained using the language models we built on the Google n-gram corpus [2].

- *Spelling errors.* Misspelled words, or *non-word errors*, are detected as writing

errors by language models given that the misspelling does not occur in the models n-gram corpus or at least very rarely. For example, the phrase "I hav a house" does not exist in the Google n-gram corpus. The correct spelling "I have a house" on the other hand appears 16,078 times.

Real-word errors can be detected using language models as well. For example, the real-word error "rather then" has a probability of 0.004, whereas the correct form "rather than" has a probability of 0.52. Hence, "rather then" is detected as an error.

- *Grammar errors.* In linguistics the grammar of a language is a set of rules that defines "how sentences are formed from words to convey meaning" [12]. As Leacock et al. observe, "*grammar does not describe the conventional usage of a language, such as what preposition is best used in a given context, or whether to use the article 'a', 'the' or non article at all.*" [13]. Leacock et al. conclude that *usage errors* are the most common type of error found in second language (L2) writing and categorize them as a type of grammar error. In this thesis we apply language models as a statistical presentation of how language is conventionally used to find errors in learner writing.

- *Semantic errors.* The algorithms presented in this work do not understand meaning. As such, they can neither decide whether a text makes sense, nor where logical errors are located within it.

- *Style errors.* Style errors that concern sentence structure, e.g. overlong sentences or overuse of brackets are not detected by the algorithms. Style errors that result from using uncommon words and expressions however, are detected using language models.

In conclusion, using language models and thresholds to classify the n-grams of a text into the binary categories correct and erroneous allows us to detect and annotate the errors in a sentence. It does not, however, allow us to precisely determine what type of error was found; even though such information might be considered useful.

Chapter 4

Evaluation Framework

This chapter describes the evaluation corpora as well as the performance measures used to evaluate algorithmic error detection quality. The evaluation data are collections of sentences with reference error annotations that we compare with our algorithmically determined error annotations to measure detection quality. We overview three types of detection measures and how algorithmic and human error annotations alike are influenced by ambiguities in a sentence's semantics and grammar.

4.1 Evaluation Data

To evaluate our algorithms we first need a gold standard which is a set of texts with reference error annotations. We then feed these text to our algorithmic error detection, and compare the output to the reference error annotations in order to determine how well the algorithms extract errors. In this section we summarize two types of corpora that contain error annotated writing: learner corpora and an artificially constructed error corpus. The former are collections of language learner writing that have been error annotated by linguists. The latter is a corpus of well-formed English texts in which errors have been introduced artificially [14].

Learner corpora.

Learner Corpora are collections of second language learner writing. They are created to analyze language usage of learners so that linguists may study how certain second language (L2) error types are related to a learner's first language (L1) knowledge, or what error types are common for learners of the same L1 background [15]. For example, Rozovskaya et al. noted that: "Chinese learners struggle with article errors because the

Corpus Name	**Corpus Size**	**Annotated**	**L2**	**Available**
CLC Cambridge LC	30M words	6M words [17]	en	no
CLEC Chinese LEC	1M words	1M words [18]	en	yes
CzeSL Czech as a second language (*currently developed*)	TBA	TBA [19]	cz	not yet
EnglishTLC E Taiwanese LC	2M	partially [20]	en	yes
FALKO German learner corpus	36k	fully [21]	ger	yes
FRIDA French Interlanguage Database	450k	300k words [22]	fr	unclear
LLC Longman LC	10M	fully	en	no
MELD Montclair Electronic Language Database	18 essays	18 essays	en	yes
NICT Japanese LC	2M	no number	en	yes
TLCE Taiwanese LCE	710k words	fully [23]	en	no
TLEC Tswana LEC	200k	fully [24]	en	academic
123 Mass Noun Sentences	123 sentences	fully	en	yes
Fehlerprojekt Uni Augsburg	7000 errors	7000 errors	ger	yes
LC = Learner Corpus	*E = English*			

Table 4.1: List of learner corpora that contain error annotations.

Chinese language does not use articles. Speakers of languages that use articles, such as French or German, make few errors involving articles but struggle with punctuation [16]."

We use errors in learner corpora that were professionally annotated by linguists as a reference collection (gold standard) of writing errors, so that we may compare algorithmic error annotations against them. As part of our research we compiled a list of learner corpora that contain error annotations. We identified thirteen learner corpora of four different languages. They are listed in Table 4.1.

The errors in learner corpora are annotated using so called *error tag sets*, which map each learner error onto a predefined linguistic category. There is no standardized tagging system for learner corpora and each tag set focuses on different linguistic aspects. Although a general categorization of tag sets can be made, as found by Díaz et al. [25], the same learner error might be assigned to spelling errors in one tag set and to grammar errors in another. Thus, comparing evaluation results drawn from different learner corpora and error tag sets is rendered difficult.

Artificial Error Corpora.

Artificial error corpora are the second type of corpora we use for evaluation. They are built through insertion of writing errors into a corpus of well-formed language. Wagner et al. [14] generated such an artificial error corpus from the British National Corpus (BNC) [26] by "*inserting, deleting or replacing words within the* BNC". They find that a "comparative evaluation of existing error detection approaches has been hampered by a lack of large and commonly used evaluation error corpora"[14]. To address this problem they built an artificial error corpus that contains four different types of grammatical errors which were previously identified to be the most frequently occurring errors. The four error types are real-word spelling errors, agreement errors, missing word errors and extra word errors. These results are backed by findings of Nicholls et al. "who obtained a similar error categorization through examination of learner corpora" [14, 27].

4.2 Ambiguity of Error Annotation

Unlike simple non-word (spelling) errors, most types of writing errors are ambiguous by nature. If the task is to find the errors contained in a sentence both humans and algorithmic detection methods struggle with ambiguities in grammar and semantics, which in turn cause the task of error annotation to be ambiguous.

Semantic ambiguity

The meaning of a sentence, its semantics, are not always clear. The sentence "The boy carried the girl with the flower." for example, can have two meanings. It either states that the girl had a flower, e.g. in her hair, or that the boy used a flower to carry the girl. While the author knows his intended meaning, a reader does not. Similarly, an annotator who has to find writing errors in a text can only guess the intended meaning,

which means errors are ambiguous in relation to how the annotator understood the text. To give an example we examine a sentence taken from the 123 Mass Noun corpus that was assembled by Brockett et al. [28]. The sentence was produced by a Chinese learner. For the example we assume that the author's original (untranslated) intention was to say: "These insights are extremely useful." But instead, he produced the sentence: "These knowledge are extremely useful"[1], which is incorrect. It is clear that the translation he chose for "insights" is a wrong usage of the word knowledge. To find the error, we gave the sentence to three native speakers, recruited using a crowdsourcing platform called Amazon Mechanical Turk[2], and asked them to correct the sentence. Table 4.2 lists the three corrections they produced.

Original sentence	**Corrected sentence**
These knowledge_ are extremely useful_	(a) **This** knowledge **is beneficial.**
	(b) These knowledges are extremely useful**.**
	(c) **This** knowledge **is** extremely useful

Table 4.2: The first correction (a) is valid given the limited information and slightly changed the meaning of the sentence so that the sentence sounds less odd. The second annotator (b), did not find the important error but added the missing full stop. The third annotator (c), corrected the obvious grammatical error, but forgot to add the missing full stop.

Though two valid corrections were produced, none of the native speakers proposed to use insights instead of knowledge, because, the author made two errors at the same time. Not only did he choose the wrong translation for insights (semantic error), but also use the singular instead of the plural (grammar error); both of which are common errors in learner writing. Though the grammar error was easily found, the underlying semantic error was only noticed by annotator (a) who attempted to adjust the word "useful" to make the sentence sound less odd.

[1] The author also forgot the full stop after "useful".
[2] Found at https://www.mturk.com.

Grammatical ambiguity

For algorithmic error annotation, additional issues arise from *grammatical ambiguities* like the full stop, which is ambiguous because it can either indicate the end of a sentence or an abbreviation [29]. Birn et al. described the problem with ambiguity as follows: "Grammatical errors may disturb disambiguation, which in turn disturbs detection of grammatical errors" [30]. Their conclusion is backed by findings of Tetreault et al. who noticed that "annotators only agree 75% of the time on the same preposition when given a choice to pick one" [31], even though prepositions are seemingly unambiguous. Grammatical ambiguities are also a problem in part-of-speech (POS) tagging, where so called POS taggers determine a word's class as verb, noun, adjective an so forth. POS tagging becomes ambiguous when a sentence contains grammatical errors, which in turn negatively affects error detection performance as investigations of Díaz and Oronoz revealed [32, 33].

Ambiguity in algorithmic error annotation

To create error annotations we use n-grams that were classified as erroneous using our algorithms. In the case of trigrams this means that a one word error is annotated along with the two words that surround it, which results in a rather fuzzy annotation. Though we made attempts at making our error annotation more precise, so that a one-word-error is detected as such, we refrained from doing so for two reasons:

1. Given the complexity of the problem this type of disambiguation is beyond the scope of this work. However, we explain two of the methods we did test to minimize error annotation length and why they did not work.

2. Looking at the semantic and grammatical ambiguities in writing we decided not to disambiguate but instead preserve the contained ambiguities.

To minimize the length of error annotations and thereby disambiguate it we tested two methods. We illustrate them in Figure 4.1 and explain why both methods do not work as desired. The examples are based on observations made on learner sentences.

Our first attempt (a) was to assume that if adjacent erroneous n-grams overlap each other their intersection must clearly be an error. The second attempt (b) was built on the assumption that a low conditional probability for an n-gram indicates that the last word in the n-gram causes the error. We were quite surprised to observe that assumption (b)

is not generally true, as the error annotation was more frequently caused by a preceding word than by the last one.

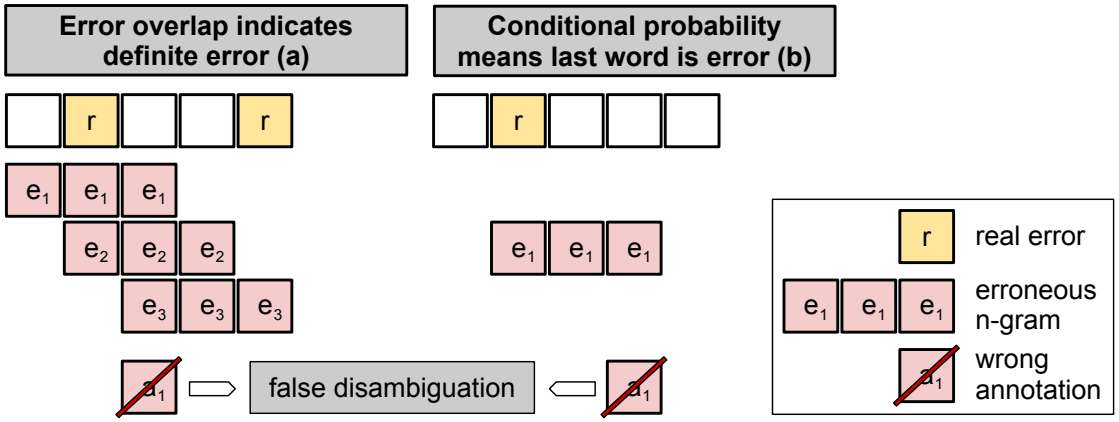

Figure 4.1: Shown are two different methods to disambiguate algorithmic error annotation. Method (a) uses n-gram overlap and method (b) only uses an n-gram's last word to construct error annotations. Both methods lead to false disambiguations.

Both methods can not generally be expected to work for the following two reasons. First, an erroneous n-gram does not specify *how many of its words are wrong*. Second, an erroneous n-gram does not specify *the position of the erroneous word(s)*, which means that low conditional probability for an n-gram can occur although the error lies with the n-gram's first word rather than with its last word, as might be expected. Experiments we conducted using the two disambiguation methods during early development of our error detection algorithms showed a consistent decrease in error detection performance when using either of the two methods. After additional manual examination of the error annotations we produced on learner errors using the two methods we decided to discard both annotation disambiguation methods as unsuitable heuristics.

Another reason why we do not disambiguate error annotation can be explained by a closer look at the example given in Table 4.2. The human annotators we hired using crowdsourcing produced three different error annotations, or more specifically corrections, that are located throughout the whole sentence because of the mentioned ambiguities. This suggests that a more precise error annotation may not be useful since it would limit the range of possible corrections to a solution that does not necessarily fit the author's original intention.

As a result of these observations we decided to use a type of error annotation that

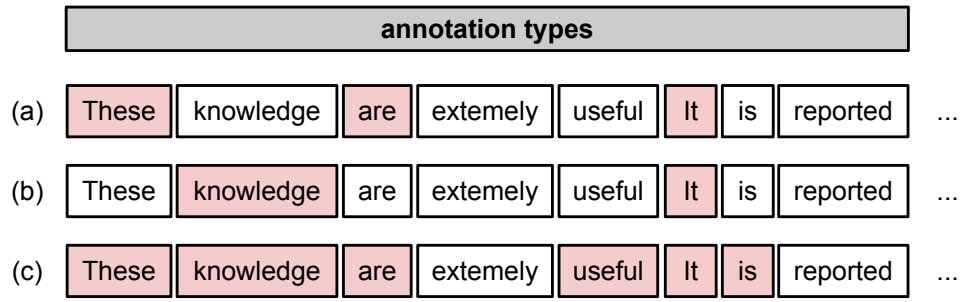

Figure 4.2: Three different ways to annotate errors. Version (a) would suggest a grammatical error to the sentence author, because the plural does not fit the noun. Annotation (b) suggests that the wrong word might have been chosen; "insights" instead of "knowledge" would fix this error. The annotation we use, (c), does not disambiguate and simply states that there is an error involving the first three words, thereby covering both annotation (a) and (b).

does not dissolve ambiguity but preserves it. In doing so we fall in line with observations made by Díaz et al. who concluded that "a disambiguation of grammatical errors is not always useful" [32]. In Figure 4.2 we illustrate how we annotate errors (c) and give examples how other annotation types may dissolve ambiguity. Since the example sentence is missing a full stop we included the beginning of the sentence that follows.

Annotation type (c) is the direct result of using our detection algorithms on the two sentences. We detected two erroneous trigrams which are used to annotate two errors. Both the errors from (a) and (b) are annotated, though it can be noted that our algorithmic error annotation is fuzzy when compared to the results produced by human annotators in Table 4.2. However, to use the human annotations, a language learner is still required to manually pick the best solution, which in turn, requires him to assemble the best solution and discard wrong results. The same ability is needed when using algorithmic error detection. While a fuzzy annotation of writing errors might severely hit measured performance of an error detection system, it says little about how useful the annotations are for a language learner. Studies conducted by Microsoft Research on second language learner data strongly suggest that learners are able to infer where the error they made is located, even though the system did not mark the exact location [34].

4.3 Detection Performance Measures

To quantify how well writing errors are detected in the previously mentioned corpora, performance measures are needed. This section overviews the measures precision and recall, and how they are applied at three different levels of annotation granularity. We describe one commonly used sentence level measure as well as word and character level detection measures that we incorporate to determine how well individual errors within a sentence are annotated rather than just finding out whether or not a whole sentence is ungrammatical.

Precision and recall in error detection.

A common way to assess the performance of an information retrieval system is to calculate its precision and recall on a test set. For algorithmic writing error detection precision and recall are determined on sets of erroneous and correct sentences that are taken from writing error corpora. The measures are defined as follows [13]: precision measures the percentage of all algorithmic error annotations that are correct, recall measures the percentage of the reference errors that are also annotated using the algorithms. We illustrate both measures in the following Figure (4.3).

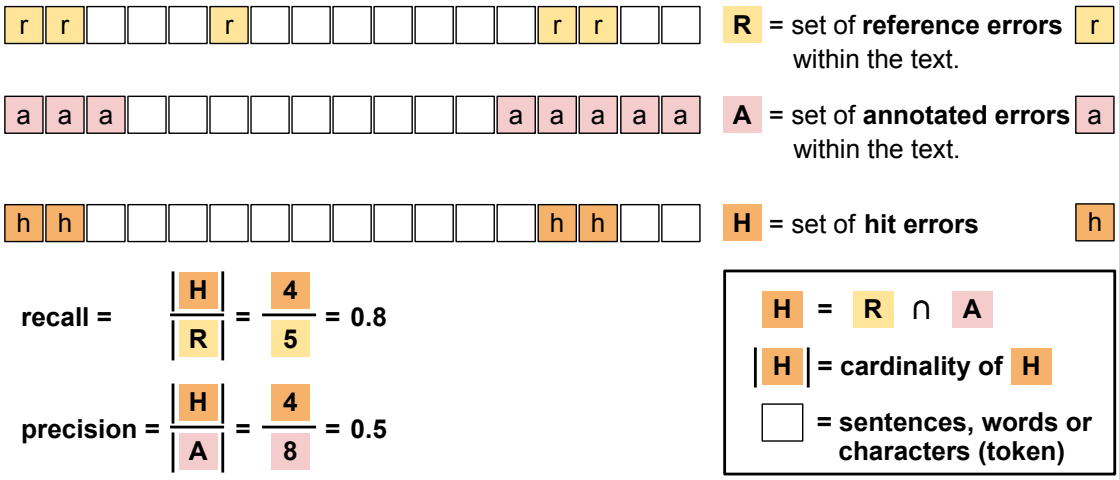

Figure 4.3: An illustration of precision and recall in algorithmic error detection.

Precision and recall can be calculated at three different levels of granularity: sentences, words and characters. We describe how they are calculated on each level using a set of three reference sentences. While the sentences remain the same, the tokens for

sentences, words and characters change accordingly at each level (figures 4.4, 4.5 and 4.6). We represent individual tokens by colored boxes as specific sentences, words and symbols would only hinder understanding.

Sentence level.

Determining detection performance at sentence level is commonly used to evaluate error detection systems [1, 14, 28]. For this measure, every square in Figure 4.3 represents a sentence that is classified as either grammatical (correct) or ungrammatical (contains an error somewhere). A sentence is a hit, i.e., a correctly found error, if both algorithmic and reference annotation classify it as ungrammatical. Therefore, whether or not an error's location within the sentence was correctly determined is not considered by this measure. Using the measure on the three sentences example in Figure 4.4 yields a recall of 1.0 and a precision of 1.0. This suggests, that all writing errors were correctly determined.

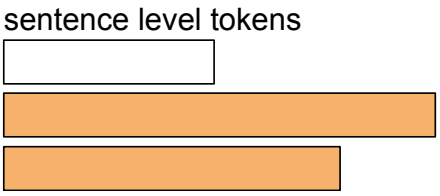

Figure 4.4: Each box represents a sentence. Boxes are colored as illustrated by Figure 4.3.

Word level.

We incorporate word level evaluation to measure how well our algorithms are able to find individual errors within a sentence. At this level, every square in figure 4.3 represents a word and every row of squares a sentence. For this measure, each word in a sentence is classified into the binary classes of correct and erroneous. A word is a hit if both algorithmic and reference annotation classify it as erroneous.

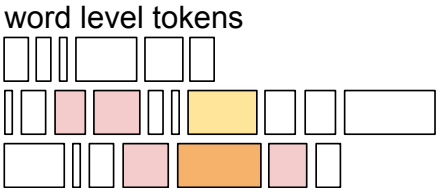

Figure 4.5: Each box represents a word and each row a sentence. Boxes are colored as introduced by Figure 4.3.

Measuring performance at word level (Figure 4.5), yields a recall and precision of 0.0 for the second sentence, because there was no word-hit. For the third sentence a recall of 1.0 and a precision of .33 is determined. Using the arithmetic mean of all precision- and recall values yields an overall performance of 0.5 for recall and 0.16 for precision. This stands in harsh contrast to the perfect scores attained by the sentence level measure.

Character level.

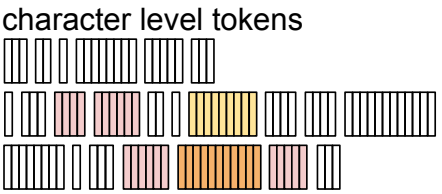

Figure 4.6: Each box represents a character and each row a sentence.

Measuring detection performance at character level was our first approach to improve upon the sentence level measure. The character level measure, like the word level measure, enables us to measure detection performance for individual error word annotations. However, it does not entirely fit our purpose since detection of wrong letters within a word is not required. For our purpose, one of this measure's major drawbacks is that it weights a word's influence on precision and recall through the word's character count. As a result, long words have a major impact on performance evaluation while short words have almost none.

An examination of sentence three in figure 4.6 demonstrates this property. Sentence three yields a recall of 1.0, which makes sense since the reference error was detected (orange hit). It also yields a precision of 0.5 which seems too high regarding that only one out of three words was a hit.

Though we did not find explicit mention of the word and character level error detection measures in the related literature or scientific publications, we believe that the word level measure is a straightforward concept for measuring detection quality. However, we noticed that our measures are based on similar ideas as the BLEU measure, which was developed by IBM to measure word level errors in machine translation [35]. Like the BLEU measure, our measures can be used with multiple reference sentences in order to alleviate error ambiguity by picking the reference sentence that best fits with the algorithmic solution [35].

Chapter 5

Experiment Setup and Evaluation

This chapter describes the corpora we built our language models on (the development set), how we built them, what corpora we used to train an optimum threshold for error detection (training set), and finally the corpora we tested our trained models on (test corpora). We discuss error detection results produced on the corpora as well as the conclusions we draw from them.

5.1 Development Set

The *development set* of a language model is the n-gram corpus the model uses to determine the probability of an n-gram. For our language models we use three different n-gram corpora: a word n-gram corpus and two class n-gram corpora.

Word n-grams

Word n-grams from the Google n-gram corpus [2] are used to classify a sentence's word n-grams as either correct or erroneous. We built a Stupid Backoff [6] smoothed language model on Google's collection of 1- to 5-grams. Their collection is built from $95,119,665,584$ sentences and contains $1,024,908,267,229$ tokens. An excerpt of the corpus shows trigrams with their corresponding frequency counts (see Table 5.1).

Parts-of-speech n-grams

The part-of-speech n-gram corpus was built by extracting part-of-speech (POS) n-grams from the British National Corpus (BNC) [26]. We use this POS n-gram corpus to detect errors in a sentence by classifying the sentence's POS n-grams as grammatical

Word trigram	**Frequency count**
serve as the incubator	99
serve as the independent	794
serve as the index	223

Table 5.1: An excerpt of 4-grams contained in the Google n-gram corpus.

or ungrammatical. The corpus was built in three steps. First, we turned each sentence in the BNC corpus into a sentence of parts of speech. Second, we deconstructed the sentences into their POS 1- to 5-grams, and third, counted how often each distinct POS n-gram occurred in the corpus. This gave us a corpus of the form presented in Table 5.2.

POS trigram	**Frequency count**
JJ HVZ BEN	53
NNS VBN IN	6378
NP HVZ HVD	2

Table 5.2: Three trigrams from our part-of-speech corpus. JJ = adjective, HVZ = has, BEN = been, NNS = noun as plural, VBN = verb in past participle, IN = preposition or subordinating conjunction, NP proper noun as singular, HVD = had.

It contains $7,736,709$ distinct n-grams and was built from $94,004,218$ word tokens. Though this corpus is considerably smaller than the Google corpus, part-of-speech corpora are less sparse than word corpora, which means smaller sizes suffice, as discussed in section 2.4.

Prefix class n-grams

The prefix n-gram corpus was derived from a Wikipedia word n-gram corpus. We built this corpus to classify n-grams by the first letter of their words. For example, the unigram, or word, "language" belongs to the prefix class 'l' because it starts with the

letter 'l'. Likewise the trigram "language is a" belongs to the prefix class "l i a".

We built the prefix corpus using a publicly available n-gram collection that was assembled from plain text previously extracted from the Wikipedia.[1] We then took this corpus and turned all its word 1- to 5-grams into a corpus of prefixes. Table 5.3 shows an excerpt of trigrams from the prefix n-gram corpus we built.

Prefix trigram	**Frequency count**
i N Y	78243
I n y	243
I n Y	87

Table 5.3: An excerpt of prefix trigrams used for our experiment. These prefixes have a length k of one letter per word (k=1).

The prefix corpus contains $146,654,396$ distinct prefix n-grams and was built from $1,050,464,624$ word tokens. Though we only use prefixes as classes for our experiments, the approach can be generalized to use other affixes such as suffixes and infixes. Along with the prefix corpus, we built respective corpora for suffixes and infixes. These corpora are built to generalize the concept of classification by first letter into classification by first k letters, where $k = \{1, \ldots, 5\}$ letters. The whole affix corpus contains $26,691,296,081$ affix n-grams of which we use $146,654,396$ $k = 1$ prefix n-grams for our experiments.

5.2 Training Set

The *training set* of a language model is the data used to optimize the model's performance for a given task. In language-model based writing error detection, corpora of reference errors are used as training set's to optimize the *classification threshold* τ by which n-grams are classified as either correct or erroneous. As mentioned in section 3.2, an n-gram with a probability below the classification threshold τ is classified as erroneous, an n-gram with a probability higher than τ is classified as correct.

[1] The corpus is available at http://nlp.cs.nyu.edu/wikipedia-data/ under the GNU Free Documentation License.

Using 80% of the sentences contained in the corpus created by Wagner et al. [14] as training set, we were able to train τ so that error detection performance, as described in section 4.3, is maximized. This means that, for different thresholds, we calculated error detection precision and recall on the corpus to determine at what threshold τ detection precision is highest. The τ was not trained separately for each of the five error types contained in the artificial BNC, but for all error types at once, since a language model merely classifies n-grams, without any information about the type of error that was detected.

To obtain optimum thresholds τ for our language models of word probabilities, word counts, part-of-speech and prefix probabilities as well as the different class combination models for word and class probabilities we:

1. Extracted reference error annotations from the BNC corpus of artificially generated errors using a Myers Difference [36] on corrected and erroneous versions of the same sentence.

2. Broke up each erroneous sentence into its word, part-of-speech and prefix n-grams to determine the respective probabilities.

3. Tested 100 different τ's per language model to determine τ so that classification of n-grams into correct and erroneous is as precise as possible.

We trained the model thresholds using $3,527,415$ pairs of correct and erroneous sentences of which some had to be discarded because the algorithms involved did not always work as expected. During training, we encountered the following problems:

- Tokenization is a highly difficult problem in the creation of an n-gram corpus. Within Google's word n-gram corpus for example, we found that the apostrophe (') is not consistently tokenized. Most of the tokenization found in the collection removes the apostrophe in contractions with not, e.g. "don't" becomes "dont", but does not in contractions with would, e.g "I'd". While our word n-gram tokenization is adjusted to the Google collection, the tokenizer we used for part-of-speech sentence tokenization is not. As a result, word n-grams with apostrophes removed are algorithmically not matching to their underlying part-of-speech n-grams which is why we skipped sentences containing apostrophes during training. We considered including a workaround for this problem but did not for two reasons: first, the apostrophe like the full stop is ambiguous in its grammatical meaning which would

require a more advanced tokenization, and second, by convention, contractions are considered a style error in written language and by common error detection systems.

- We skipped sentences for which the reference implementation we used to extract part-of-speech n-grams does not consistently handle number tokenization (QTag [37]). While short numbers lead to valid POS) n-grams, longer numbers, especially if they contain punctuation, lead to n-grams of invalid length.

- Another problem we encountered was that, for some sentences, the Myers Difference algorithm [36] failed to calculate a difference and thus, without reference error annotation to compare to, we had to skip the affected sentences.

- Finally we skipped sentences that were shorter than the currently used n-gram size as such sentences cannot be classified by our approach..

Table 5.4 lists training results for optimum probability thresholds and their respective word level precision. The table is subdivided into the type of n-gram counts and probabilities each language model uses to detect errors. Probability threshold are given in logarithmic scale. Hence, e.g. the table's last probability entry 3.173 has an actual value of $10^{-3.173} = 0.000671$. Since we already choose Stupid Backoff to minimize probability calculations we also trained our models using conditional probabilities, but omitted the corresponding values from Table 5.4 since conditional probabilities consistently produced a lower precision when compared to Stupid Backoff.

N-Gram Type	**Optimum Threshold**	**Resulting Precision**
n-gram counts for		
3-grams	0	0.451
4-grams	0	0.478
n-gram probabilities for		
word 3-grams	−3.755	0.474
word 4-grams	−3.727	**0.521**
prefix 3-grams	−1.298	0.347
prefix 4-grams	−1.297	0.424
POS 3-grams	−1.397	0.404
POS 4-grams	−1.128	0.462
interpolation of probabilities for		
word and prefix 3-grams	−2.290	0.433
word and prefix 4-grams	−2.102	0.494
word and POS 3-grams	−2.290	0.455
word and POS 4-grams	−2.198	0.504
normalization of probabilities for		
word and prefix 3-grams	−3.311	0.449
word and prefix 4-grams	−2.852	0.503
word and POS 3-grams	−3.850	0.487
word and POS 4-grams	−3.173	**0.522**

Table 5.4: List of the each model type's optimum threshold τ obtained during language model training along with the precision that was achieved. Language models are sorted by the type of n-gram information they use for error detection.

5.3 Testset

We evaluate our error detection approaches on three corpora. First, a 10% test set from Wagner et al.'s corpus of generated errors [14], second, the Montclair Electronic Language Database (MELD) corpus [38] and third, the 123 Mass Noun Sentences corpus by Brockett et al. [28]. While we trained our language models on an 80% training set of the first corpus the latter two corpora are separate collections with different error types. As described in section 4.1, detection performance is measured by comparing *reference error annotations from a corpus* with *algorithmically obtained error annotations*. To make the corpora error annotations comparable to our algorithmic error annotations, we first extracted the position of each corpus error annotation along with the erroneous sentence that contains it. That way, for each erroneous corpus sentence, we can detect errors and compare our algorithmically obtained error positions with the reference error positions from the corpora to measure error detection performance at word and sentence level. We overview each of the three corpora and explain how we extracted the required erroneous sentences and accompanying reference error positions from them.

10% *Subsample of Wagner et al.'s corpus of generated errors*

The first test set consists of artificially created errors. It contains $9,413,338$ words and is organized into five error types: agreement errors, extra word errors, missing word errors, spelling errors and verb form errors. The reference error positions in this corpus were determined using the Myers string difference algorithm [36] on the correct and erroneous version of each sentence. To get the required sentences, we split the corpus up by error type and extracted a correct and an erroneous version of each sentence using a script Wagner et al. were kind enough to provide us with. This allows us to measure error detection performance for each of the five error types separately.

The Montclair Electronic Language Database

The second corpus consists of 58 essays written by second language learners from different language backgrounds [38]. The corpus contains $6,553$ words and was annotated by linguists using a simple {error,replacement} annotation scheme. We turned the corpus into a collection of pairs of correct and erroneous sentences so that we can calculate reference and algorithmic error annotations on the fly in order to measure detection performance.

123 *Mass Noun Sentences*

The last test set we use is a 1,813 word collection that was collected by Brockett et al. as a corpus of mass noun (uncountable noun) confusions [28], but also contains subject-verb agreement errors such as "knowledge are" and count noun (countable noun) errors. We manually annotated the errors in this corpus to create a gold standard to measure against. Furthermore, we acquired another set of human error annotations using a crowdsourcing service provided by Amazon[2] to be able to compare algorithmic detection quality with human detection quality.

5.4 Experiment Evaluation

In this section we present word and sentence-level error detection results for each of the three test set corpora. We analyse each of the corpora and compare their results to find answers to the following questions:

- *Influence of n-gram size.* How does the length of n-grams influence detection performance and what n-gram length, 3 or 4, results in better detection performance?
- *Influence of class-based models.* Can the class-based language models outperform the word n-gram models and what combination method, interpolation or normalization, works best?
- *Comparing sentence with word measures.* How well is the word-level measure suited to measure error detection performance and how do results compare to sentence-level detection performance?
- *Comparing algorithms with humans.* How well do laymen human annotators detect errors compared to algorithmic error detection?

[2] The service can be found at https://www.mturk.com.

*The Montclair Electronic Language Database (*MELD*)*

N-Gram Type	P_s	R_s	P_w	R_w
n-gram counts for				
3-grams	0.753	0.988	0.291	0.391
4-grams	0.751	0.994	0.447	0.751
n-gram probabilities for				
word 3-grams	**0.806**	0.850	0.376	0.403
word 4-grams	**0.803**	0.856	**0.494**	0.521
POS 3-grams	0.752	0.982	0.315	0.608
POS 4-grams	0.747	0.994	0.430	0.793
prefix 3-grams	0.745	1.0	0.317	0.870
prefix 4-grams	0.747	0.994	0.421	0.869
normalization of probabilities for				
word, POS 3-grams	0.797	0.874	0.276	0.349
word, POS 4-grams	0.775	0.910	0.423	0.512
word, prefix 3-grams	0.788	0.892	0.387	0.461
word, prefix 4-grams	**0.800**	**0.934**	**0.512**	**0.589**
interpolation of probabilities for				
word, POS 3-grams	0.793	0.898	0.353	0.407
word, POS 4-grams	**0.803**	**0.928**	0.487	0.548
word, prefix 3-grams	0.794	0.856	0.382	0.402
word, prefix 4-grams	0.780	0.916	**0.505**	**0.581**

Table 5.5: Error detection performance results for the MELD corpus. P_s = precision at sentence level, R_s = recall at sentence level, P_w = precision at word level, R_w = recall at word level, POS = parts of speech.

The results in Table 5.5 indicate that at *sentence level* the n-gram length only has an effect on the word class combination methods where 4-grams consistently outperform 3-grams. The highest precision values can be observed for word 3- and 4-gram probabilities, the interpolation of word and POS 4-gram probabilities, and a normalization approach that uses word and prefix 4-grams. Compared to word n-gram probabilities both class combination methods improve recall by around $7 - 8\%$.

At word level, 4-grams clearly outperform their 3-gram counterparts throughout all approaches. Though the class combination methods achieve the highest word-level precision values using 4-gram normalization and interpolation, it can be noticed, that word 4-gram probabilities, at 49.4% precision, already perform better than most other approaches.

Although we did not find any previous sentence-level evaluation for the MELD corpus, we did find word-level evaluations conducted by Fitzpatrick and Seegmiller who created the corpus [38, 39]. They measured annotation agreement between multiple human annotators and found that annotators not just annotate grammatical errors but also stylistic errors like long sentences and corrected sentences to their rhetorical preference, e.g. correcting "you" to "one's". While the latter two kinds of errors are not detectable by our algorithms, we compare our detection results with their evaluation of annotator agreement in Table 5.6 [38]. The table compares three different annotation methods, two produced by Fitzpatrick and Seegmiller and the third by our algorithmic error detection. The first method measures annotation agreement between human annotators. It can be observed that agreement results vary strongly between different annotators (marked bold) but that it averages around 55% recall and 64% precision [38, 39]. The second method measures agreement using the authors of the essays as annotators which produces a notably higher agreement. The third approach represents our algorithmic word-level error annotation, which coincides with the definitions for precision and recall given by Fitzpatrick and Seegmiller. Compared to the human annotation, precision values produced by our algorithms are 4% higher, which is rather unexpected since humans should generally find errors more accurately than our algorithms. On the other hand we lack behind in recall by 13%, which fits our expectations. Though both human and algorithmic annotation agreement show similar results, we merely take this as an indication of our previously made observations regarding the ambiguity of errors (see section 4.2). To avoid low agreement rates due to error ambiguity, error detection results should perhaps be evaluated by humans to find out how useful the algorithmic annotation are for an actual user. But such an in-depth evaluation is beyond the scope of this work.

Annotators	**Agreement Recall**	**Agreement Precision**
Human annotators		
J and L essay set 1	0.54	0.58
J and L essay set 2	0.57	0.78
J and N essay set 1	0.58	0.48
J and N essay set 2	**0.37**	**0.54**
L and N essay set 1	0.65	0.70
L and N essay set 2	**0.60**	**0.78**
average	0.55	0.64
Self-annotations by the		
original essay authors set 1	0.73	0.84
original essay authors set 2	0.76	0.90
Algorithmic annotation		
MELD reference and algorithm	0.59	0.51

Table 5.6: Annotator descriptions and results are taken from [38]. J, L, N are the annotators used for the evaluation. The values describe annotator agreement for two sets of essays from the MELD corpus.

ESL *123 Mass Noun corpus*

This corpus is relatively small and thus the results drawn from it carry little statistical weight. As no reference error annotations were provided with this corpus we created our own set of reference error annotations to measure against. The detection results are presented in Table 5.7.

At sentence level, detection results are near optimum which, given that the corpus contains very few error-free sentences, does not surprise. N-gram length only seems to improve detection results for the combination models, however the best performance is achieved using word 4-gram counts and prefix 3-gram probabilities, which most likely

can be ascribed to the size of the corpus. This becomes even more obvious because the combination methods are actually outperformed by n-gram counts and probabilities.

At word-level, 4-grams clearly improve detection performance when compared to 3-grams, and the combination approaches outperform the other methods with the exception of word 4-gram probabilities, which show the second best precision of all methods but lack behind in recall.

Humans outperform algorithmic detection at both sentence and word level. They achieve a higher precision at word level and a considerably larger recall. The performance results, like the results drawn from the MELD corpus suggest that even in this small corpus with relatively few different error types, ambiguities seem to influence the measurements. Another reason for the unexpectedly low results achieved by the human annotators might be the fact that we created the reference error annotations ourselves, which could explain the low agreement between humans and the reference annotation. Additionally, we did not check the human error annotations for correctness prior to using them for evaluation as removing errors from the human annotations would have skewed the test results towards our expectations. To further investigate the problem of error ambiguity, we measured algorithmic detection performance against the human annotations and obtained a precision of 0.494 at a recall of 0.609, which roughly coincides with the results we obtained against the reference error annotations. Though the ESL mass noun corpus results confirm the results drawn from the MELD corpus, the 123 ESL mass noun corpus is too small in size to produce conclusive results.

N-Gram Type	P_s	R_s	P_w	P_w
n-gram counts for				
3-grams	1.0	0.981	0.389	0.592
4-grams	1.0	**0.990**	0.470	0.820
n-gram probabilities for				
word 3-grams	1.0	0.945	0.441	0.601
word 4-grams	1.0	0.909	**0.502**	0.627
POS 3-grams	1.0	0.944	0.352	0.478
POS 4-grams	1.0	0.916	0.443	0.567
prefix 3-grams	1.0	**0.990**	0.346	0.821
prefix 4-grams	1.0	0.972	0.451	0.798
normalization of probabilities for				
word, POS 3-grams	1.0	0.842	0.374	0.412
word, POS 4-grams	1.0	0.842	0.440	0.480
word, prefix 3-grams	1.0	0.936	0.436	0.628
word, prefix 4-grams	1.0	0.927	0.494	0.676
interpolation of probabilities for				
word, POS 3-grams	1.0	0.898	0.415	0.554
word, POS 4-grams	1.0	0.888	0.469	0.599
word, prefix 3-grams	1.0	0.927	0.434	0.600
word, prefix 4-grams	1.0	0.927	**0.517**	**0.686**
Human annotation				
manual	1.0	**1.0**	**0.533**	**0.776**

Table 5.7: Error detection performance results for the ESL mass noun corpus. P_s = precision at sentence-level, R_s = recall at sentence-level, P_w = precision at word-level, R_w = recall at word-level, 3g = 3-grams, 4g = 4-grams, POS = parts-of-speech.

BNC *corpus of artificially created errors*

The BNC corpus is our largest test set and split into errors of different types, which allows us to compare detection performance per type. Due to its size it is also the statistically most valid corpus and allows us to make more general assumptions. Table 5.8 shows a selection of the best error detection performance results, organized by type. A complete list of all 80 results would be too long.

At sentence level, error detection results are close to optimum, which we expected because the test set only contains erroneous sentences and language model error detection approaches tend to detect too many errors rather than too few [14]. The best detection performance for all five error types was consistently achieved by word counts of 4-grams.

At word level, the best error detection results were produced by normalizing word n-gram probabilities with part-of-speech n-gram probabilities as described in section 3.2. The detection results for missed word errors do not fit the other results as both precision and recall are considerably lower than for the rest of the error types, which is likely caused by a weakness in our reference error extraction mechanism. Furthermore, it can be noticed that the results show a similar error detection precision as was observed for the previous two corpora. Recall, however, is significantly higher for the BNC corpus, which does not surprise since we optimized our language models using 80% of the corpus as training set.

sentence level	**Agreement**	**Extra**	**Miss**	**Spell**	**Verb**
Type	4g counts	4g counts	4g counts	4g counts	4g counts
Precison	1,0	1.0	1.0	1.0	1.0
Recall	0.996	0.996	0.982	0.994	0.995
word level					
Type	POS 4g norm.	POS 4g norm.	4g counts	POS 4g norm.	POS 4g norm.
Precision	0.509	0.522	0.438	0.520	0.545
Recall	0.759	0.730	0.821	0.741	0.791

Table 5.8: Error detection performance results for the artificial BNC corpus sorted by error type. Agreement errors, extra word errors, missed word errors, spelling errors, and verb form errors.

5.5 Experiment Conclusions

In this section we summarize the test results for the three corpora and answer our previously stated questions about the influence of n-gram length, how well the measures perform, whether or not the advanced approaches improve detection performance and what the relation between error ambiguity and the different measures is.

- *Influence of n-gram size.* Throughout the different approaches it has become clear that using 4-grams instead of 3-grams increases error detection performance, especially for the class-word combination methods.

- *Influence of class-based models.* The most naive approach we use are word n-gram counts. At sentence level, these seem to produce the highest recall when used on test sets that mostly consist of ungrammatical sentences. However, as the sentence-level evaluations performed on the MELD corpus show, they are outperformed by other methods on test sets that also contain grammatical sentences, i.e., sentences without errors. The lowered precision in conjunction with the near-perfect sentence-level recall (see Table 5.5) leads us to assume that the naive word n-gram detection method tends to detect errors where there are none (false positives). The second error detection approach described in Section 3.2 uses word n-gram probabilities obtained via Google's Stupid Backoff method [6]. This method has proven to provide good baseline error detection performance and is only outperformed by our advanced models. Results for the last error detection approach, the class-word combination methods, are somewhat inconclusive. While normalization between word and part-of-speech probabilities performs best on the artificial errors, prefix-based combination methods worked better on actual learner errors. We can only assume that this occurs due to ambiguities in POS tagging as mentioned by Ilarraza and Oronoz [32, 33].

- *Comparing the sentence-level measure with word-level measures.* The differences in error detection performance at sentence and word level are salient. While detection at sentence level neither tells us where an error is located within a sentence nor how many errors a sentence contains, the sentence measure seems to be less susceptible to error ambiguities when compared to the word-level measure. This might be the reason why it is so frequently used for error detection evaluation and as Ilarraza and Oronoz state "the influence of error ambiguity is usually forgotten during the design of error detection systems" [32, 33]. Furthermore, our evaluation of

the MELD and the ESL mass noun corpus suggest that error annotation at word level measures annotation agreement between different annotators, and that, as we pointed out in section 4.2, annotation agreement is influenced by error ambiguities as well as an annotator's personal preference [38, 39]. A possible solution to the agreement problem could be an error detection system that allows its users to rate the usefulness of the algorithmic error annotations they are presented with by the system, but such in-depth evaluation is beyond the scope of this work. Additionally, these last findings are to be taken with a grain of salt as the results they are based on were drawn from data sets that are to small to be fully conclusive. We therefore conclude that the impact error annotation ambiguities have on error detection performance measures need further investigation to produce conclusive results.

Bibliography

[1] D. Jurafsky and J. H. Martin, *Speech and Language Processing*. Prentice Hall, 2 ed., May 2008.

[2] T. Brants and A. Franz, "Web 1t 5-gram version 1." Linguistic Data Consortium, Philadelphia, 2006.

[3] L. Carroll, A. B. Woodward, and S. Shields, *Alice's adventures in wonderland*. Edward Arnold Australia, Caulfield East, Vic., 1985.

[4] J. Tetreault, J. Burstein, and C. Leacock, "Fifth Workshop on Innovative Use of NLP for Building Educational Applications," 2010.

[5] S. Chen and J. Goodman, "An Empirical Study of Smoothing Techniques for Language Modeling," in *Proceedings of the 34th annual meeting on Association for Computational Linguistics*, pp. 310–318, Association for Computational Linguistics, 1996.

[6] T. Brants, A. Popat, P. Xu, F. Och, and J. Dean, "Large language models in machine translation," in *In EMNLP*, Citeseer, 2007.

[7] C. Samuelsson and W. Reichl, "A class-based language model for large-vocabulary speech recognition extracted from part-of-speech statistics," in *Proceedings of the Acoustics, Speech, and Signal Processing, 1999. on 1999 IEEE International Conference - Volume 01*, (Washington, DC, USA), pp. 537–540, IEEE Computer Society, 1999.

[8] J. P. Broderick Ph.D., *Modern English Linguistics: A Structural and Transformational Grammar*. Thomas Y Crowell Company, 1 ed., 1975.

[9] "Merriam webster definition of spelling." online dictionanry.

[10] D. Fossati and B. Di Eugenio, "A mixed Trigrams Approach for Context Sensitive Spell Checking," *Computational Linguistics and Intelligent Text Processing*, pp. 623–633, 2010.

[11] "The plain language campaign." official government webarticle. http://www.plainlanguage.gov/whatisPL/definitions/eagleson.cfm.

[12] I. Fraser and L. Hodson, "Twenty-one kicks at the grammar horse," *English Journal*, vol. 67, no. 9, pp. 49–54, 1978.

[13] C. Leacock, M. Chodorow, M. Gamon, and J. Tetreault, "Automated Grammatical Error Detection for Language Learners," *Synthesis Lectures on Human Language Technologies*, vol. 3, no. 1, pp. 1–134, 2010.

[14] J. Wagner, J. Foster, and J. V. Genabith, "Genabith. a comparative evaluation of deep and shallow approaches to the automatic detection of common grammatical errors," in *In Proceedings of the 2007 Joint Conference on Empirical Methods in Natural Language Processing and Computational Natural Language Learning (EMNLP-CoNLL*, pp. 112–121, 2007.

[15] Y. Tono, "Learner corpora: design, development and applications," in *Proceedings of the 2003 Corpus Linguistics Conference*, pp. 800–809, Citeseer, 2003.

[16] A. Rozovskaya and D. Roth, "Annotating ESL Errors: Challenges and Rewards," *Urbana*, vol. 51, p. 61801, 2010.

[17] D. Nicholls, "The Cambridge Learner Corpus-error coding and analysis for lexicography and ELT," in *Proceedings of the Corpus Linguistics 2003 conference*, pp. 572–581, 2003.

[18] "Chinese learner english corpus (clec)." official website. http://langbank.engl.polyu.edu.hk/corpus/clec.html.

[19] J. Hana, A. Rosen, S. Škodová, and B. Štindlová, "Error-tagged Learner Corpus of Czech," in *Proceedings of the Fourth Linguistic Annotation Workshop (LAW IV), ACL*, pp. 11–19, 2010.

[20] "English Taiwan Learner Corpus (TLC)." official website. http://lrn.ncu.edu.tw/Teacher

[21] P. Siemen, A. Lüdeling, and F. Müller, "FALKO-ein fehlerannotiertes Lernerkorpus des Deutschen," *Proceedings of Konvens 2006*, 2006.

[22] S. Granger, "Error-tagged learner corpora and CALL: a promising synergy," *CALICO JOURNAL*, vol. 20, no. 3, pp. 465–480, 2003.

[23] R. Shih, "Compiling Taiwanese learner corpus of English," *Computational Linguistics*, vol. 5, no. 2, pp. 87–100, 2000.

[24] S. Granger, E. Dagneaux, F. Meunier, and M. Paquot, *International Corpus of Learner English V2*. Louvain-la-Neuve: Presses universitaires de Louvain. Available from http://www. i6doc. com, 2009.

[25] A. Díaz-Negrillo and J. Fernández Domínguez, "Error tagging systems for learner corpora," *Revista española de lingüística aplicada*, no. 19, p. 83, 2006.

[26] "The british national corpus." official website.

[27] J. Bigert and O. Knutsson, "Robust error detection: A hybrid approach combining unsupervised error detection and linguistic knowledge," in *Proc. 2nd Workshop Robust Methods in Analysis of Natural language Data (ROMAND02), Frascati, Italy*, pp. 10–19, 2002.

[28] C. Brockett, W. Dolan, and M. Gamon, "Correcting ESL errors using phrasal SMT techniques," in *Proceedings of the 21st International Conference on Computational Linguistics and the 44th annual meeting of the Association for Computational Linguistics*, pp. 249–256, Association for Computational Linguistics, 2006.

[29] P. Meyer, *Synchronic English linguistics: an introduction.* Gunter Narr Verlag, 2005.

[30] J. Birn, "Detecting grammar errors with Lingsoft's Swedish grammar checker," in *The 12th Nordic Conference of Computational Linguistics*, pp. 28–40, 2000.

[31] J. Tetreault and M. Chodorow, "The ups and downs of preposition error detection in ESL writing," in *Proceedings of the 22nd International Conference on Computational Linguistics-Volume 1*, pp. 865–872, Association for Computational Linguistics, 2008.

[32] A. de Ilarraza, K. Gojenola, and M. Oronoz, "Evaluating the Impact of Morphosyntactic Ambiguity in Grammatical Error Detection," *Recent Advances in Natural Language Processing ISSN*, pp. 1313–8502.

[33] M. Oronoz, A. Díaz de Ilarraza, and K. Gojenola, "Design and Evaluation of an Agreement Error Detection System: Testing the Effect of Ambiguity, Parser and Corpus Type," *Advances in Natural Language Processing*, pp. 281–292, 2010.

[34] C. Leacock, M. Gamon, and C. Brockett, "User input and interactions on microsoft research esl assistant," 2009.

[35] K. Papineni, S. Roukos, T. Ward, and W.-J. Zhu, "Bleu: a method for automatic evaluation of machine translation," in *ACL*, pp. 311–318, 2002.

[36] E. W. Myers, "An o(nd) difference algorithm and its variations," *Algorithmica*, vol. 1, no. 2, pp. 251–266, 1986.

[37] O. Mason, "QTAG-A portable probabilistic tagger," 1997.

[38] E. Fitzpatrick and M. Seegmiller, "The Montclair electronic language database project," *Language and Computers*, vol. 52, no. 1, pp. 223–237, 2004.

[39] E. Fitzpatrick and M. Seegmiller, "Practical aspects of corpus tagging," *Practical Applications in Language Corpora*, 2003.