Classification and Generation of Grammatical Errors

by

Anthony Penniston

Bachelor of Science, Ryerson University, 2009

A thesis

presented to Ryerson University

in partial fulfillment of the

requirements for the degree of

Master of Science

in the Program of

Computer Science

Toronto, Ontario, Canada, 2014

AUTHOR'S DECLARATION FOR ELECTRONIC SUBMISSION OF A THESIS

I hereby declare that I am the sole author of this thesis. This is a true copy of the thesis, including any required final revisions, as accepted by my examiners.

I authorize Ryerson University to lend this thesis to other institutions or individuals for the purpose of scholarly research.

I further authorize Ryerson University to reproduce this thesis by photocopying or by other means, in total or in part, at the request of other institutions or individuals for the purpose of scholarly research.

I understand that my thesis may be made electronically available to the public.

Classification and Generation of

Grammatical Errors

Master of Science 2014

Anthony Penniston

Computer Science

Ryerson University

Abstract

The grammatical structure of natural language shapes and defines nearly every mode of communication, especially in the digital and written form; the misuse of grammar is a common and natural nuisance, and a strategy for automatically detecting mistakes in grammatical syntax presents a challenge worth solving. This thesis research seeks to address the challenge, and in doing so, defines and implements a unique approach that combines machine-learning and statistical natural language processing techniques. Several important methods are established by this research: (1) the automated and systematic generation of grammatical errors and parallel error corpora; (2) the definition and extraction of over 150 features of a sentence; and (3) the application of various machine-learning classification algorithms on extracted feature data, in order to classify and predict the grammaticality of a sentence.

Acknowledgements

I express my greatest gratitude to my supervisor, Dr. Eric Harley, for introducing and piquing my interest in the topic; I am humbled and grateful for his enduring assistance, tireless patience, and thoughtful encouragement. He has provided advice and direction, especially where I have encountered pause or hesitation, and has inspired new ideas and avenues for exploration within this research. I am thankful for his endless support.

I also extend thanks to the members of my thesis dissertation committee, Dr. Alex Ferworn, Dr. Cherie Ding, and Dr. Isaac Woungang, for their time and effort in reviewing my work. Their valuable feedback and insights have served to improve the relevancy and composition of this thesis, as well as my academic mettle.

Lastly, I wish to convey my appreciation to the Department of Computer Science at Ryerson University, the faculty and staff, who have instructed and encouraged me to pursue my academic goals along the way.

Contents

Declaration . . . iii
Abstract . . . v
Acknowledgements . . . vii
List of Tables . . . xi
List of Figures . . . xiii
List of Appendices . . . xv

1 Introduction **1**
1.1 Contributions . . . 3
1.2 Thesis outline . . . 4

2 Review of NLP & Related Research **5**
2.1 Tagging . . . 6
2.1.1 Rule-Based . . . 7
2.1.2 Maximum Entropy . . . 7
2.2 Parsing . . . 10
2.2.1 Probabilistic Context-Free Grammar (PCFG) . . . 10
2.2.2 Maximum Entropy . . . 12
2.3 Grammar Verification . . . 14
2.3.1 Heuristic approaches . . . 15
2.3.2 Machine-learning approaches . . . 17

3 Methodology **21**
3.1 Common NLP tasks . . . 22
3.1.1 Tokenization . . . 23
3.1.2 Part-of-Speech Tagging . . . 23
3.1.3 Parsing . . . 23
3.1.4 Natural Language Generation (NLG) . . . 23
3.2 Collecting & generating example data . . . 24
3.2.1 Example data for grammatical correctness . . . 24
3.2.2 Example data for grammatical incorrectness . . . 24

3.2.3 CorpusTools . . . 25
3.2.4 Corpora . . . 31
3.3 Extracting feature data . . . 32
3.3.1 GrammarTools . . . 33
3.3.2 Feature extraction . . . 35
3.4 Machine-learning classification . . . 36
3.4.1 Classification algorithms . . . 36
3.4.2 Mitigating bias in unbalanced datasets . . . 39
3.4.3 Training & testing . . . 40

4 Results & Discussion **41**
4.1 Results by corpus . . . 42
4.1.1 Verbatim . . . 42
4.1.2 Non-Fiction . . . 46
4.1.3 Travel Guides . . . 49
4.1.4 Berk . . . 52
4.1.5 MASC . . . 56
4.2 Discussion of classifier performance . . . 60
4.3 Comparison with similar methods . . . 64

5 Conclusion & Future Work **67**
5.1 Findings & contributions . . . 67
5.2 Limitations . . . 68
5.3 Future Work . . . 68

References **103**

Glossary **105**

Acronyms **107**

List of Tables

2.1 Sentence features defined in [1][1] . . . 17
2.2 Classification results from [1] . . . 18

4.1 Classification algorithms and their abbreviated labels . . . 42
4.2 Verbatim: Displace . . . 44
4.3 Verbatim: Omit . . . 44
4.4 Verbatim: Tense . . . 45
4.5 Verbatim: Number . . . 45
4.6 Verbatim: Object . . . 45
4.7 Verbatim: Insert . . . 45
4.8 Non-Fiction: Displace . . . 48
4.9 Non-Fiction: Omit . . . 48
4.10 Non-Fiction: Tense . . . 48
4.11 Non-Fiction: Number . . . 48
4.12 Non-Fiction: Object . . . 49
4.13 Non-Fiction: Insert . . . 49
4.14 Travel Guides: Displace . . . 51
4.15 Travel Guides: Omit . . . 51
4.16 Travel Guides: Tense . . . 52
4.17 Travel Guides: Number . . . 52
4.18 Travel Guides: Object . . . 52
4.19 Travel Guides: Insert . . . 52
4.20 Berk: Displace . . . 54
4.21 Berk: Insert (*verb*) . . . 54
4.22 Berk: Omit . . . 54
4.23 Berk: Tense . . . 54
4.24 Berk: Omit (*verb*) . . . 55
4.25 Berk: Number . . . 55
4.26 Berk: Object . . . 55
4.27 Berk: Insert . . . 55

4.28 Berk: Displace (*verb*) 56
4.29 Berk: Number (*noun, pronoun*) 56
4.30 Displace 59
4.31 MASC: Omit 59
4.32 MASC: *Any* 59
4.33 MASC: Tense 59
4.34 MASC: Number 60
4.35 MASC: Object 60
4.36 MASC: Insert 60

List of Figures

2.1 An example parse tree. 10
2.2 A PCFG parse tree with associated probabilities. 12
2.3 Trees produced after each pass of *build* and *check*. 14
2.4 Classification accuracy by error type [1] 18
2.5 Classification accuracy by error quantity [1] 19

3.1 A machine-learning approach to grammar verification. 22

4.1 Verbatim: Displace, Omit 43
4.2 Verbatim: Tense, Number 43
4.3 Verbatim: Object, Insert 44
4.4 Non-Fiction: Displace, Omit 46
4.5 Non-Fiction: Tense, Number 47
4.6 Non-Fiction: Object, Insert 47
4.7 Travel Guides: Displace, Omit 50
4.8 Travel Guides: Tense, Number 50
4.9 Travel Guides: Object, Insert 51
4.10 Berk: Insert (*verb*), Omit (*verb*) 53
4.11 Berk: Displace (*verb*), Number (*noun, pronoun*) 53
4.12 MASC: Displace, Omit 57
4.13 MASC: Tense, Number 57
4.14 MASC: Object, Insert 58
4.15 MASC: *Any* 58
4.16 Verbatim: Tense (1, 2 errors) comparison 62
4.17 Verbatim: Tense (3, 4 errors) comparison 62
4.18 Verbatim: Tense (5 errors) comparison 63

List of Appendices

1 Sentence features **71**

2 Sample corpus **91**

2.1 Grammatically correct sentences 91

2.2 Grammatically incorrect sentences 95

3 Sample error corpus **97**

3.1 Generated error sentences 97

Chapter 1

Introduction

Digital communication permeates many facets of modern technology, and with it comes a prevalence and persistence of language in the written form: it comprises the documents we write, e-mails we send, instant messages we exchange, the web pages we view. Naturally, this gives way to a higher tendency for its misuse, especially in the form of grammatical mistakes. The mode of written communication – such as electronic documents or printed paper – inherently conveys some degree of permanency: the reader is usually free to revisit sections of the document; thus the presence of grammatical errors may become more apparent as a reader attempts to re-parse errorful or incoherent sentences. Frequent or particularly egregious grammatical mistakes can often be impression-forming of the work and its author.

Naturally, there is good incentive to eliminate grammatical errors wherever possible, and authors may go to great lengths to do so: tedious proofreading of multiple drafts, or even outsourcing to professional editing services. Even common everyday tasks, such as composing an e-mail or writing a letter, can often result in multiple rereads, checking for grammar and style. It would seem highly desirable then, to have an automated and feasible way of detecting grammatical errors where they exist. This premise, especially as it applies to automatic methods of detecting grammatical errors, forms the basis of the research work presented in this thesis.

For several decades, computers have made it possible to automate some of the work involved in proofreading, most reliably in the area of detecting and correcting errors in spelling (commonly referred to as *spell checking*)[2]. Similar attempts to automate the detection of grammatical errors exist, but largely grammar checkers to date have been met with limited success [3]. Grammar-checking utilities have come bundled with some notable document processing applications, such as various versions of *Microsoft Word* [4]. However, these and other such grammar checkers can notoriously be unreliable in their detection of errors and the corrections they suggest; often to the point of nuisance [3].

Some approaches to grammar checking rely on collections of prescriptive rules (known as *heuristics*) in order to detect and provide corrections for various forms of invalid grammar [5, 6]. Purely rule-based techniques may not adapt well to syntax that has not been anticipated by the original set of rules. To overcome this limitation, a more dynamic and flexible approach is desirable; one conceivably similar to the way humans learn and acquire language: by example [7, 8].

Given sufficiently many examples (or *evidence*) of grammatically correct or incorrect constructions, it may be possible to discern a pattern between the two classes. Certainly, humans perform this kind of learning intuitively: grammar is acquired fluidly, through observing its use by native speakers [7]. Similarly, it is possible to have computational algorithms learn by example and make statistical observations about the frequency of patterns and *features* in data that may aid in predicting future events. This process is commonly referred to as *machine learning* [9]. It is the primary interest and focus of this research to develop a machine-learning method to detect and classify various forms of grammatical incorrectness, as learned from natural and synthesized example data.

There are a variety of challenges to address when dealing with natural human language from the context of computational algorithms. An entire domain of artificial intelligence, referred to as natural language processing (NLP) [10], is dedicated to analyzing and modeling natural human language through computational algorithms. Tasks common to NLP include basic text processing, such as grouping characters into words and sentences, as well as higher-level analysis such as determining the part of speech (POS) category for each word, and parsing sentences into relational structures such as syntax trees [10].

Another important consideration is the abundance and variety of grammatical errors. Sources of error can result from many common mistakes, for example, *subject-verb disagreements* in tense or number [11]. Grammatical errors may also arise from mistakes in word placement, and erroneous omissions or additions to a sentence. As well, they may vary with the author's grasp of the language – authors who are not fluent with the language (referred to as second-language learners [12]) are likely to make different types of errors than authors for whom the language is native. For second-language learners, even syntactically correct constructions may be regarded as problematic, if the wording is unnatural or otherwise atypical of what a native speaker would write given the same intention. Even when errors are present in the intention of a sentence, they may not manifest as errors in syntax. It is possible for an author to commit an error in the intended meaning of a sentence, and yet have the sentence remain syntactically valid:

Example 1.0.1.

The election officials gathered the ballots.
The election officially gathered the ballots.

The likely intended sentence in Example 1.0.1, is the first; however both sentences form a grammatically valid construction. Such errors often result from the inherent ambiguity present in natural languages [10]. Understanding the context and semantic meaning of a sentence is often required to disambiguate and detect these types of errors. Thus it is useful to distinguish between two high-level categories of errors in language use: (1) syntactical errors and (2) semantical errors. Since the research in this thesis seeks to detect grammatical errors only, its focus is primarily on errors of syntax at the sentence level.

An important part of predicting grammatical errors is in defining the types of errors that will be considered to render a sentence grammatically incorrect. Example data can then be constructed from sentences that exhibit these types of errors, and used in conjunction with example data that exhibits

no such errors (considered to be examples of grammatical correctness). Finally, feature extraction and machine-learning algorithms can be applied to the example data in order to learn, and make predictions about, the grammatical correctness of a sentence.

In support of this approach, this research accomplishes several major goals: (1) the collection and generation of a suitable dataset of example sentences; (2) the definition and extraction of a set of features from the example data; and (3) the application of machine-learning classification algorithms to the extracted feature data in order to predict the grammatical correctness of a sentence.

1.1 Contributions

This thesis research contributes a novel method of predicting grammatical errors using a machine-learning approach. The machine-learning process is supported by *feature data* that is extracted from the example data, using current techniques in statistical NLP. Results and comparisons are given for several different types of grammatical errors across various collections of English writing. In order to obtain sufficient example data, this thesis also contributes a method of constructing collections sentences containing synthesized grammatical errors. The tools that we provide for this are especially important, given that collections of purely incorrect sentences are not openly or widely available in the field. Further contributions are realized through our software implementations of the aforementioned methods, namely in the form of two freely-available, open-source software tools: *CorpusTools* and *GrammarTools* [13, 14].

Machine-learning methods for grammatical error prediction

A primary contribution of this research are methods for applying machine learning and natural language processing towards detecting and predicting grammatical errors. A multitude of sentence features (over 150, detailed in Chapter 3) have been defined using NLP techniques, and are extracted from a prepared dataset of sentences. The results of the machine-learning process are presented in Chapter 4, categorized by the grammatical error type, quantity, and domain of English writing.

CorpusTools *tools for generating grammatically parallel corpora*

Finding sources of grammatically correct examples may often be as trivial as collecting sentences from well-proofed sources of written text. However, collections of grammatically incorrect sentences are more difficult to find, since the natural tendency is to ignore or discard them. A software application was implemented during this research in order to generate collections of example sentences, correct and incorrect, in support of the machine-learning process. *CorpusTools* implements algorithms described in this research to generate various types of commonly occurring grammatical errors (such as *subject-verb disagreement*). It does this by aggregating collections of sentences – typically from commonly available sources – and producing a parallel collection of sentences that is nearly identical, except where each sentence exhibits one or more synthesized grammatical errors, as specified by the user. The algorithms and implementation are described in Chapter 3. CorpusTools has proved highly useful in the creation of the datasets necessary for the machine-learning process, and may prove as useful in similar NLP research, where currently, sources of grammatically errorful sentences are not yet widely available.

GrammarTools *tools for feature data extraction and classification*

GrammarTools is a suite of software designed to perform various forms of natural language processing and analysis of sentences in English, especially the extraction of feature data from written text sentences. GrammarTools performs various tasks such as part-of-speech tagging, chunking, and syntactical parsing, many of which rely on statistical NLP techniques (further described in Chapter 2). These tasks form the basis of higher-level analysis, such as the extraction of feature data from sentences, and preparing the extracted data for the machine-learning process; finally, it can use the resulting trained model to predict the grammatical correctness of a sentence. GrammarTools serves as an experimental platform for the research and results presented in this thesis, and may prove similarly useful to related research in natural language processing and grammar verification.

1.2 Thesis outline

The thesis is structured into five main chapters. The following, Chapter 2, surveys the field of natural language processing, and other such research that is related and of interest to this thesis. It gives treatment of the background theory and current techniques in NLP that are supplemental to this research, as well as discussing similar and related research in grammatical analysis and verification.

Methodology and implementations are described in Chapter 3. This chapter establishes three major approaches pursued in this research: grammatical error generation, feature extraction, and machine-learning classification. The rationale and theory supporting these approaches is discussed, and the implementing software systems are described in detail. Supplemental techniques and tools for NLP and machine learning are also given treatment at the beginning and end of the chapter.

Experimental results are presented in Chapter 4, particularly the results of machine-learning classification for various types of grammatical errors. The performance and accuracies of the classification algorithms used are measured and compared across several different variables, including grammatical error type, quantity, and domain of English writing. Interesting observations are discussed, and comparisons are drawn with results from related research.

A conclusive summary of the findings and contributions of this research, along with its limitations, are provided in Chapter 5. Further ideas and directions for future research are discussed towards the end.

Chapter 2

Review of NLP & Related Research

Natural language processing (NLP) is an emerging domain of computer science, comprised of artificial intelligence and linguistics [10, 15]. Its research and applications are compelling and far-reaching, spanning topics from text and syntactical parsing, to document classification, and even artificially intelligent conversational agents. Several important topics of NLP form the supporting theory of this thesis work: processing textual data, parsing it into words and grammatical constructs, and analyzing and verifying their grammatical structure. Text and characters are received as input and converted into tokens and words that form the lexicon of a language; these are then grouped into sentences. Words in a sentence are associated with their grammatical parts of speech (such as *noun* or *verb*), in a process known as *tagging*. Syntactical structure is parsed from the words to form a syntax tree. These fundamental techniques will be reviewed further in this chapter and often form the basis for higher-level applications of NLP, such as grammatical and semantical analysis.

Many of the approaches covered in this chapter (along with the thesis research itself) rely on collections of examples of written text, referred to as a *corpus* (plural, *corpora*), in order to learn and make statistical inferences about an observed phenomenon [15]. There exist several well-known corpora for the English language, such as the *Brown* corpus, the *Wall Street Journal* corpus, the *British National Corpus*, and a hybrid of the former two: the *Penn Treebank* corpora [16]. An important aspect of these (and any corpora intended for NLP use) is that it is *annotated* – sentences are labeled with information that is useful for NLP applications. This often includes details such as the beginning and end of a sentence; the parts of speech for each word, phrasal groupings, syntactic structure, and even semantic and contextual information about the sentence [16]. Example 2.0.1 below is a brief, arbitrary sample of text that may appear in a simple unannotated corpus:

Example 2.0.1.

> The election official gathers the ballots. The official results are quickly tallied and a winner is announced. Voters mark the ballots with their top three choices.

Annotating the first sentence in Example 2.0.1 with part-of-speech tags and syntactic structure gives it an appearance that is similar to what may be found in an annotated corpus:

Example 2.0.2.

```
<sentence>
 <nounphrase>
  <determiner>The</determiner>
  <noun>election</noun>
  <noun>official</noun>
 </nounphrase>
 <verbphrase>
  <verb>gathers</verb>
  <nounphrase>
   <determiner>the</determiner>
   <noun>ballots</noun>
  </nounphrase>
 </verbphrase>
</sentence>
```

Example 2.0.2 is a sample *treebank* corpus which contains part of speech (POS) tags (such as *noun* and *verb*) and the hierarchical syntactic structure of the sentence. In real-world corpora, the complete set of annotations (such as the set of allowed tags) is much more thorough; for example, The Penn Treebank defines over 50 word and syntax tags [16].

2.1 Tagging

Tagging is the process of categorizing each word in a sentence into its part of speech (e.g. *noun*, *verb*, *adjective*). During the tagging process, a part-of-speech tag is assigned to each word from the set of possible tags for that word, as defined in a tag dictionary [16]. However, words in English may have more than one usage (e.g. the word *'official'* in Example 2.0.1 appears twice, once as an adjective and once as a noun); due to this ambiguity, tagging may also be considered a process of disambiguation. There are several approaches to resolving the ambiguity, and the leading methods are categorized into *heuristic* and *probabilistic*. Heuristic tagging relies on a set of defined rules in order to determine the part of speech of a given word [10]. A straight-forward approach is to define rules that assign a part of speech to a word based on its most common or likely use (for example, a rule for the word *'official'* might simply assign it a tag of *adjective*). An improvement over this strategy is to have rules automatically *learned* from an annotated corpus. This technique is referred to as *transformation-based learning* [17]; it iteratively modifies the definition of its rules in order to minimize the rate of error in the tags it assigns. In contrast, probabilistic approaches view tagging as a process of statistical inference [15], where each word and its corresponding part of speech can be assigned a probability based on how likely it is to occur in the corpus, in a given context. A particularly effective probabilistic approach is the concept of *maximum entropy* [18, 19] applied towards tagging; it is able to capture and exploit contextual information from words and tags, and is especially effective when encountering unknown words.

2.1.1 Rule-Based

Heuristic approaches to part-of-speech tagging rely on a set of rules that assign POS tags. Rules may take into consideration context, such as the surrounding tags (preceding and following), or the characters that form the words. Initial attempts at rule-based tagging relied on hand-crafted rules in an attempt to capture the grammatical reasoning used by humans [17]; however, this is prone to bias and error, and can quickly become tedious and intractable. A better approach is to automatically derive the rules from a corpus of annotated text. This is what Brill proposed through *transformation-based learning* [17].

Brill's tagger begins by tagging each word in a sentence with an initial tag. The initial tags for each word may be arbitrarily chosen from the tag set, or assigned the most common tag, or even generated from another tagger. Once an initial tagging has been assigned, the algorithm applies a series of *transformations* that alter the initial taggings in an attempt to improve the accuracy of the tags (in comparison to the actual tags as annotated in the corpus). Transformations are defined by rules of the general form shown in Example 2.1.1:

Example 2.1.1.

change tag A to tag B if X

where X is a boolean condition. A specific instantiation of this rule appears in Example 2.1.2.

Example 2.1.2.

change tag $Noun$ to tag $Verb$ if previous tag is To

This rule specifies that a *noun* should be changed to a *verb* if it follows the word *'to'*. The general forms of these rules are given by templates (Brill defines six types [17]) which follow the form of Example 2.1.2 but vary in the types of conditions for X. For example, one condition might compare the suffix or prefix of the current word, another may compare the previous or following adjacent tags.

Brill's tagger performs several passes, and instantiates rules from the templates during each pass, with different values for tags A and B and condition X. A newly created rule is discarded if it does not improve tagging accuracy on the corpus. The algorithm terminates when either there are no more rules that will increase overall accuracy, or until a sufficiently-high accuracy has been achieved.

Brill reports an accuracy of as high as 97.2% using transformation-based learning on The Penn Treebank's Wall Street Journal corpus [17]. An accuracy of 97% can still be achieved with only 200 transformations (starting from an initial tagging produced by a simple tagger), and only 82 transformations are needed to achieve an accuracy of 96.7% (matching the probabilistic tagger used as comparison). Brill devised further strategies to deal with unknown words by comparing suffix (such as "-ing"), prefix, and capitalization, resulting in an accuracy of 96.6% [17] for tagging unknown words.

2.1.2 Maximum Entropy

The concept of *maximum entropy*, as it applies to statistical modeling, is the principle that the preferred model is the one that makes the least amount of assumptions given what has been observed [18]. In

the context of natural language processing and this thesis, the *observation* is a corpus of written text. In the case of tagging, the model attempts to predict the part-of-speech tag for any given word. For example, consider a simplified tag set that consists of only the tags *noun*, *verb*, and *adjective*. In the simple corpus given earlier Example 2.0.1, an observation can be made that 23% of the words are *nouns*. If this is the only observation available, there are several possible models that assign a probability to each of the tags (*noun*, *verb*, or *adjective*) that agree with what is observable from the corpus; Example 2.1.3 offers three such models:

Example 2.1.3.

$p(noun) = 0.230$	$p(noun) = 0.230$	$p(noun) = 0.230$
$p(verb) = 0.570$	$p(verb) = 0.600$	$p(verb) = 0.385$
$p(adj) = 0.200$	$p(adj) = 0.170$	$p(adj) = 0.385$

In each of the these models, the probability of a noun, $p(noun)$, takes into account the observation that a *noun* occurs 23% of the time in the corpus. However, the first two models seemingly make arbitrary decisions about the probabilities of a *verb* or *adjective*. If nothing is yet known about these probabilities, the third model presents the most *uniform* choice; it distributes the probability evenly for unknown events. While it is trivial to determine the model with maximum entropy in this case, as more observations are taken into account, more complex constraints are placed on the model. The constraints ensure that the model agrees with what has been observed from the corpus; they are typically expressed as binary-valued functions known as *features* [18]. For example, suppose for a word w an observation is made that its tag is *noun* and it is preceded by an *adjective*. This observation can be encoded as the following feature:

Example 2.1.4.

$$f(h_i, t_i) = \begin{cases} 1 & \text{if } t = noun \text{ and } t_{i-1} = adjective \\ 0 & \text{otherwise} \end{cases}$$

where t_i is the observed tag and h_i is the context of the tag; the statement $t_{i-1} = adjective$ is true if an *adjective* appears directly before tag t_i given the context h_i. The context of a tag can contain arbitrary information such as the surrounding tags and words, the lexical characters of a tag, its position in the sentence, or anything else deemed useful for the application. Once a set of features has been defined, the expected value of each feature is calculated from the corpus based on the frequency of each tag t and its context h, which expresses how often each feature $f(h, t)$ is active. A constraint is then introduced: the expected value of each feature function f (as observed empirically from the corpus) must match the expected value for that feature as predicted by the model. This constraint is described by Equation 2.1

$$\sum_{h,t} \tilde{p}(h) p(t|h) f(h, t) = \sum_{h,t} \tilde{p}(h, t) f(h, t) \tag{2.1}$$

where $\tilde{p}(h)$ is the probability of the context h occurring in the corpus; $\tilde{p}(h,t)$ is the probability of context h occurring given a tag t; and $p(t|h)$ is the probability of the tag t occurring given context h as predicted by the model. The left-hand side of Equation 2.1 is the expected value of the feature function f as given by the model p over all pairs (h,t), and the right-hand side is the expected value of f as observed empirically from the corpus. Applying the concept of maximum entropy, the model that satisfies Equation 2.1 and exhibits the most uniformity should be preferred. The measure of uniformity is given by conditional entropy [18]:

$$H(p) = -\sum_{h,t} \tilde{p}(h)p(t|h)\log p(t|h) \tag{2.2}$$

Thus the model p which maximizes entropy is the preferred model; furthermore, it can be shown that this model is unique [18].

The mathematical framework of maximum entropy can be applied to many different problems [18] providing that an appropriate feature set is defined. Ratnaparkhi's maximum entropy tagger [19] applies these concepts to the NLP task of tagging. In the general case, feature functions are generated for each tag t, with the context including the previous two words and tags, the current word, and the following two words and tags. In the case of rare words (appearing less than five times), additional context is provided, including the prefix and suffix of the word, and whether it contains numbers, uppercase characters, or hyphens. Example 2.1.5, gives the template for a feature function that considers a tag t_i and it's previous tag t_{i-1}.

Example 2.1.5.

$$f(h_i,t_i) = \begin{cases} 1 & \text{if } t_{i-1} = Y \text{ and } t_i = X \\ 0 & \text{otherwise} \end{cases}$$

where h_i refers to the context or history of tag t_i, and X and Y represent arbitrary tags (such as *noun* or *verb*). Specific features can be instantiated from the above template; for example, the following feature is active when the current word is tagged *noun* and the previous word is tagged *adjective*:

Example 2.1.6.

$$f(h_i,t_i) = \begin{cases} 1 & \text{if } t_{i-1} = \textit{adjective} \text{ and } t_i = \textit{noun} \\ 0 & \text{otherwise} \end{cases}$$

Features dealing with rare words may also consider the suffix of the current word (w_i):

Example 2.1.7.

$$f(h_i,t_i) = \begin{cases} 1 & \text{if } \textit{suffix}(w_i) =' \textit{ing}' \text{ and } t_i = \textit{verb} \\ 0 & \text{otherwise} \end{cases}$$

The maximum entropy tagger was trained and tested on the Penn Treebank's Wall Street Journal corpus, achieving as much as 96.6% accuracy [19, 20] on a separate test set. The major advantage of maximum entropy in contrast with rule-based methods (such as transformation-based learning, subsection 2.1.1) is that a maximum entropy tagger is able to produce statistical models that associate probabilities to each tagging decision they make. This allows for a measure of confidence which can be further leveraged in higher-level applications.

2.2 Parsing

Full syntactic parsing of a sentence involves generating a parse tree from the *constituents* of a sentence [10]. Constituents are meaningful groups of words, formed from sub-constituents such as part-of-speech tags and phrasal groups. Probabilistic constituency parsing attempts to generate a tree that represents the most likely intended parse of a given sentence. For example, the first sentence from the sample corpus in Example 2.0.1 is expressible by the following parse tree:

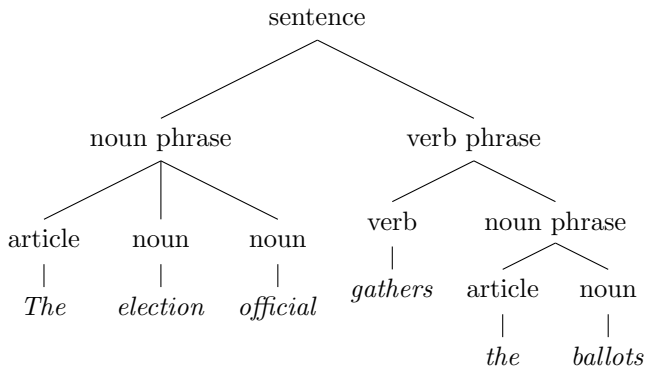


Figure 2.1: An example parse tree.

The root of the parse tree (*sentence*) is comprised of sub-constituents *noun phrase* and *verb phrase*; each containing further sub-constituents, such as *article*, *noun*, and *verb* [1]

Statistical approaches comprise some of the leading techniques towards parsing in natural language processing [21, 22]. One approach is probabilistic context-free grammar (PCFG) [22], which learns probabilities of *phrase-structure rules* [23, 21] from an annotated corpus. Another approach, *maximum entropy* parsing, follows similarly from the concepts described in subsection 2.1.2, applied to the task of parsing [20].

2.2.1 Probabilistic Context-Free Grammar (PCFG)

A probabilistic context-free grammar (PCFG) expresses the valid formation of a sentence using *phrase-structure rules*, such in Example 2.2.2; each rule is associated with a probability learned from the corpus

[1] These tag labels serve as intuitive examples; the actual set of tag emitted by taggers and parsers using the Penn Treebank can be found in [16].

[15]. A parse tree is formed by applying the various grammar rules in an attempt to form a sentence, and the preferred tree is the one which yields highest probability [21].

Phrase-structure rules in a PCFG are assigned probabilities based on how frequently a rule appears in the annotated corpus. Rules may take on a form such as:

Example 2.2.1.

$$A \rightarrow B\ C$$

where the constituent A is said to be formed from sub-constituents B and C. Example 2.2.2 specifies a grammar for forming a sentence S; specifically, how to form a *noun phrase* (NP) from a *determiner* (DT) and *noun* (N); a *verb phrase* (VP) from a *verb* (V) and *noun phrase*; and a *sentence* (S) from a *noun phrase* and *verb phrase*.

Example 2.2.2.

$$NP \rightarrow DT\ N$$
$$VP \rightarrow V\ NP$$
$$S \rightarrow NP\ VP$$

A PCFG is a collection of these rules, each having an associated probability. For example:

Example 2.2.3.

$NP \rightarrow DT\ N$	0.75
$VP \rightarrow V\ NP$	0.68
$S \rightarrow NP\ VP$	0.42

The simple probabilistic grammar defined in Example 2.2.3 allows a parse tree for a sentence to be constructed from a determiner, noun, and verb. Each rule has an associated probability and the probability of the entire tree T, derived from a sentence S, is given by the product of probabilities of all rules used in forming T [10]:

$$p(T, S) = \prod_{rule_i \in rules(T)} p(rule_i) \tag{2.3}$$

Consider the following example sentence extended from Example 2.0.1, annotated with part-of-speech tags below each word:

Example 2.2.4.

The	official	gathers	the	ballots	.
DT	*N*	*V*	*DT*	*N*	

Using the simple grammar defined in Example 2.2.3, a valid parse tree for Example 2.2.4, along with the associated probabilities for each rule are given in Figure 2.2.

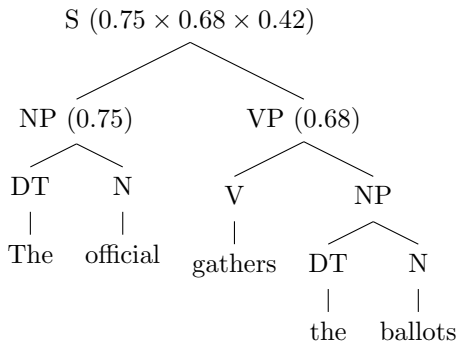


Figure 2.2: A PCFG parse tree with associated probabilities.

The probability calculations for this parse tree are shown at each constituent node (tag probabilities are omitted for simplicity). In this simple example, the probability for the entire tree $p(T, S)$ is 0.21. In cases where several valid parses exist, the tree with highest probability is selected.

Charniak describes an implementation of a PCFG that ranks parses for sentences based on two probabilities: the probability of a given word being the *head* of a constituent, and the probability associated with the phrase-structure rule for that constituent [21]. The *head* refers to the central word that dictates the type of the constituent, and the head of the entire sentence is determined by the head of its constituents. The head of a noun phrase is considered to be the main noun contained within it; for example, the head of the noun phrase *"the official results"* is the noun *"results"*. Likewise, the head of a verb phrase is the main verb, such as the verb *'tallied'* in the verb phrase *"are quickly tallied"*. The heads of each constituent propagate upwards and contribute to determining the head word of the entire sentence [21]. Charniak's implementation is able to achieve a recall and precision of 86% on a test corpus of 2,416 sentences [21].

2.2.2 Maximum Entropy

Extending from the concept of maximum entropy applied to tagging, described in subsection 2.1.2, Ratnaparkhi has also implemented parsing via maximum entropy. The training data required to build a *maxent* parser consists of the *actions* taken to build a parse tree [20]. Each action is assigned a probability score, and a series of actions produces a parse tree, thus the resulting score for the tree is based on the probability of the actions required to build it. Parsing actions and the contexts for an action (such as the surrounding tags and sub-constituents) are learned from an annotated corpus and used to define features for the maximum entropy model.

In total, Ratnaparkhi defines four procedures for building the model: *tag*, *chunk*, *build*, and *check*. The input to each procedure is a series of actions (a *derivation*), to which the currently running procedure appends further actions to create a new derivation. The procedures run in passes, such that the output from a previous procedure is available as part of the input for the next. For example, the *chunk* procedure runs after the *tag* procedure, and will have available to it all of the POS tags for each word in the sentence. This information can then be used to define a feature that considers the surrounding context of tags in

order to determine which action to be performed. Example 2.2.5 defines such a feature:

Example 2.2.5.

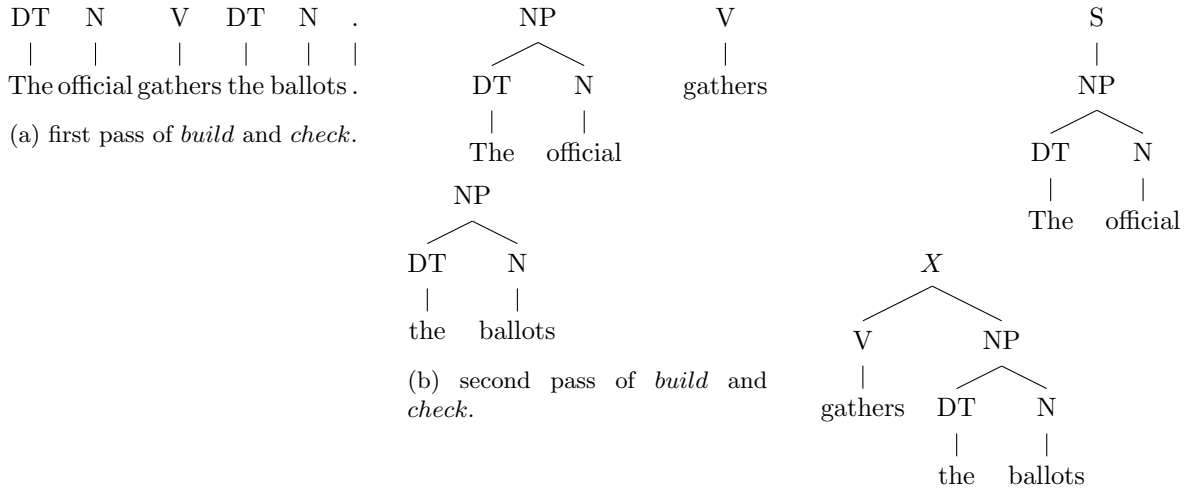
$$f(h,a) = \begin{cases} 1 & \text{if } previousTag(h) = adjective \text{ and} \\ & currentTag(h) = noun \\ & a = StartNounChunk \\ 0 & \text{otherwise} \end{cases}$$

where feature f is active when action a is to start a new *noun chunk*, and when the previous tag in the context of h is an *adjective* and its following tag is a *noun*. Input to the *build* and *check* procedures is a forest of subtrees [20], built around different head words. The following is an example of a feature that may be learned during the *build* and *check* procedures:

Example 2.2.6.

$$f(h,a) = \begin{cases} 1 & \text{if } previousChunk(h) = noun\ chunk \text{ and} \\ & previousHead(h) = \text{'that' and} \\ & a = StartNounPhrase \\ 0 & \text{otherwise} \end{cases}$$

The example feature in 2.2.6 is active when action a is to start a new *noun phrase* constituent, and when the context h contains a preceding noun chunk and a previous subtree with head word *"that"*. As in subsection 2.1.2, the features apply constraints to the maximum entropy model. In order to avoid noise and over-fitting of the training data, features that occur less than five times in the corpus are discarded [20]. Ratnaparkhi's implementation constructs a model in three passes: the first pass runs the *tag* procedure and generates POS tags for an input sentence; the second pass uses the tagged results as input to the *chunk* procedure, where each word is paired with its tag and assigned to a new or preexisting chunk; the third and final pass runs the *build* and *check* procedures iteratively until a complete parse tree is formed [20].



(a) first pass of *build* and *check*.

(b) second pass of *build* and *check*.

(c) third pass of *build* and *check*.

Figure 2.3: Trees produced after each pass of *build* and *check*.

Figure 2.3 shows the trees constructed after one or more passes of the *build* and *check* procedures (using the first sentence from Example 2.0.1), where X denotes the constituent currently being considered for action. Each parse tree is scored based on the product of the probabilities associated with the actions taken to build it, and the k-best trees are yielded by the parser (Ratnaparkhi uses $k = 20$). A score is assigned to each tree T by the following function:

$$score(T) = \prod_{a_i \in derivation(T)} p(a_i|h_i) \tag{2.4}$$

where the score of tree T is the product of the conditional probabilities of the actions a_i used to form T, given their contexts h_i.

The Wall Street Journal corpus was used to train and test the implementation, with as much as 40,000 sentences used for training and 2,416 sentences for testing [20]. The *maxent* parser is able to achieve a precision of 87.5% and recall of 86.3%.

2.3 Grammar Verification

Perhaps one of the most familiar and commonplace grammar and style checking applications can be found in the *Microsoft Word* document-processing software [24, 4]. It leverages heuristic and statistical methods for natural language processing [4]; however it's implementation is propriety, and its performance is notorious for its dubiety [3]. Increasingly, there are now several proprietary online grammar checking systems as well, such as *WhiteSmoke* [25] and *Ginger* [26]. While the operations of these systems are interesting, their lack of transparency makes them difficult to examine further, thus this section limits discussion to non-proprietary research and approaches in grammar verification. *LangaugeTool*,

an English-language grammar checker [5], and *GRANSKA*, a Swedish language grammar checker [6] are examples of grammar verification based on heuristics. A more recent and adaptive approach to the problem of grammatical verification is to apply machine-learning techniques to model and predict grammatical errors. Two such approaches are discussed towards the end of this chapter.

2.3.1 Heuristic approaches

LanguageTool

LanguageTool [5] is an open-source rule-based grammar and style checker for the English language. The core components of LanguageTool are the rule matching system and the rule base. The author suggests a method of developing rules for the rule base by considering a grammatical error and reducing it to its general form, expressed in its parts of speech; the rule should then be verified to ensure it does not produce an excessive amount of false positives. Expanding on Example 2.0.1, a typical grammatical error might manifest itself in the form of a *number disagreement* as in Example 2.3.1.

Example 2.3.1.

The official results *is* quickly tallied and a *winners* is announced.

The error is underlined, where the plurality of the verb *is* (singular) does not agree with the plurality of the subject *results* (plural). The corresponding POS tags [2] [3] are as follows:

Example 2.3.2.

The	official	results	*is*	quickly	tallied	.
DT	*JJ*	*NNS*	*VBZ*	*RB*	*VBN*	

The error can be generalized in terms of tags as the occurrence of a plural noun (*NNS*) directly preceding a singular verb (*VBZ*); this can be expressed as a rule in LangaugeTool's XML syntax:

Example 2.3.3.

```
<rule id="NO_AGREE" name="Agreement error">
   <pattern mark_from="0" mark_to="0">
      <token postag="NNS"></token>
      <token postag="VBZ"></token>
   </pattern>
   <message>
      Number agreement error — use plural verb or singular noun:
   </message>
</rule>
```

[2] expressed using tags from the Penn Treebank tag set [16]

[3] LanguageTool has an embedded POS tagger (based on Brill's tagger (subsection 2.1.1) and Qtag [5]) as well as phrase chunker.

The rule defined in Example 2.3.3 matches when an *NNS* tag is followed by a *VBZ* tag – a *number disagreement* error that occurs when a plural noun directly precedes a singular verb. Rules can include regular expression pattern matching, along with feedback messages and corrective suggestions. The author suggests that a system of hand-crafted rules with explanation and correction mechanics is preferred over probabilistic methods due to rules being able to offer feedback to the user. The system is also said to be language-agnostic since the tagger and rule base can be adapted to meet the needs of other languages [5].

In tests on a small corpus of collected errors, LanguageTool was compared with *Microsoft Word 2000*'s grammar checker, detecting 42 errors in comparison to 49 detected by Microsoft Word [5].

GRANSKA

GRANSKA is a grammar checker that targets specific errors common to the Swedish language [6] (although it can be adapted to other languages). Particular focus is given to three types of errors: *split compounds* (similar to compound nouns in English), *noun-phrase disagreement* (e.g. in number, tense, or gender), and *subject-predicative disagreement* (improper use of adjectives).

The structure of the system consists of a tokenizer and part-of-speech tagger at the lower layers, which supply input to a grammatical rule matcher and spelling checker [6]. The POS tagger is implemented using hidden Markov models on *trigrams* – tag sequences of length three [6]. The tagger is able to accurately tag 97% of known words and 93% of unknown words with a tag dictionary containing 140 parts of speech. Grammatical verification occurs by matching rules in its rule base against the sequence of generated tags. The strategy for selecting rules to match is done in an efficient manner: if a rule condition is determined to be statistically rare, then that rule need only be checked when the rarest of its conditions occurs. This avoids needlessly checking rules at each partial match.

Example 2.3.4.

```
cong22@incongruence {
    X(wordcl=dt),
    Y(wordcl=jj)*,
    Z(wordcl=nn & (gender!=X.gender | num!=X.num | spec!=X.spec))
-->
    mark(X Y Z)
    corr(X.form(gender:=Z.gender, num:=Z.num, spec:=Z.spec))
    info("The␣determiner" X.text "does␣not␣agree␣with␣the␣noun" Z.
        text)
    action(scrutinizing)
}
```

The rule description language is reminiscent of C-based languages, and includes a matching section (the condition that causes a positive match) followed by a corrective section (actions to take once the

condition has been matched). The rule in Example 2.3.4 matches in the case where a determiner (followed by one or more adjectives) does not agree in gender, number, or species with its following noun. The corrective action (preceded by →) is to mark the determiner and noun, and any adjectives, as an error, and to offer an explanation for the error. Rules such as 2.3.4 can be matched directly by the rule-matcher or included within other rules, even recursively [6]. Matching is performed from left to right, and any corrections are also matched against all other rules to ensure the corrected sentence does not introduce new errors.

The performance of *GRANSKA* was tested by the authors on various corpora and compared with other grammar checking systems. On a test set containing 418 grammatical errors and spanning news, science, and essays, it achieved an overall 53% precision and 52% recall. In comparison to *Microsoft Word 2000*'s Swedish grammar checker on the *Svante* corpus containing 32,452 words, *GRANSKA* performed at 82% precision and 63% recall, while Microsoft Word's performance was 47% and 66%, respectively [6].

2.3.2 Machine-learning approaches

Grammatical error prediction

During the course of this research, a related doctoral thesis entitled *Grammatical error prediction* [1] was published by the University of Cambridge. Of particular interest is Andersen's focus on a machine-learning approach, applied to the binary classification of a sentence as grammatically correct or incorrect. Three principle tasks are required by the machine-learning approach: evidence gathering, feature extraction, and classification [1].

The Cambridge Learner Corpus (CLC) [27] is used as *evidence* data in Andersen's research; it is an error-annotated *learner* corpus [4] that contains both grammatically correct and incorrect sentences. The annotation scheme encodes various types of errors and their corrected forms.

Andersen defines a set of ten features to be extracted from each sentence, listed in Table 2.1.

Feature	**Examples**
Word	$a; thought; occur + ed$
Word bigram	$athought; thoughtoccur + ed$
Part of speech	DT; NN; VBD
Part-of-speech bigram	DT NN; NN VBD
Part-of-speech trigram	DT NN VBD
Grammatic relation (words)	$thought \rightarrow a; occur + ed \rightarrow thought$
Grammatic relation (parts of speech)	NNI → ATI; VVD → NNI
Parsing error (binary)	parsing error
Sentence fragments (binary)	fragment
Sentence fragments (clause type)	NP fragment; PP fragment

Table 2.1: Sentence features defined in [1][5]

[4] A *learner* corpus primarily contains works by authors learning the language as a second-language [12].
[5] Tags have been adapted to the Penn Treebank tag set.

The *Word* and *Word bigram* features capture information about the words used in a sentence; the *Part of speech* features capture the grammatical categories of these words in the sentence; the *Grammatic relation* features represent the grammatical relations between words, and parts of speech, in a sentence; the *Parsing error* feature indicates whether a complete parse [6] could be formed for the sentence; the *Sentence fragments* features indicate if the parse was generated using a fragment rule and if so, the types of the fragments involved [1].

The features extracted from the sentences are used as training data for various machine-learning algorithms: *Naive Bayes*, *Balanced Winnow*, *Maximum Entropy*, and *Support-Vector Machines (SVM)*. Andersen was able to achieve a classification accuracy of above 70% on separate test data; the highest accuracy achieved was 71.27%, using the *Maximum Entropy* classifier.

Classifier	Training data	Test data
Naïve Bayes	80.36%	70.39%
Balanced winnow	95.25%	69.76%
Maximum entropy	79.41%	71.27%
Support vector machine	73.65%	70.88%

Table 2.2: Classification results from [1]

However, detection rates greatly depend on the types and number of errors present in a sentence, as seen in the experimental results shown in Figure 2.4. Spelling errors and inflection errors (such as improper pluralization) result in the highest detection accuracies, between 77% and 79%; by contrast, errors resulting from improper verb conjugation or omitted words can be much more difficult to detect, with accuracies of 58% to 62%. Andersen notes that sentences containing such errors may still remain syntactically correct, despite their unnaturalness to a native speaker.

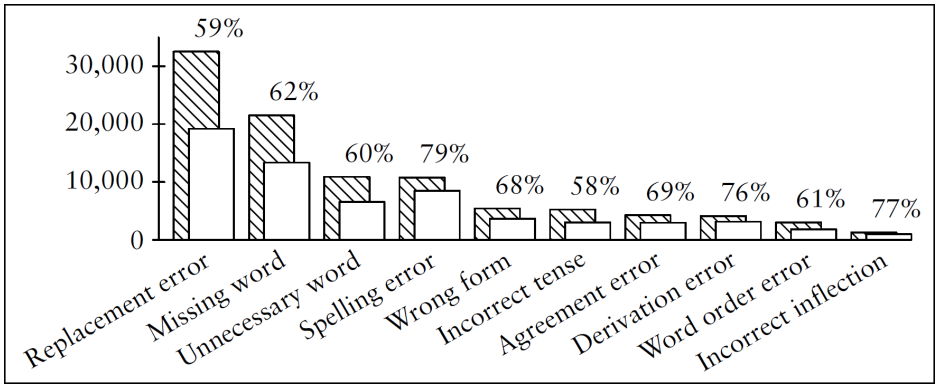


Figure 2.4: Classification accuracy by error type [1]

A particular limitation is the lesser performance of the system on conjugation errors, such as errors in the tense of a verb:

[6] Parsing is performed by an extended version of *RASP*, called *RASP4UIMA*, implemented in Andersen's research [1]

> *"As one might expect, the system is more successful at handling morphological and typographical errors than syntactic ones, and the lowest success rate is observed for verb tense errors, which are likely to depend on extra-sentential context."* [1]

Increasing the overall number of errors present in a sentence greatly increases the likelihood that any error is detected; a single error is detected in only about 50% of occurrences, and as much as seven errors may be detected in 80% of occurrences. Figure 2.5 depicts the effect of error quantity on error detection.

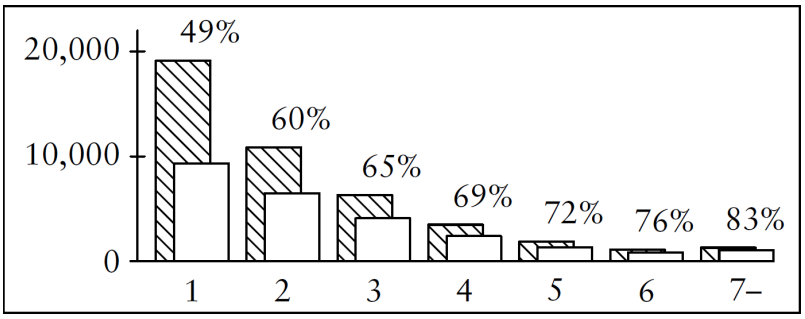


Figure 2.5: Classification accuracy by error quantity [1]

Andersen's thesis describes ways of achieving a higher level of control over the presence and types of errors – *GenERRate* is one such method mentioned by the research, a tool implemented to synthesize errors such as omissions, insertions, and inflection errors. However, the tool does not appear to be publicly available.

The results of Andersen's grammatical binary classification of sentences appear to corroborate that a machine-learning approach is at least viable in predicting grammatical errors, provided that there is a good set of examples for both grammatical correctness and incorrectness. Andersen notes that the set of sentence features defined is not exhaustive [1], and that further experimentation may seek to define more and better features.

Attempts to verify grammar

Prior to the appearance of Andersen's work, we published a similar machine-learning approach to grammatical verification via sentence classification [14]. In contrast to Andersen's approach (using *bigrams* and *trigrams* to define features), we defined a feature set derived from statistical natural language processing techniques, such as the probabilities associated with POS tags and parse trees:

> *"(i) score obtained from both parsers..., (ii) parse scores from both parsers, (iii) POS tag sequence probability, (iv) first POS tag probability, (v) lowest POS tag probability, (vi) highest POS tag probability, (vii) sentence length, (viii) sentence clausality, from both parsers."* [14]

Example data for the machine-learning process was extracted from the Open American National Corpus (OANC), which is a collection of written English text, spanning several domains of writing such

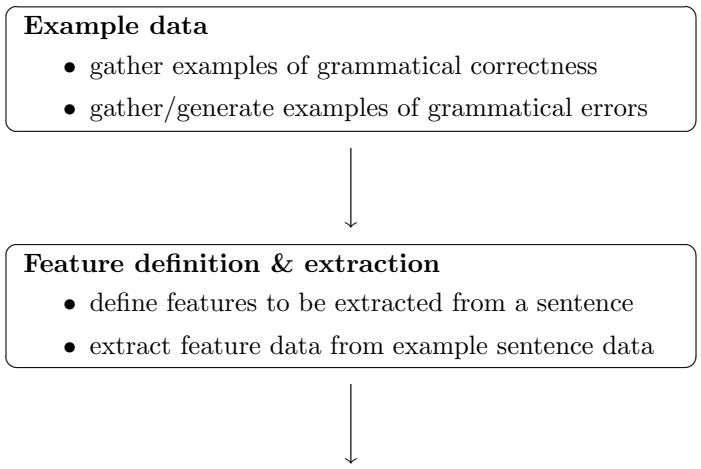
as articles, magazines, travel guides, non-fiction and fictional works [28, 29]. Examples of grammatical incorrectness were procedurally synthesized from this corpus by injecting errors into the original sentences, specifically *displacement errors*, which occur when certain words are misplaced within a sentence. The approach we take to grammatical verification in this thesis extends the earlier research, especially that of [14], and is described in detail in the coming chapters.

Chapter 3

Methodology

A major objective for this research is the prediction of grammatical errors and grammatical correctness using statistical natural language processing and machine-learning techniques. Succinctly, machine-learning encompasses algorithms for pattern recognition and representation [9]; it is a sub-domain of artificial intelligence, and as described in **??**, it leverages example data in order to learn and make predictions about the phenomenon of interest. Patterns and concepts *learned* from the example data are expressed by a *model*, which is capable of making predictions about future, unknown data.

In the context of this research, the primary concept of interest is grammatical correctness – determining whether a sentence is grammatically correct or incorrect. This chapter discusses three important methods in support of the machine-learning process: (1) gathering example data in the form of correct and incorrect sentences; (2) defining and extracting features from a sentence; and (3) performing machine-learning classification on the extracted feature data, using various algorithms. These major phases are summarized in Figure 3.1.



Classification

- apply classification algorithms to feature data
- use models to predict grammatical correctness and presence of errors in a sentence

Figure 3.1: A machine-learning approach to grammar verification.

To bolster the machine-learning process, a wealth of example data is a desirable requisite – as much data should be collected as possible and feasible that represents the concept to be modeled. For the concept of grammatical correctness, an appropriate data set is comprised of sentences that exhibit grammatical correctness and grammatical incorrectness (manifesting one or more errors). Example 3.0.5 is a trivial (and insufficient) data set constructed from just two sentences, adapted from Example 2.0.1 and Example 2.3.1 (grammatically correct and incorrect, respectively).

Example 3.0.5.

The official results are quickly tallied and a winner is announced.
The official results *is* quickly tallied and a *winners* is announced.

There exist many corpora of overwhelmingly correct sentences, such as the corpora mentioned in the beginning of Chapter 2; however the same cannot be said for grammatically incorrect sentences – where they exist, they are either from non-public sources (such as the CLC) [30, 31], or tend to be interspersed amongst correct sentences, making their extraction a difficult task. The novel approach taken in this research is to *generate* examples of grammatical incorrectness by synthesizing errors in preexisting sentences that are assumed to be correct. This approach to gathering example data is explained further in Section 3.2.

Another important requisite for the machine-learning process is a set of defined *features* (or *attributes*) that represents interesting characteristics about the data [9]. Features may be defined over a sentence as a function of the underlying data, such as the words, tags, or parse tree. This research defines over 150 features for a sentence, derived from natural language processing techniques. The methods for extracting feature values are discussed in Section 3.3.

Machine-learning algorithms operate on feature data (or attribute values), extracted from example data [9]. Concepts and patterns learned from the data are encapsulated in a *model*, in a representation dependent on the algorithm (for example, a set of rules, or a probability distribution). Machine-learning classification is used to generate models capable of predicting grammatical correctness, and the presence of grammatical errors in a written sentence; this phase is described further in Section 3.3.

3.1 Common NLP tasks

Described in Chapter 2, various NLP techniques exist for analyzing sentences, these include: tokenization, part-of-speech tagging, and parsing. This section briefly describes the implementations of these

techniques that are used within this research, especially those provided by the *OpenNLP* [32] and *Stanford NLP* [33, 22, 34] libraries.

3.1.1 Tokenization

The process of tokenization involves transforming a stream of symbols (alpha-numeric characters, whitespace, and punctuation) into a stream of delimited tokens consisting of one or more lexical elements, such as words. A common preprocessing step in tokenization is *sentence splitting*, which splits the token stream along sentence boundaries. Sentence splitting and tokenization are performed by both the OpenNLP and Stanford NLP libraries, implemented for the *Java* programming language [35]. Generally, and where otherwise unspecified in this research, tokenization is performed by the Stanford NLP library.

3.1.2 Part-of-Speech Tagging

POS tagging assigns each word a grammatical function in the sentence (e.g. *noun*, *verb*, *adverb*, etc). Probabilistic tagging further assigns probabilities to each tag, indicating the likelihood of the tag given the context. Implementations of *maximum entropy* taggers (described in subsection 2.1.2) are provided by both OpenNLP and Stanford NLP; both have been trained on the Penn Treebank Wall Street Journal corpus. However, only the former emits probabilities for its tagging decisions, and thus where tag probabilities are required, they are generated by OpenNLP.

3.1.3 Parsing

The parsing process constructs a syntax tree from the words in a sentence. Each word is tagged and then parsed into a hierarchy of constituents. Probabilistic parsing (described in Section 2.2) associates a probability with the parse tree and its nodes. Both OpenNLP and *Stanford NLP* offer constituency parsers that emit parse probabilities [1]. The former provides an additional metric, referred to as the *tag sequence probability*, which is the probability of the sequence of POS tags assigned by the parser [32]. OpenNLP implements *maximum entropy* parsing, whereas Stanford NLP performs parsing uses probabilistic context-free grammar (PCFG). Both parsers provide models that have been trained on the Penn Treebank Wall Street Journal corpus.

3.1.4 Natural Language Generation (NLG)

Another important strategy used in this research involves generating new sentences from preexisting sentences by modifying or inserting words, such as altering their inflection or conjugation. An entire subfield of NLP exists for such tasks, referred to as natural language generation (NLG) [36]. The implementation used in this research is *SimpleNLG* [37]; it is used wherever the lexical structure of certain words needs to be modified, or new forms substituted, such as altering the tense of a verb or the plurality of a noun.

[1] given in log-product probabilities.

3.2 Collecting & generating example data

Modeling the concept of grammatical correctness requires sufficiently many positive and negative examples of it; respectively, correct and incorrect sentences. Thus this research first sets about the task of gathering examples of grammatically correct sentences, discussed in (subsection 3.2.1, and generating examples of grammatically incorrect sentences, discussed in (subsection 3.2.1.

3.2.1 Example data for grammatical correctness

As discussed in Chapter 2, there exist several well-known English *corpus*, such as the *Wall Street Journal* corpus [38], the *British National Corpus (BNC)* [39], the *American National Corpus (ANC)* [28, 40], the *Brown Corpus* [41], and others [16]. They consist of writings from multiple domains of English, including published articles, news wires, reports, stories, and other publications [28]. Providing that the published writings are largely free of grammatical errors, these collections serve as convenient sources of example data. While this does incur some degree of noise [2], we deem it acceptable, and moreover, the only systematic and automated way of obtaining example data for grammatical correctness.

The Open American National Corpus (OANC) [29], and the Manually Annotated Sub-Corpus (MASC) [42, 43, 44] are used in this research as sources of correct sentences. Both corpora are freely-available derivatives of the American National Corpus (ANC) [28, 40]; they are comprised of published works by native English speakers and span several domains of writing, including fictional and non-fictional works, newspaper articles, blogs, technical papers, travel guides, magazines, etc). The MASC is a manually-annotated, 500,000-word subset of the OANC, which itself contains over 15 million words. We constrain these corpora to a subset that is representative of the most common uses of English, such as newspaper and magazine articles. A sample set of the corpora used in this research can be found in Appendix 2; the full textual data can be retrieved from [13].

3.2.2 Example data for grammatical incorrectness

Obtaining sources of purely incorrect sentences proves difficult: where these sources are isolated (or at least isolatable), they tend to be rare and non-public (as is the case with the CLC); where grammatically incorrect sentences are publicly abundant, they tend to be interspersed amongst grammatically correct sentences, presenting the problem of extraction. For example, the International Corpus of Learner English (ICLE) [12] features collected writings by second-language learners of English, which is expected to contain a much higher incidence of grammatical errors. Another example is the source of grammatical errors to be found in unmoderated comments on public forums or social media websites. However, in both examples the challenge then becomes extracting the incorrect sentences from the likely many grammatically correct sentences, and filtering by hand is typically infeasible for large corpora.

[2] A sample of 100 sentences taken from the *Verbatim* sub-corpus of the OANC [29] yields above 80% (±10%) grammatically correct sentences (found in Appendix 2). Textual fragments such as titles, headings, figures, listings, dates, serve to form part of this noise; and though not treated in this research, it may be possible to reduce this noise to some degree through filtering and preprocessing techniques.

Generating grammatical errors

A more feasible approach to obtaining grammatically incorrect sentences is to *generate* them. Using natural language generation (NLG), sentences can be constructed with grammatical errors systematically injected into them, thus producing grammatically incorrect sentences. While it is possible to use NLG to construct malformed sentences, it still requires some semantic context be provided (e.g. *subjects*, *objects*, *actions*). Rather than generate arbitrary prose, it is more convenient to modify preexisting sentences so that they become incorrect; this also has the advantage of retaining the semantics and naturalness of the original sentence, as much as possible. As alluded to in Example 1.0.1, ambiguity can lead to situations in which errors in the intention of a sentence may not necessarily manifest as errors in its syntax; however the impact of this type of noise is minimal [3]. Most importantly, a systematic method of generating errors provides a degree of control to the experimental design. An implementation of this method is discussed further in Section 3.2.3.

Grammatically parallel corpora (GPC)

Generating grammatically incorrect sentences from pre-existing natural sentences is not only convenient and feasible, but also especially useful for isolating the concept of grammatical incorrectness. Sentences which have been modified to contain grammatical errors typically retain most of their other linguistic properties — domain and semantics (such as the subjects, objects, and actions) tend to remain similar or unchanged (as is the case in Example 3.0.5). Thus, variations between the original sentences and their error-induced parallels are isolated to the presence of grammatical errors. For example, the pair of sentences in Example 3.0.5, while differing in syntax and validity, still largely retains much of the same meaning. Given a corpus of grammatically correct sentences, it is possible to generate a *parallel* corpus of grammatically incorrect sentences, similar in most ways except where it exhibits grammatical errors. In this research, we refer to a pair of corpora constructed in this way as grammatically parallel corpora (GPC) [4] . By modifying sentences from an existing corpus rather than artificially creating new sentences, the resulting corpus generally retains much of the same words, meaning, context, and domain as the original, thereby eliminating variations that may not necessary be useful to the concept of grammatical correctness. Conceptually, a grammatically parallel corpus is the juxtaposition of grammatically correct sentences with the ways in which these sentences may become grammatically incorrect.

3.2.3 CorpusTools

Implementations of the methods discussed in this section are provided by *CorpusTools* [13] – software developed during the course of this research to construct and preprocess corpora, especially grammatically parallel corpora. Provided a source of grammatically correct sentences, it offers an automated, systematic method of generating various types of grammatical errors, specifiable in type, number, and

[3] A sample of 20 error sentences generated from the *Verbatim* sub-corpus of the OANC [29] appears to contain purely grammatically incorrect sentences (found in Appendix 3).

[4] The usage of the term *parallel* here borrows from a similar use – *parallel corpora* – when referring to a corpus that has been translated across multiple languages.

context. CorpusTools is implemented in the Java programming language [35], and relies on natural language generation (provided by *SimpleNLG* [37]) to generate various types of grammatical errors in a sentence. The sections that follow describe the types of grammatical errors that can be generated.

Error Types

insertion	insertion of a duplicate token into the sentence
omission	removal of a token from the sentence
displacement	interchanging of positions of adjacent tokens in a sentence
number	alteration of the plurality of nouns, pronouns, and determiners
tense	alteration of the tense (future, past, present) or form of a verb
objectivity	alteration of the objective or subjective use of a pronoun
case	alteration of the capitalization of a noun

Each type of error is constrained to a set of word types to which it may be validly applied.CorpusTools distinguishes between the following POS categories: *noun*, *verb*, *determiner* (or *article*), *adjective*, *adverb*, *preposition*, *pronoun*, *punctuation*, and *"other"* (referring to any part of speech not falling into the previous categories). These categories are in congruence with the POS types mentioned in Chapter 2; they align closely and inter-operate with the POS types recognized by SimpleNLG, OpenNLP, and Stanford NLP. Each type of error, along with its default set of POS types, is described following.

insertion

applies to: all (noun, verb, determiner, adjective, adverb, preposition, pronoun, punctuation, other)

Insertion errors are applied to a sentence by selecting a token (word or punctuation) at random from the original sentence and re-inserting a duplicate of the word at a random location within the sentence. Example 3.2.1 shows a manifestation of this error.

Example 3.2.1.

> The official results are quickly tallied and a winner is announced.
> The official results are quickly tallied and a winner is <u>are</u> announced.

This error is intended to represent grammatical errors that occur from superfluous, or in general, inappropriate use of a word in a sentence. In cases where the duplicate token is re-inserted adjacent to its original location (as in Example 3.2.2), this may represent errors that occur from unintended repetition of a word.

Example 3.2.2.

> The official results are quickly tallied and a winner is announced.
> The official results are <u>are</u> quickly tallied and a winner is announced.

omission
applies to: all (noun, verb, determiner, adjective, adverb, preposition, pronoun, punctuation, other)
Omission errors are created within a sentence by removing a token at random.

Example 3.2.3.

> The official results are quickly tallied and a winner is announced.
> The official results __ quickly tallied and a winner is announced.

The intention of omission errors is to represent common grammatical errors that occur when words or punctuation have been erroneously omitted from a sentence. The effects of this error may be less useful where the omitted token is a punctuation mark, or in rarer cases, where the sentence contains an insufficient number of tokens to be considered meaningful.

displacement
applies to: all (noun, verb, determiner, adjective, adverb, preposition, pronoun, punctuation, other)
Displacement errors are generated in a sentence by selecting a random token and exchanging its position with an adjacent token. This represents grammatical errors that may occur from the mis-ordering of proximate words in a sentence. In the general case where the selected token is adjacent to two tokens, the preceding or following token is chosen at random.

Example 3.2.4.

> The official results are quickly tallied and a winner is announced.
> The official results are quickly tallied *a and* winner is announced.

number
applies to: noun, determiner, pronoun
Number errors are created by altering the plurality of words. The types of words valid for this error are constrained by default to the set of POS types that convey grammatical information about *quantity* – nouns, determiners, and pronouns. Applying a number error to a word involves changing its inflection, or replacing it with a word of the same grammatical function, such that it posses a different plurality than its original form. The intended result is a change to the implied quantity associated with the word and any other word that is grammatically linked with it. Generally, number errors represent a type of *agreement* error, where words carrying quantitative information are inconsistent about the quantities that they imply [45]. Number disagreement can be created among determiners, nouns, and pronouns; subject-verb disagreement [11] is created between the altered words and any associated verbs. Number errors apply (by default) to nouns, pronouns, or determiners. The application of the error to each POS type is described below.

noun Number errors applied to nouns will attempt to alter the plurality of the noun. A noun in singular form is inflected to its plural form; likewise, a noun in plural form is inflected to its singular

form. In Example 3.2.5, the word *"results"* is a plural noun; altering its inflection to *"result"* retains its function and meaning, but changes the quantity to which it refers.

Example 3.2.5.

> The official results are quickly tallied and a winner is announced.
> The official *result* are quickly tallied and a winner is announced.

determiner Number errors applied to determiners and articles modify their implied quantity. The affected article or determiner is replaced with one of the same function, but implying a different plurality. A determiner that implies a single entity is replaced with one that implies multiple, and vice-versa. In Example 3.2.6, the article *"The"* is replaced with the article *"A"*.

Example 3.2.6.

> The official results are quickly tallied and a winner is announced.
> *A* official results are quickly tallied and a winner is announced.

pronoun Number errors applied to pronouns modify or replace the pronoun with one of opposite plurality. In Example 3.2.7, the word *"They"* is a plural pronoun that is replaced with the singular pronoun *"He"*, creating an agreement error with the verb *"mark"*.

Example 3.2.7.

> They mark the ballots with their top three choices.
> *He* mark the ballots with their top three choices.

tense
applies to: verb
Tense errors affect verbs. A verb selected for a tense error will be conjugated to a form that is likely to be grammatically incorrect given its context. For example, a verb that is originally in *past-tense* form will be conjugated to its *present-tense* or *infinitive* (base) form. This type of error represents grammatical errors that result from the improper conjugation of verbs, often leading to subject-verb disagreement, specifically in tense, number, or person [11]. Example 3.2.8 shows a tense error applied to two verbs in the sentence.

Example 3.2.8.

> The official results are quickly tallied and a winner is announced.
> The official results *being* quickly tallied and a winner *be* announced.

objectivity

applies to: pronoun

Objectivity errors operate on pronouns, and invert their function as the subject or object of the sentence, if they are of a personal type (*"I", "me", "we", "us"*, etc.). A pronoun that is personally *subjective* will be inflected to its *objective* form (e.g. *"I"* becomes *"me"*), and similarly an objective personal pronoun becomes subjective (e.g. *"us"* becomes *"we"*). Objectivity errors model the improper use of personal pronouns, especially where it results from confusion about the role of *subject* and *object* in a sentence. Example 3.2.9 shows commonly mistaken noun phrases in which uses of the personal pronoun *"me"* occur in place of *"I"*, *"him"* in place of *"he"*, or *"her"* in place of *"she"*:

Example 3.2.9.

> They and I mark the ballots with our top three choices.
> He and she mark the ballots with their top three choices.
> They and *me* mark the ballots with our top three choices.
> *Him* and *her* mark the ballots with their top three choices.

case

applies to: noun

Case errors alter the capitalization of nouns. Applying this error to a selected noun inverses the capitalization of its first character: a noun that initially begins with a lowercase character will have its first letter capitalized, and vice-versa. These errors may occur from mis-capitalization of proper nouns or regular nouns, or from improper capitalization at the beginning of a sentence. The impact of capitalization on grammar and perceived correctness is often subtle or inappreciable, thus case errors are not considered further in this research.

Error selection & application

CorpusTools tokenizes, tags, and analyzes each input sentence for ways in which it may be altered by the aforementioned error types. For each input sentence, a set of applicable error types is constructed based on the types of words present in the sentence (although additional constraints can be specified). In the case where an input sentence is immutable because it lacks applicable tokens, then the sentence is ignored and does not appear in the error-generated corpus. Otherwise, in the general case, one or more error types will be selected and applied to the tokens in a sentence.

CorpusTools associates each error type with a specifiable *weight* – an arbitrary positive integer that implies the desired frequency [5] with which the associated error should appear in the generated error corpus. Errors are then selected via *weight-proportional probability* (or *probability proportional to size (PPS)*) [46]: the probability of selecting an error type e is proportional to its weight w_e, in proportion

[5] The weights merely imply the *desired* frequencies of error types in the resulting corpus; they do not statistically guarantee them. This is due to two reasons: (1) the pseudo-random nature of the error selection process; and (2) the possibility that a generated error may not result in a valid change to the sentence and is discarded. A constant seed can be specified for further control over the determination of the pseudo-random process (defaulting to 1).

to all other error weights that are valid for a sentence:

$$p_e = \frac{w_e}{\sum_{i=1}^{N} w_j} \tag{3.1}$$

where p_e is the probability of selection for error e; w_e is the weight assigned to error i; and N is the total number of applicable errors for the sentence. Trivially, an error with an assigned weight of zero will not participate in the error generation process.

Once an error type has been selected for a given input sentence, a set of valid tokens (to which the selected error is applicable) is constructed from the sentence. A token is selected at random from this set, and the selected error type is applied to the token in the manner prescribed in Section 3.2.3. The token is then marked as *consumed* by the error generation process and is not available for future selection (in the case where more than one error is being applied, described later in this section) [6] . The process of error selection and application iterates until a valid error is generated or until there are no more valid, unconsumed tokens remaining. If an error is generated successfully, the resulting sentence is retained and made available for further error generation, or it is output to the resulting corpus.

CorpusTools is capable of generating multiple errors per sentence. In this scenario, the error generation algorithm iterates until a specified number of errors has been generated, or until there are no more valid, unconsumed tokens remaining in the sentence. It is further possible to specify that only *exactly* the specified number of errors be generated (and no less); this ensures that input sentences which do not successfully generate exactly the specified number of errors are not included in the resulting output corpus. The error generation procedure is summarized by algorithm 1.

Data: sentences, error types, error quantity
Result: sentences containing errors of the specified types and quantity
foreach *sentence s* **do**
  $tokens \leftarrow tokenize(s)$;
  $tags \leftarrow tag(tokens)$;
  $n \leftarrow 0$;
  while $n <$ *error quantity and* $\exists t \in tokens, \neg consumed(t)$ **do**
    $errors \leftarrow \{e \in$ *error types* $\mid \exists t \in tokens, e$ applies to *tag(t)*$\}$;
    $error \leftarrow selectError(errors)$;
    $token \leftarrow random(\{t \in tokens \mid error$ applies to *tag(t)*$\})$;
    $applyError(error, token, tokens)$;
    $consume(token)$;
    $n \leftarrow n + 1$;
  end
end

Algorithm 1: GenerateErrorSentences

[6] While it is conceivable to have multiple errors applied to a single token, the implementation refrains from this to avoid possibly undoing or reversing any previously applied errors.

The result of the error-generation process, applied to a corpus of correct sentences, is an output corpus of grammatically incorrect sentences, modified with one or more errors, up to (or exactly) the number of errors specified. When taken together with the original corpus, the result is a grammatically parallel corpus.

3.2.4 Corpora

Mentioned in subsection 3.2.1, two freely available English corpora are used throughout this research as sources of grammatically correct sentences: the Open American National Corpus (OANC) [29] and the Manually Annotated Sub-Corpus (MASC) [42, 47, 43, 44]; both are sub-corpora of the American National Corpus (ANC) [28, 40]. Subsets were extracted from each corpora to ensure a manageable size (the full OANC contains over 15 million words), and to limit the domain to commonly-used English (such as found in magazine articles or newspapers). The specific subset corpora used in this research are listed in the following section; the full corpora text can be retrieved from [13].

Open American National Corpus

verbatim
size: 582,384 words
A collection of issues of the *Verbatim* magazine:

> *"Verbatim is a 'magazine of language and linguistics for a person without a Ph.D', containing articles about linguistics and language use. The ANC Second Release contains 32 issues of Verbatim from 1990 to 1996."* [29]

non-fiction
size: 330,524 words
A collection of works of non-fiction:

> *"The OUP sub-corpus contains a quarter million words of non-fiction drawn from five Oxford University Press publications authored by Americans."* [29]

travel guides
size: 1,012,496 words
A collection of travel guides written in English.

> *"Several Berlitz Travel Guides written by and for Americans were contributed by Langensheidt Publishers. The Berlitz sub-corpus is split into separate files by country/city and section."* [29]

Manually Annotated Sub-Corpus

size: 506,768 words
A subset of the OANC, containing various English writings including essays, government documents, journals, blogs, news wires, fiction, non-fiction, and travel-guides. [42, 47, 43, 44]

CorpusTools was used with each of these corpora to generate a collection of grammatically parallel corpora, each exhibiting one or more of the error types defined in Section 3.2.3, in varying quantities. The error corpora are generated in a systematic manner: for each error type e, n different error corpora are generated, where n is the number of errors per sentence. Thus, the total number of error corpora generated from each input corpus is $n \times m$, where m is the number of error types. In this research, $m = 7$ (the seven error types defined in Section 3.2.3), and $n = 5$ (up to five errors per sentence); this results in 35 grammatically parallel corpus generated per input corpus [7] . The procedure for generating error corpora from a collection of input corpora is summarized in algorithm 2.

Data: corpora
Result: corpora containing grammatical errors
foreach *corpus c* **do**
  foreach *error type e* **do**
    foreach *error quantity n* **do**
      GenerateErrorSentences(c,e,n)
    end
  end
end

Algorithm 2: GenerateParallelCorpora

Generating corpora in this systematic manner allows results to be compared and categorized individually by corpus, error type, and number of errors per sentence, as presented in Chapter 4. In total, over one million sentences are used in this research (across four corpora).

3.3 Extracting feature data

Machine-learning algorithms operate on datasets of values defined by *features* [9]. A *feature* (also known as an *attribute*) represents a piece of interesting information about an instance in the example data. The intent in defining a feature is to discover and encode information that may directly or indirectly contribute to learning the concept of interest. The example data in this research consists of sentences; the concept to be learned is grammatical correctness. Thus, a feature captures some information about a sentence that may have an impact on its grammatical correctness. For example, the parse probability of a sentence may constitute a feature; another feature may capture the average probability of the POS tags in the sentence, and perhaps another may record the lowest tag probability and its associated word;

[7] Case errors are not treated extensively in this research.

and so on. A collection of such features forms the *feature set*, and each *instance* in the example data takes on a set of feature values.

A total of 152 sentence features are defined by this research. The values extracted by each feature assume one of two types: *numeric* – an integer or real-valued number; or *nominal* – a predefined value from a set of finite values (such as *true*, *false*, or *noun*, *verb*, *article*). Particular interest is given to defining features that are derived from statistical natural language processing techniques, especially those that associate probabilities with the decisions they make (such as tagging and parsing). Much of the features defined in this research rely extensively on the implementations mentioned in Section 3.1. A comprehensive list of all 152 features is given in Appendix 1.

3.3.1 GrammarTools

The GrammarTools software [13] provides an implementation for feature extraction, along with integration of several other useful NLP tools. It was developed during the course of this research in support of the methods in this section. An important function of GrammarTools is to transform a corpus of input text into a output dataset of feature values, suitable for the machine-learning process [8] . In conjunction with the *Weka* data mining software [48] (described further in Section 3.4), it serves as the primary platform for experimentation in this research. GrammarTools provides the following set of tools:

tag	part-of-speech tagging
chunk	phrase chunking
parse	syntactical parsing
xml	sentence processing and storage in XML format
stats	sentence analysis and statistics
DCG	parsing via definite clause grammar (DCG)
function	invokes a user-defined function on a sentence
dataset	extracts and outputs feature values from sentences
classify	classifies a sentence as grammatically correct or incorrect

tag

The *tag* tool performs part-of-speech tagging, as described in Section 2.1 and subsection 3.1.2.

chunk

The *chunk* tool performs phrase chunking using the OpenNLP chunker. Text is tokenized, and tokens are grouped into noun and verb phrases.

parse

The *parse* tool performs sentence parsing as described in Section 2.2, using the software libraries men-

[8] Feature data generated by GrammarTools can be encoded in either *ARFF* or *CSV* file format; these are particularly well-suited for use with *Weka* [48].

tioned in subsection 3.1.3. Text is parsed into a constituency tree, which is output along with its log-probability.

xml

The *xml* tool performs tokenization, tagging, and parsing of the input text and saves the output of these tasks into XML format [49] for persistent storage. This allows the results of the processing tasks to be reused without needing to regenerate them [9]. Information stored about a sentence includes its tokens, all possible taggings, parses, probabilities, and a designation for grammatical correctness (or number of errors), specified by the user. Example output from this tool can be retrieved from [13].

stats

The *stats* tool is designed to collect interesting statistical information from sentences captured by the *xml* (3.3.1) tool. Information gathered by this tool includes the average number of tokens in a sentence, the average parse probability of a sentence, the number of sentences which contain *clauses* (having a parse in which a properly formed clause is a constituent of the root), along with several other pieces of information. It is also capable of tracking the frequency of specified parse constituents, along with the sub-constituents that comprise them; the result is displayed as a list of production rules for each tracked constituent and its sub-constituents, along with the number of its occurrences.

DCG

The *DCG* tool parses input sentences against a definite clause grammar (DCG) [10]. Production rules are defined for constituents and literals that are allowable in the grammar, and are implemented in Prolog's [50] DCG notation [51], which is suitable for parsing. Each input sentence is tokenized and tagged, and the tags are provided as an argument list to the Prolog engine (an implementation of which is provided by *tuProlog* [52]). The engine then determines if a valid sentence can be formed from the sequence of input tags. Rules may be further constrained by contextual features (such as tense and plurality). The *stats* tool's constituency tracking feature may help to inform the types of rules that may be useful in a grammar. An example grammar (provided by default with the tool) can be retrieved from [13].

function

The *function* tool defines an arbitrary function of a sentence's features; the arguments of the function correspond to the sentence features, mentioned earlier in this section and listed in Appendix 1. The feature function is defined using *JavaScript* [53], and executed using the Rhino [54] script engine.

dataset

The *dataset* tool performs feature extraction as discussed earlier in this section and later (subsection 3.3.2). Each sentence from the input text is processed and its feature values are extracted and

[9] While tasks such as tokenization and tagging complete in often less than a second, parsing may require in some cases as much as 5-10 seconds per sentence.

stored [8]. The user must specify a value for the nominal *class* feature, indicating whether the input sentences are to be considered grammatically correct or incorrect [10] . The feature data is output in *comma-separated values (CSV)* or *Attribute-Relation File Format (ARFF)* [48], for use with the *Weka* data mining software and libraries (as detailed in Section 3.4). The full feature data generated by this tool and used in this research can be retrieved from [13].

classify

The *classify* tool predicts the grammatical correctness of an input sentence using a provided prediction model. The supplied model is typically the product of the machine-learning process performed on feature data generated by the *dataset* tool (3.3.1). For each input sentence, the *classify* tool will extract its feature data, supply it as input to the prediction model, and classify the sentence as grammatically correct or incorrect. The format of the model is expected to be that as generated by *Weka*, which provides *Java* libraries for generating, exporting, and importing prediction models.

3.3.2 Feature extraction

With the necessary tools in place, the next phase in this research is the extraction of feature data using the methods in Section 3.3 from the various corpora generated using the methods in Section 3.2; the resulting feature data may then be used as input to the machine-learning process. As mentioned earlier, GrammarTools provides the *dataset* tool (3.3.1) for the task of extracting feature data from corpora. It implements algorithms and calculations – especially those that involve statistical natural language processing – that operate on a sentence in order to derive the values of each feature. Simple examples of features are the *opennlpParseProb* and *stanfordParseProb* features which record the parse probabilities returned by the *OpenNLP* and *Stanford* parsers (respectively). Another example is the *opennlpDeltaParseProbOmitMin* feature, which records the change in parse probability when the word with minimum tag probability has been removed from the sentence. A more complex example is the *avgDeltaVerbChangeOpennlpParseProb* feature, defined by Equation 3.2, which calculates the average difference (*delta*) in the parse probability of a sentence (as obtained using the *OpenNLP* parser) that results from altering the form of each verb in the sentence such that it yields the highest parse probability for that sentence:

$$\frac{\sum_i^n p(parse(s)) - p(parse(modify(s, v_i)))}{n} \tag{3.2}$$

where n is the number of verbs in the sentence s, and v_i is the verb currently being altered. Consider the following example of an incorrect sentence with improperly conjugated verbs:

Example 3.3.1.

A winner *be* *announces*.

In Example 3.3.1 the verb *"be"* will be altered to the form which yields highest parse probability for the sentence (i.e. *"is"*) and the difference is taken between the parse probability of the original sen-

[10] The *dataset* tool is also capable of handling a numeric class value, indicating the number of errors present, however only the nominal boolean class value is discussed within this research.

tence and its altered form; likewise, *"announces"* is changed to *"announced"* and the difference is taken; the average difference is then calculated and used as the value for the *avgDeltaVerbChangeOpennlpParseProb* feature. This feature, along with the *minDeltaVerbChangeOpennlpParseProb*, *maxDeltaVerbChangeOpennlpParseProb*, *totDeltaVerbChangeOpennlpParseProb* and related other features, may serve to indicate possible errors in verb form: a positive average delta may suggest that better forms (or at least statistically more common forms) exist for one or more verbs within the sentence. The many other features defined in Appendix 1 leverage statistical natural language processing in similar ways to calculate information about a sentence that may be useful to the machine-learning process.

Given the large amounts of data from the various corpora, the process of extracting feature data was expedited by executing it in batch and distributing it across available computing hardware [11]. Time for completion varies depending on the size of the dataset: from several minutes to hours for smaller corpora (such as the MASC), and several days to weeks for larger corpora (such as the OANC). The extracted feature data from each corpus is then post-processed (removing any invalid values such as occurring from parsing failures) and merged with the feature data from its original corpus. This results in a collection of feature datasets that each contain both grammatically correct and incorrect instances, categorized by corpus, error type, and quantity. The feature datasets, in their entirety, can be retrieved from [13].

3.4 Machine-learning classification

The *Weka* data mining software [48] is used as an experimentation platform to investigate the performance and accuracy of several well-known classification algorithms on the feature data discussed in the previous section. This research leverages several classification algorithms, with implementations provided through *Weka*[12]; each is briefly described following.

3.4.1 Classification algorithms

1R One Rule

The *1R* machine-learning algorithm is notable for its simplistic approach to classification: a rule is learned on a single feature, chosen to yield the lowest error rate [55]; thus, the learned model can effectively be considered a single-level decision tree. Missing feature values are permitted, and numeric values are discretized into intervals which help to minimize specificity and over-fitting. In contrast with more complex algorithms (such as C4 decision trees [56]), it has been shown that it is possible for simple rules to perform relatively well on many well-known datasets. The implementation provided by *Weka* is *OneR* [48].

[11] The typical computer used for this task was equipped with an *Intel i7* quad-core processor, and 8 gigabytes of main memory.

[12] Where otherwise unspecified, the classifiers in *Weka* are invoked in their default configurations.

C4.5

C4.5 is a well-known decision tree machine-learning algorithm [56]. Decision trees are comprised of nodes, each of which is a test condition for a feature value. Two or more child nodes stem from each node, representing the paths to be followed (for example, *true* or *false*) based on the evaluation of the test condition at the parent node (in the case of a terminal node, a class prediction is emitted) [9]. C4.5 relies on *entropy* and *gain* measures [56] to optimally select nodes such that each decision point best discriminates between remaining instances of the dataset, meaning that the test condition at each node splits instances (captured by the current path through the tree) into groups that are as disjoint as possible. *Weka* provides an implementation of C4.5 as *J48* [48].

LMT Logistic Model Tree

Logistic Model Tree (LMT) is a hybrid approach to machine learning, leveraging both tree-based and linear regression methods [57]. Similar to tree-based algorithms, the LMT algorithm constructs a tree-like structure formed from nodes which serve as decision points; however, rather than terminal nodes directly emitting a class prediction, they instead represent a logistic regression function over feature values, the evaluation of which is then used in prediction [57].

REPTree Reduced-Error Pruning Tree

REP Tree is a fast tree-learning algorithm that generates decision and regression trees using information gain and reduced error pruning [48], which occurs by progressively replacing subtrees, from the bottom-up, with the majority class of all examples contained within the tree, as long as the replacement incurs an acceptable level or of error.

DecisionStump

The *Decision Stump* algorithm builds simple, one-level decision trees, similar to *One Rule* (3.4.1). [48] [58].

DecisionTable

A Decision Table constructs a collection of instances in which examples from the training data are represented by the values of features from an optimal subset, as determined by the algorithm [59]. Classification of unknown examples is performed by looking-up the feature values (considering only those features from the optimal subset) and selecting the majority class from all training examples that share the same values; if no training examples exist with the same feature values, the majority class of the entire training data is chosen.

RIPPER Repeated Incremental Pruning to Produce Error Reduction

The *RIPPER* algorithm, implemented in *Weka* as *JRIP* [48], learns a collection of pruned and optimized association rules. In the first stage, the building stage, rules are *grown* and *pruned*. A single rule selected to be grown is modified by repeatedly adding conditions to it until all examples captured by the rule are correctly classified; existing rules are pruned using reduced error pruning [60]. After the initial set

of rules has been constructed, further variants of each rule are created by appending to the original rule, and by growing a new rule, using the methods from the build stage. The most efficient variant rule (with minimal *descriptive length*) is chosen for inclusion in the rule set.

PART Partial Decision Trees
The *PART* algorithm is a rule-based classifier that builds partial, pruned decision trees [61]. The terminal node with maximal coverage is extracted from each tree and transformed into a rule to be included in the rule set. PART uses a *separate-and-conquer* strategy to remove examples from the training data as they are covered by a generated rule.

BayesNet Bayesian Network
Weka provides *BayesNet* as an implementation of a classifier based on Bayesian networks [62]. Classes of attribute values exist as nodes within the Bayesian network (a directed graph), and each node is associated with a probability distribution conditional on its parent nodes.

NaiveBayes
Weka also offers an implementation of a *naive* Bayes classifier. It is a simplified Bayesian network in which attribute values are assumed (naively) to be conditionally independent from one another and that there exist no hidden or unknown attributes influencing the class attribute [63]. Thus the value of each attribute contributes directly and independently to the prediction of the class.

Logistic Logistic Regression with Ridge Estimator
The *Logistic* classifier uses a logistic function to predict a binary class, learning linear coefficients or *weights* for each attribute from the training data [48]. A maximum likelihood estimator is used as a ridge estimator to improve prediction [64].

SimpleLogistic
The *SimpleLogistic* classifier uses logistic regression along with *LogitBoost* [9], which results in automatic attribute selection, reducing the size of the attribute set [57, 65].

SGD Stochastic Gradient Descent
The *SGD* classifier provided by *Weka* implements stochastic gradient descent [66] for learning various types of linear models, including *Support Vector Machines (SVM)* [67], and logistic and linear regression.

SMO Sequential Minimal Optimization
The *SMO* algorithm trains a support vector machine in an optimized manner by reducing the size of the *quadratic problem* (QP) needing to be solved [68]. As a result, the savings in time and memory are particularly well suited towards sparse or large data sets.

VotedPerceptron

The *voted perceptron* algorithm performs linear classification using the perceptron algorithm, which learns weights for each attribute in a vector of attributes [69]. Additionally, the algorithm leverages meta-information recorded during the training process, such as the number of iterations before weights in the vector are adjusted. This meta-information is then itself used as a weight to determine the contribution of each vector in a majority vote to predict the class.

3.4.2 Mitigating bias in unbalanced datasets

Some error types, especially with a higher number of errors per sentence, are not able to manifest in sufficiently-many sentences. For example, a *tense* error operates on verbs only, and while the presence of at least one or two verbs can be expected in most sentences, tense errors that are requested in quantities of three or more per sentence may only manifest in a small subset of sentences (those which have at least three or more verbs). The same limitation affects *objectivity* errors, for which pronouns are not as frequent in quantities beyond two or three in a sentence. In such cases, the resulting error corpus may be under-representative in relation to the original corpus from which it was generated.

This research considers a data set to be imbalanced when the examples of the minority class are less in number than two-thirds of the majority class. When such an imbalance occurs, several preprocessing techniques are applied to the data set in order to reduce the effects of over-representation in the majority class. The first technique is to apply sampling: if the minority class is represented by a sufficient number of examples, chosen to be 1000 in this research, then the majority class is simply under-sampled until it is equal in size with the minority class; however if the minority class has fewer than this, then the *Synthetic Minority Over-sampling Technique (SMOTE)* [70] is applied to the dataset in order to synthesize similar examples from preexisting ones. If, after applying one of the aforementioned sampling techniques, the dataset remains imbalanced, then *cost-sensitive* classification and *boosting* [9] techniques are applied. *AdaCost* [71] [13] is employed as a cost-sensitive boosting technique for classifiers operating on imbalanced data sets. However, *AdaCost* does not perform optimally with all classifiers (especially functional classifiers, such as those using regression), and in these cases cost-sensitive classification is used instead (using the *CostSensitiveClassifier* provided by *Weka*). In both cases, a *cost matrix* is supplied to represent the cost of misclassification errors. The cost matrices are defined in such a way that the number of examples of each class are directly and inversely used as the costs for misclassification for that class; thus, an under-represented class will have a higher cost for misclassification than its over-represented counterpart. For example, a data set containing 1000 instances of the grammatically correct class and only 500 instances of the grammatically incorrect class will have its cost matrix adjusted such that the cost for misclassifying a grammatically incorrect instance is 1000, while misclassifying a grammatically correct instance will only incur a cost of 500, thus indicating that misclassification of the minority class should incur relatively twice as much penalty as misclassification of the majority class.

[13] An implementation of *AdaCost* is not provided through *Weka* by default; it can be found in [13].

3.4.3 Training & testing

Using *Weka*, each of the machine-learning classification algorithms from subsection 3.4.1 is applied to the feature data extracted in subsection 3.3.2. Stratified ten-fold cross-validation [9] is used to ensure the relevancy and applicability of the results. This process entails that the training data is divided into n subsets or *"folds"* ($n = 10$); on each iteration, training data is made available from all folds except one (which is held out) and the trained model is tested on the held-out fold. The process iterates n times for each fold and the results are averaged.

Due to the large size and number of corpora (over 3000 separate classification tasks to be performed) it was necessary to expedite the process through distribution and automation. *Weka* provides a library (for *Java*) which has been integrated within the GrammarTools suite of software in order to automate and distribute classification tasks across available computing hardware [11]. Time for completion varies, and depends on the size of the feature data as well as the computational complexity of the classification algorithm; simplistic algorithms (such as *OneR*) applied to smaller datasets (such as the MASC) completed within minutes to hours, while computationally more complex algorithms (such as *Logistic*) applied to larger data sets (such as the OANC) required days or weeks to complete (or in some cases did not complete at all).

The results of the completed classification tasks are collected and analyzed; classification accuracies for each algorithm are organized by corpus, error type, and number of errors per sentence. An analysis and comparison of the results is presented in the following Chapter 4.

Chapter 4

Results & Discussion

This chapter presents the results of the machine-learning classification process discussed in subsection 3.4.3 using the algorithms from subsection 3.4.1, applied to the datasets described in Section 3.2. The classification accuracy is given for each classifier and categorized by corpus (subsection 3.2.4), type of grammatical error (Section 3.2.3), and quantity of errors per sentence. Accuracy is given in percentage, and indicates the rate of success with which the top-performing classifier (using stratified ten-fold cross-validation) is able to predict the grammatical correctness of a sentence.

The results are presented in bar-chart format, displaying classifier accuracy versus the number of errors per sentence. The accuracy appears above each plotted bar value, rounded to the nearest percentage, along with the name of the classifier used to achieve it; two numbers appear directly below this, corresponding to the number of grammatically correct (upper number) and incorrect (lower number) instances that were present in the training data. The names of some classification algorithms have been abbreviated to fit within the plots; the displayed names and their corresponding algorithms are listed in Table 4.1. Bar values plotted with a dashed outline indicate results that have been obtained from an imbalanced dataset, and have had boosting or cost-sensitive classification techniques applied as described in subsection 3.4.2. For completeness, the accuracies obtained from all classifiers are reported in the tables that follow each set of chart results. The highest classification accuracy (per error quantity) appears in bold; values which result from imbalanced training data appear italicized. Where no value has been obtained (due to time or computational constraints), a dash (–) is substituted instead.

label	algorithm	type
1R	One Rule	rule
J48	C4.5 (J48)	tree
LMT	Logistic Model Trees	tree, regression
REPTree	REPTree	tree, regression
DS	Decision Stump	tree
DT	Decision Table	rule
JRip	JRip	rule
PART	PART	rule
BN	Bayes Net	bayesian
NB	Naive Bayes	bayesian
Logistic	Logistic	regression
SL	Simple Logistic	regression
SGD	Stochastic Gradient Descent	svm, regression
SMO	Sequential Minimal Optimization	svm
VP	Voted Perceptron	perceptron

Table 4.1: Classification algorithms and their abbreviated labels

The following sections detail the results for each dataset, organized by corpus (largest to smallest).

4.1 Results by corpus

4.1.1 Verbatim

The *Verbatim* dataset (3.2.4) is a subset of the OANC, featuring articles on language and linguistics from the *Verbatim* magazine [29]. The corpus approximately contains 23,000 original sentences, from which errorful sentences were generated (in most cases, doubling the dataset in size). The following figures show the accuracy of the best-performing classifiers in predicting grammatical correctness in the presence of various errors, as the quantities of these errors increase from 1 to 5 per sentence. Figure 4.1 shows between 64% and 81% classification accuracy for *displace* errors; and between 55% and 70% for *omit* errors. Figure 4.2 shows between 75% and 93% accuracy for *tense* errors, and between 54% and 72% for *number* errors. Figure 4.3 shows between 67% and 99% [1] accuracy for *object* errors, and between 60% and 70% for *insert* errors.

[1] The values plotted with dashed outlines, as in Figure 4.3, Insert, indicate results from imbalanced datasets, and thus irregularities in classification accuracy may be a result of bias.

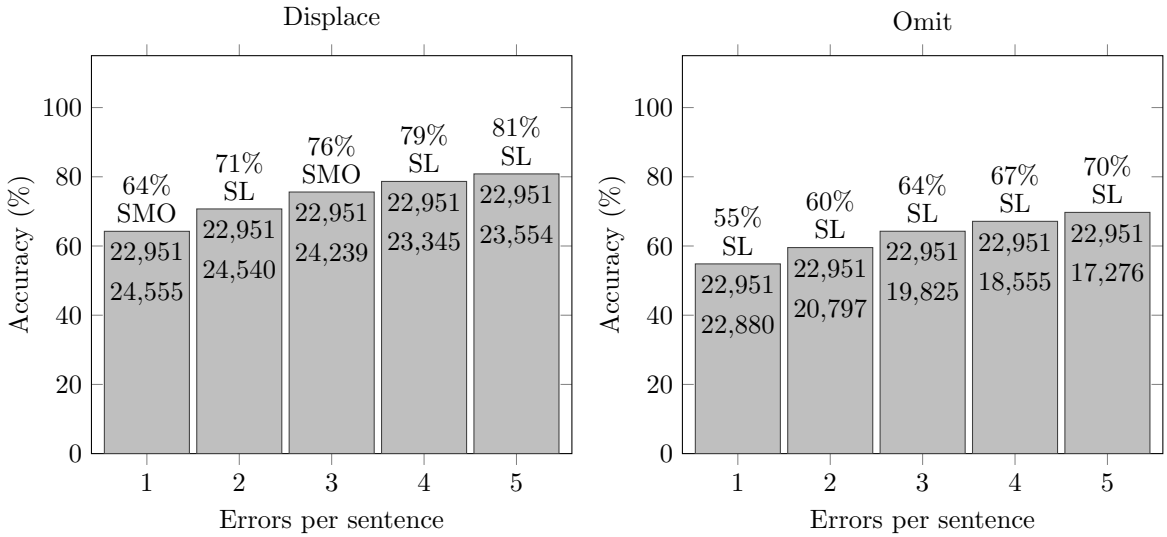


Figure 4.1: Verbatim: Displace, Omit

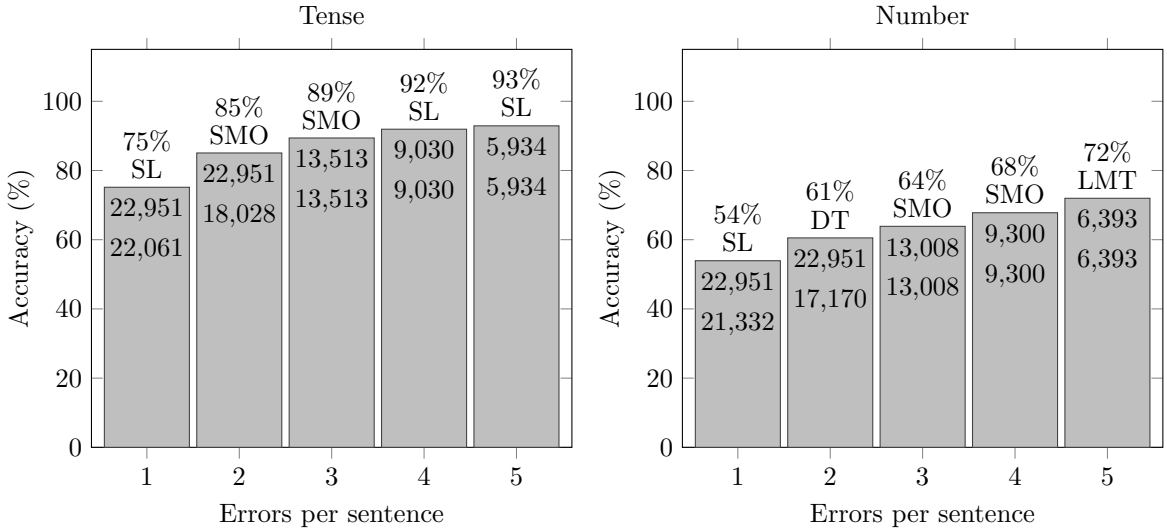


Figure 4.2: Verbatim: Tense, Number

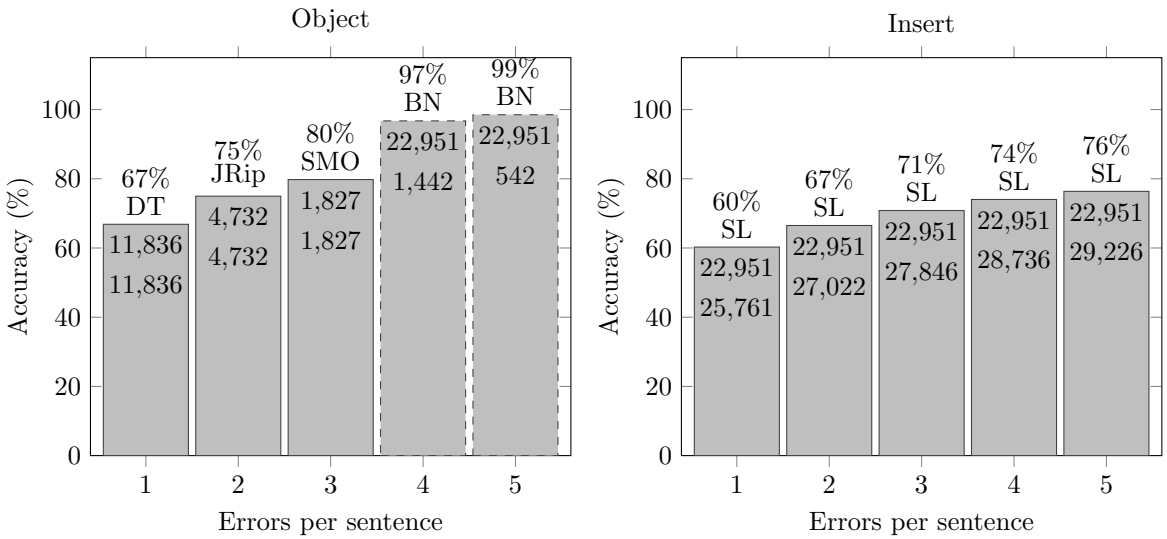


Figure 4.3: Verbatim: Object, Insert

Table 4.2: Verbatim: Displace

Errors per sentence	Accuracy (%)														
	1R	J48	LMT	REPTree	DS	DT	JRip	PART	BN	NB	Logistic	SL	SGD	SMO	VP
1	52	59	–	60	55	63	63	–	62	59	–	64	61	**64**	63
2	56	67	–	66	60	67	70	–	66	63	–	**71**	67	–	69
3	60	72	–	71	62	70	74	–	69	67	–	75	72	**76**	74
4	63	75	–	–	–	73	77	–	72	–	–	**79**	77	–	76
5	65	77	–	–	–	74	79	–	74	–	–	**81**	79	–	79

Table 4.3: Verbatim: Omit

Errors per sentence	Accuracy (%)														
	1R	J48	LMT	REPTree	DS	DT	JRip	PART	BN	NB	Logistic	SL	SGD	SMO	VP
1	50	46	–	48	53	54	53	–	54	54	–	**55**	53	–	54
2	52	52	–	53	53	56	58	–	57	56	–	**60**	56	–	58
3	54	57	–	57	56	60	62	–	60	59	–	**64**	60	–	62
4	55	–	–	–	–	62	64	–	62	61	–	**67**	64	–	65
5	57	–	–	–	–	–	68	–	64	–	–	**70**	65	–	68

Table 4.4: Verbatim: Tense

Errors per	Accuracy (%)														
sentence	1R	J48	LMT	REPTree	DS	DT	JRip	PART	BN	NB	Logistic	SL	SGD	SMO	VP
1	70	73	–	72	72	74	75	–	70	69	–	**75**	74	–	71
2	81	83	–	83	82	83	84	–	78	78	–	85	84	**85**	82
3	85	88	–	87	86	88	88	–	84	83	–	89	89	**89**	86
4	–	91	–	90	88	91	91	90	–	–	–	**92**	91	92	–
5	90	92	–	92	90	92	92	–	89	89	–	**93**	92	93	89

Table 4.5: Verbatim: Number

Errors per	Accuracy (%)														
sentence	1R	J48	LMT	REPTree	DS	DT	JRip	PART	BN	NB	Logistic	SL	SGD	SMO	VP
1	50	49	–	51	54	53	53	–	53	52	–	**54**	52	–	53
2	55	55	–	58	60	**61**	60	–	57	56	–	60	58	–	60
3	57	57	–	59	56	63	62	–	62	62	–	63	59	**64**	62
4	60	–	–	–	65	67	67	–	66	66	–	67	65	**68**	66
5	65	68	**72**	–	69	71	71	–	70	69	–	71	68	72	69

Table 4.6: Verbatim: Object

Errors per	Accuracy (%)														
sentence	1R	J48	LMT	REPTree	DS	DT	JRip	PART	BN	NB	Logistic	SL	SGD	SMO	VP
1	58	64	–	65	62	**67**	67	60	64	62	–	66	62	66	64
2	65	74	–	74	69	75	**75**	71	70	69	–	74	73	75	71
3	71	78	–	77	74	79	79	76	75	74	–	79	78	**80**	75
4	*89*	*96*	–	*95*	–	*96*	*96*	*96*	***97***	*73*	–	*81*	*83*	*83*	*65*
5	*96*	*98*	–	*98*	*81*	*98*	*98*	*98*	***99***	*78*	–	*85*	*90*	*88*	*72*

Table 4.7: Verbatim: Insert

Errors per	Accuracy (%)														
sentence	1R	J48	LMT	REPTree	DS	DT	JRip	PART	BN	NB	Logistic	SL	SGD	SMO	VP
1	51	54	–	53	56	58	59	–	58	55	–	**60**	57	–	59
2	54	61	–	59	59	63	64	–	63	57	–	**67**	65	–	65
3	56	66	–	63	61	66	69	–	66	61	–	**71**	68	–	69
4	57	70	–	–	–	68	72	–	69	–	–	**74**	72	–	72
5	61	72	–	–	–	70	75	–	70	–	–	**76**	73	–	74

4.1.2 Non-Fiction

The *Non-Fiction* dataset (3.2.4) is a subset of the OANC, featuring non-fictional writings by several different authors [29]. The corpus contains 5,754 original sentences from which errorful sentences were generated. The following figures show the accuracy of the best-performing classifiers in predicting grammatical correctness in the presence of various errors, as the quantities of these errors increase from 1 to 5 per sentence. Figure 4.4 shows between 67% and 86% accuracy for *displace* errors; and between 54% and 74% for *omit* errors. Figure 4.5 shows between 77% and 95% accuracy for *tense* errors, and between 53% and 73% for *number* errors. Figure 4.6 shows between 74% and 99% [1], accuracy for *object* errors, and between 60% and 81% for *insert* errors.

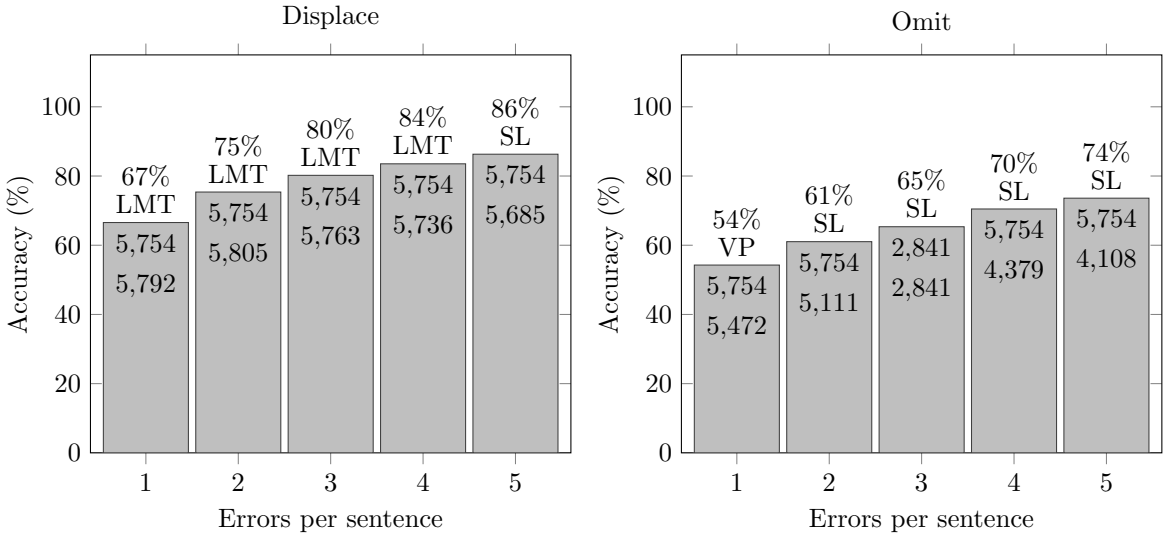


Figure 4.4: Non-Fiction: Displace, Omit

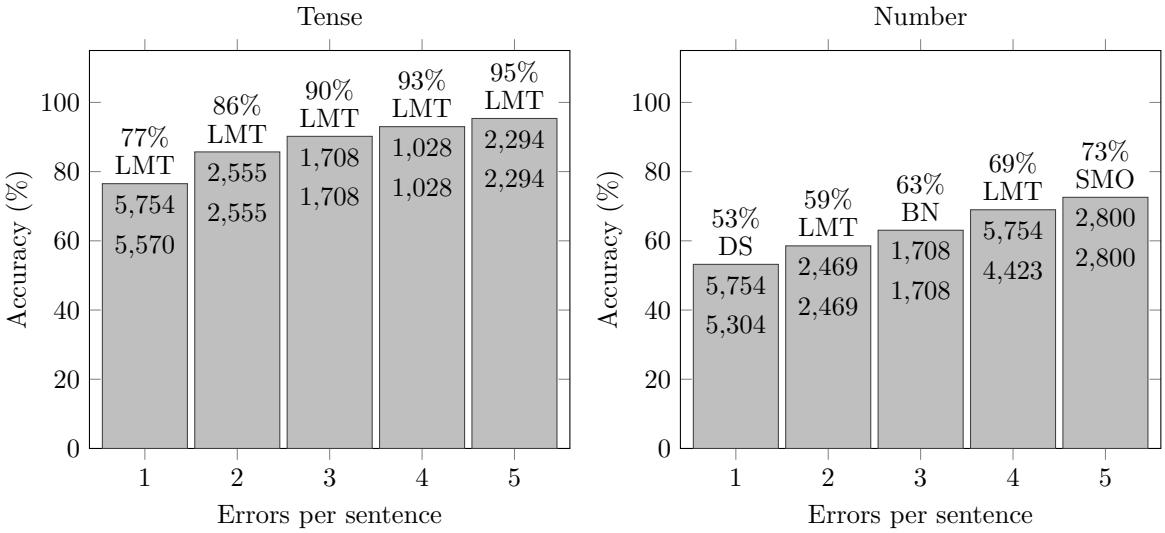


Figure 4.5: Non-Fiction: Tense, Number

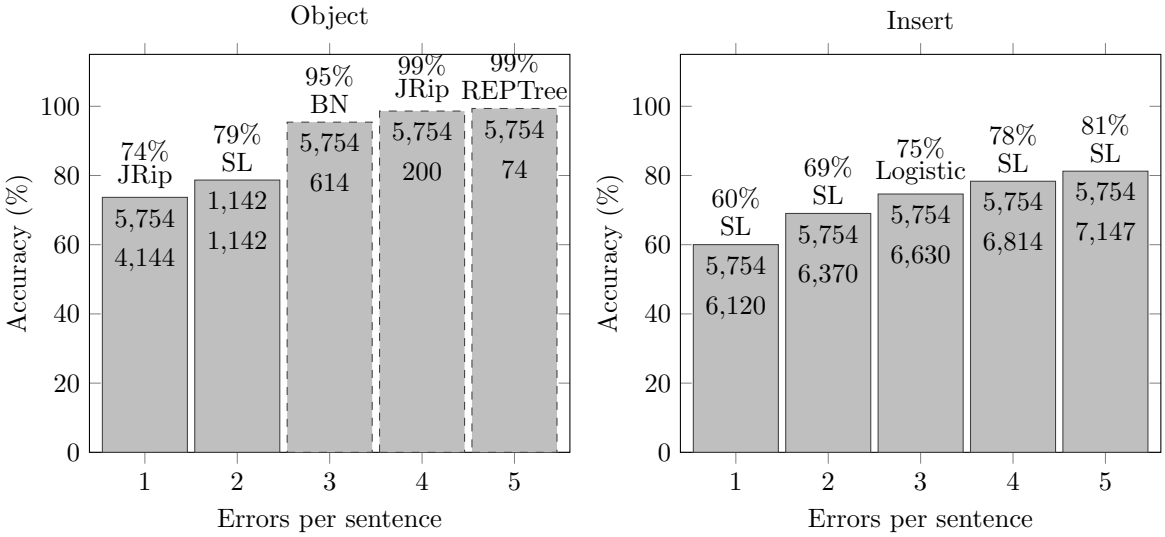


Figure 4.6: Non-Fiction: Object, Insert

Table 4.8: Non-Fiction: Displace

Errors per	Accuracy (%)														
sentence	1R	J48	LMT	REPTree	DS	DT	JRip	PART	BN	NB	Logistic	SL	SGD	SMO	VP
1	53	59	**67**	61	59	64	63	54	62	62	65	66	63	65	64
2	58	69	**75**	68	62	70	72	65	68	66	75	75	73	75	72
3	63	75	**80**	73	65	73	77	72	72	71	80	80	78	79	77
4	65	78	**84**	76	69	76	81	77	74	72	83	83	81	83	80
5	68	80	86	79	73	78	83	79	76	75	86	**86**	85	85	82

Table 4.9: Non-Fiction: Omit

Errors per	Accuracy (%)														
sentence	1R	J48	LMT	REPTree	DS	DT	JRip	PART	BN	NB	Logistic	SL	SGD	SMO	VP
1	50	42	–	48	53	54	54	37	54	53	–	54	52	51	**54**
2	52	52	–	54	56	59	59	48	58	57	–	**61**	59	59	60
3	54	58	–	58	60	61	63	56	62	60	–	**65**	62	65	62
4	59	63	–	62	60	66	67	61	64	63	–	**70**	68	70	68
5	60	66	–	65	62	68	69	64	67	65	–	**74**	72	73	70

Table 4.10: Non-Fiction: Tense

Errors per	Accuracy (%)														
sentence	1R	J48	LMT	REPTree	DS	DT	JRip	PART	BN	NB	Logistic	SL	SGD	SMO	VP
1	71	73	77	74	74	75	76	68	70	69	–	**77**	74	76	72
2	81	83	86	82	82	83	84	80	80	79	–	**86**	84	85	79
3	86	88	**90**	87	88	88	90	88	86	85	–	90	89	89	84
4	90	91	93	91	89	91	92	90	90	90	–	**93**	91	92	87
5	92	94	95	94	92	94	95	94	92	91	–	**95**	95	95	91

Table 4.11: Non-Fiction: Number

Errors per	Accuracy (%)														
sentence	1R	J48	LMT	REPTree	DS	DT	JRip	PART	BN	NB	Logistic	SL	SGD	SMO	VP
1	49	46	53	52	**53**	51	52	29	51	51	–	53	50	50	52
2	53	50	**59**	54	56	57	57	48	58	57	–	58	56	57	58
3	60	58	62	61	63	63	62	55	**63**	62	–	62	60	62	63
4	61	62	**69**	64	62	66	67	60	66	66	–	69	67	69	66
5	66	67	72	69	69	71	71	64	70	69	–	72	69	**73**	69

Table 4.12: Non-Fiction: Object

Errors per	Accuracy (%)														
sentence	1R	J48	LMT	REPTree	DS	DT	JRip	PART	BN	NB	Logistic	SL	SGD	SMO	VP
1	65	70	–	70	69	71	**74**	66	67	64	–	71	70	72	68
2	66	76	–	73	71	75	75	72	73	71	–	**79**	76	77	74
3	*84*	*95*	–	*94*	*79*	*94*	*95*	*95*	***95***	*83*	–	*84*	*85*	*84*	*67*
4	*95*	*98*	–	*98*	*86*	*97*	***99***	*98*	*98*	*86*	–	*89*	*95*	*95*	*66*
5	*98*	*99*	–	***99***	*91*	*99*	*99*	*99*	*99*	*95*	–	*93*	*99*	*99*	*73*

Table 4.13: Non-Fiction: Insert

Errors per	Accuracy (%)														
sentence	1R	J48	LMT	REPTree	DS	DT	JRip	PART	BN	NB	Logistic	SL	SGD	SMO	VP
1	52	53	–	54	56	58	59	43	59	56	60	**60**	56	60	60
2	54	61	–	60	59	65	65	58	64	59	69	**69**	65	69	67
3	58	68	–	65	62	68	70	65	68	62	**75**	74	73	74	71
4	58	72	–	69	65	71	74	71	71	65	–	**78**	76	78	74
5	60	75	–	71	66	73	77	73	73	66	–	**81**	80	80	77

4.1.3 Travel Guides

The *Travel Guides* dataset (3.2.4) is a subset of the *Berlitz Travel Guides* sup-corpus of the OANC; it is comprised of articles describing various travel destinations and activities [29]. The corpus originally contains 9,172 sentences from which errorful sentences were generated. Its treatment within this research is supplemental, as such, fewer errors were generated per sentence. The following figures show the accuracy of the best-performing classifiers in predicting grammatical correctness in the presence of various errors, as the quantities of these errors increase from 1 to 3 per sentence. Figure 4.7 shows between 67% and 82% accuracy for *displace* errors; and between 56% and 67% for *omit* errors. Figure 4.8 shows between 77% and 92% accuracy for *tense* errors, and between 52% and 60% for *number* errors. Figure 4.9 shows between 73% and 98% [1], accuracy for *object* errors, and between 61% and 76% for *insert* errors.

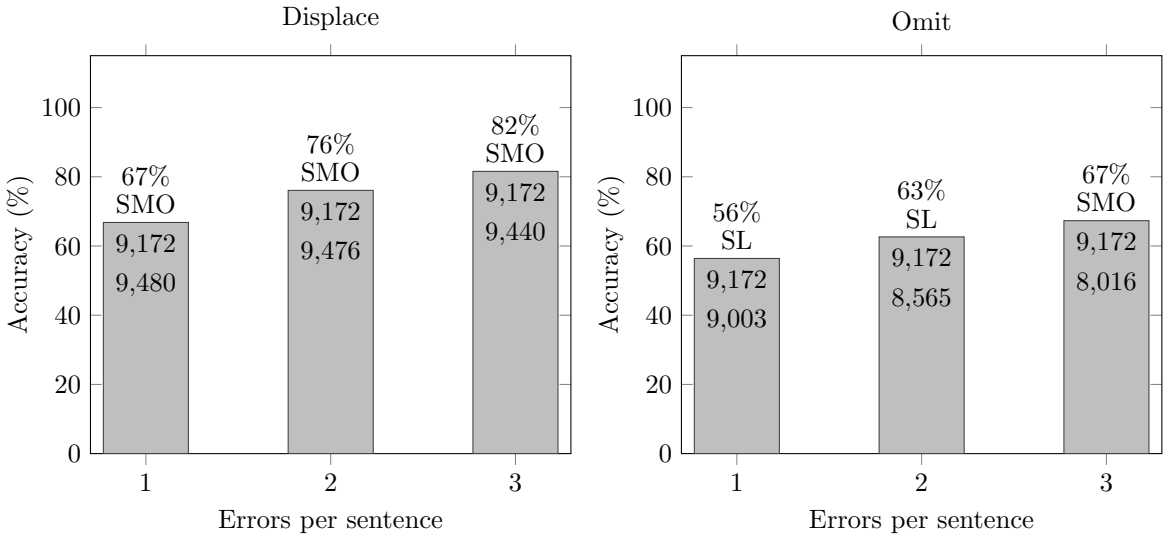


Figure 4.7: Travel Guides: Displace, Omit

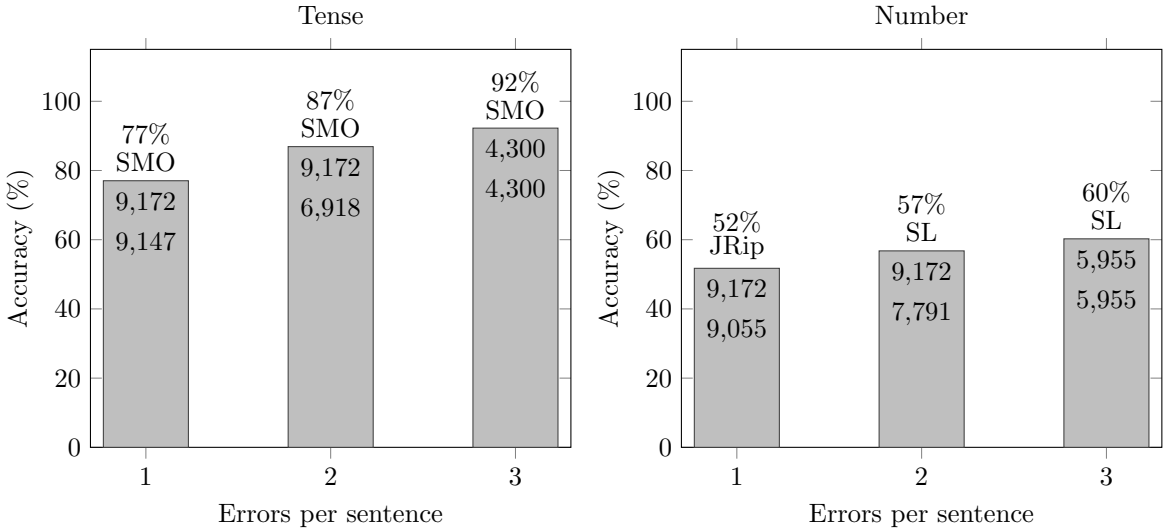


Figure 4.8: Travel Guides: Tense, Number

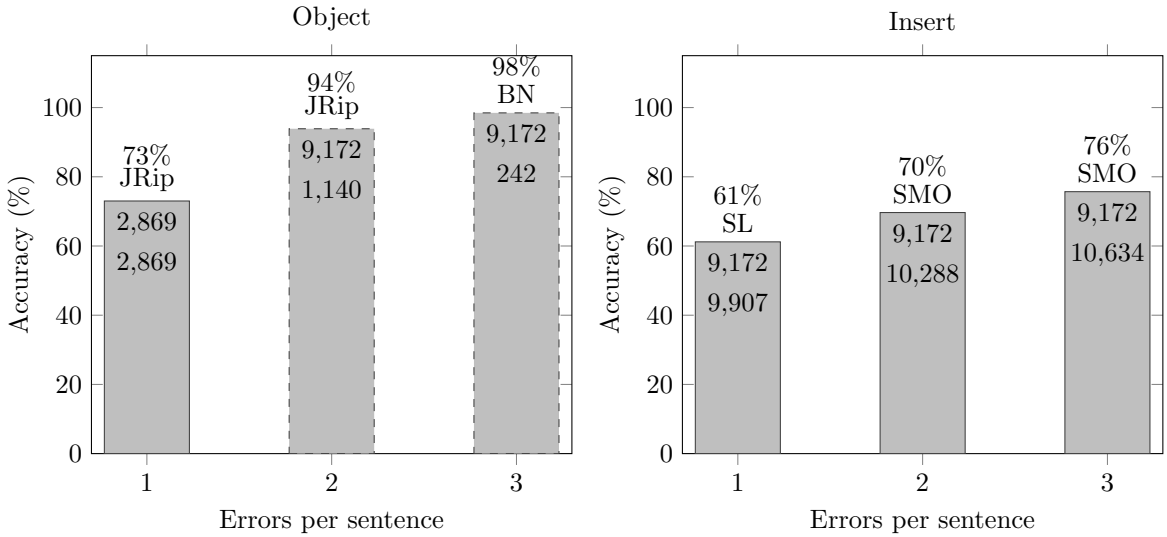


Figure 4.9: Travel Guides: Object, Insert

Table 4.14: Travel Guides: Displace

Errors per	Accuracy (%)														
sentence	1R	J48	LMT	REPTree	DS	DT	JRip	PART	BN	NB	Logistic	SL	SGD	SMO	VP
1	54	61	–	62	59	64	65	54	64	60	–	67	65	**67**	65
2	60	71	–	69	64	71	74	67	69	66	–	76	74	**76**	73
3	64	77	–	73	66	76	78	73	73	70	–	81	80	**82**	78

Table 4.15: Travel Guides: Omit

Errors per	Accuracy (%)														
sentence	1R	J48	LMT	REPTree	DS	DT	JRip	PART	BN	NB	Logistic	SL	SGD	SMO	VP
1	52	43	–	48	52	54	55	39	55	53	–	**56**	53	56	55
2	54	54	–	54	57	60	60	49	59	57	–	**63**	59	62	61
3	56	60	–	61	59	64	65	56	62	60	–	67	66	**67**	65

Table 4.16: Travel Guides: Tense

Errors per	Accuracy (%)														
sentence	1R	J48	LMT	REPTree	DS	DT	JRip	PART	BN	NB	Logistic	SL	SGD	SMO	VP
1	72	74	–	74	72	74	76	–	71	69	–	77	75	**77**	74
2	82	84	–	84	83	84	85	–	81	80	–	87	86	**87**	84
3	87	90	–	89	88	89	90	–	88	87	–	92	92	**92**	88
4	90	92	–	92	90	92	93	–	91	90	–	95	93	**95**	90
5	92	95	–	94	92	94	95	–	94	93	–	**97**	95	96	93

Table 4.17: Travel Guides: Number

Errors per	Accuracy (%)														
sentence	1R	J48	LMT	REPTree	DS	DT	JRip	PART	BN	NB	Logistic	SL	SGD	SMO	VP
1	49	43	–	48	51	51	**52**	30	51	51	–	51	50	50	51
2	51	47	–	54	56	56	55	41	55	55	–	**57**	52	56	56
3	53	51	–	55	57	58	58	49	58	56	–	**60**	56	60	58

Table 4.18: Travel Guides: Object

Errors per	Accuracy (%)														
sentence	1R	J48	LMT	REPTree	DS	DT	JRip	PART	BN	NB	Logistic	SL	SGD	SMO	VP
1	60	69	71	70	60	70	**73**	65	68	64	–	71	68	70	65
2	*76*	*93*	–	*91*	*68*	*90*	***94***	*94*	*94*	*80*	–	*77*	*79*	*78*	*63*
3	*96*	*98*	–	*98*	*86*	*97*	*98*	*98*	***98***	*90*	–	*88*	*95*	*94*	*70*

Table 4.19: Travel Guides: Insert

Errors per	Accuracy (%)														
sentence	1R	J48	LMT	REPTree	DS	DT	JRip	PART	BN	NB	Logistic	SL	SGD	SMO	VP
1	52	53	–	51	53	58	59	45	59	53	–	**61**	59	61	60
2	54	63	–	59	55	65	67	58	65	58	–	70	68	**70**	66
3	57	69	–	67	61	69	72	67	69	61	–	76	73	**76**	72

4.1.4 Berk

The *Berk* dataset is an overlapping subset of the *Non-Fiction* dataset; it is taken from a single collection of writing from a single author [29]. The corpus originally contains 2,297 sentences from which errorful sentences were generated. The error types generated within this dataset have had additional constraints placed on the type of word to which they may apply; for example, a *displace* error is constrained to

operate on only the verbs of a sentence (as described in subsection 3.2.3). The following figures show the accuracy of the best-performing classifiers in predicting grammatical correctness in the presence of various (constrained) errors, as the quantities of these errors increase from 1 to 5 per sentence. Figure 4.10 shows between 66% and 93% accuracy for *insert* errors that have been applied to *verbs* only; and between 61% and 88% for *omit* errors, likewise. Figure 4.11 shows between 65% and 89% accuracy for *displace* errors applied to *verbs* only, and between 60% and 94% for *number* errors constrained to *nouns* and *pronouns*.

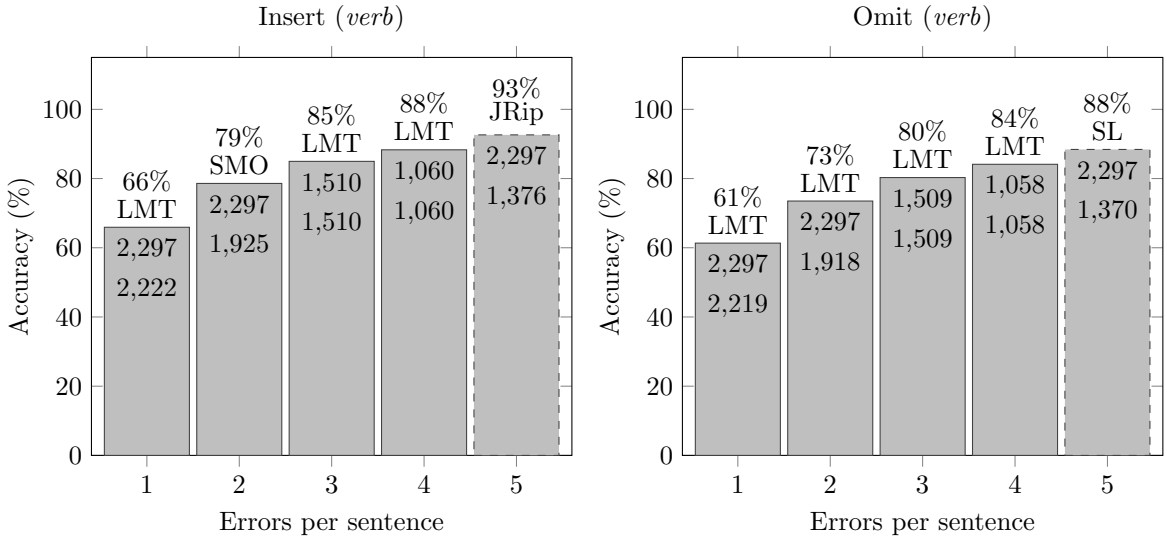


Figure 4.10: Berk: Insert (*verb*), Omit (*verb*)

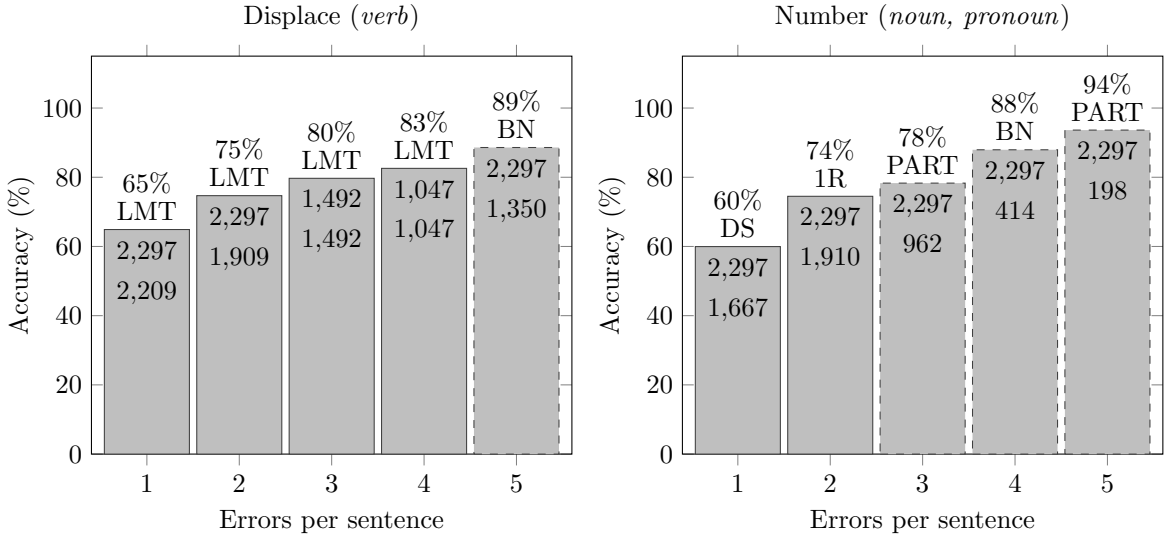


Figure 4.11: Berk: Displace (*verb*), Number (*noun, pronoun*)

Table 4.20: Berk: Displace

Errors per sentence	Accuracy (%)														
	1R	J48	LMT	REPTree	DS	DT	JRip	PART	BN	NB	Logistic	SL	SGD	SMO	VP
1	52	55	64	57	56	60	61	49	61	59	62	**64**	59	62	61
2	54	65	72	62	59	65	67	59	66	64	71	**72**	69	71	68
3	60	69	78	68	66	69	72	68	70	67	77	**78**	75	77	72

Table 4.21: Berk: Insert (*verb*)

Errors per sentence	Accuracy (%)														
	1R	J48	LMT	REPTree	DS	DT	JRip	PART	BN	NB	Logistic	SL	SGD	SMO	VP
1	57	58	66	60	61	62	65	52	62	61	65	**66**	63	66	62
2	64	73	78	72	72	73	76	70	72	71	78	78	77	**79**	70
3	75	80	**85**	78	78	79	81	79	79	78	83	84	82	84	74
4	83	84	88	83	84	84	85	83	82	81	84	**88**	87	87	75
5	*84*	–	–	–	–	–	***93***	–	–	–	*91*	*92*	*90*	*92*	–

Table 4.22: Berk: Omit

Errors per sentence	Accuracy (%)														
	1R	J48	LMT	REPTree	DS	DT	JRip	PART	BN	NB	Logistic	SL	SGD	SMO	VP
1	49	43	54	46	53	53	54	36	52	52	48	**54**	49	50	53
2	50	49	58	52	54	55	57	46	56	56	57	**58**	55	58	58
3	54	56	64	55	55	58	61	51	60	58	62	**64**	61	62	62

Table 4.23: Berk: Tense

Errors per sentence	Accuracy (%)														
	1R	J48	LMT	REPTree	DS	DT	JRip	PART	BN	NB	Logistic	SL	SGD	SMO	VP
1	68	67	71	69	70	70	71	62	66	64	68	**71**	67	69	66
2	78	79	81	78	79	80	80	76	74	72	79	**81**	80	80	77
3	82	84	86	82	83	85	86	82	80	79	82	**86**	82	85	80
4	87	91	**93**	90	88	92	91	91	89	85	91	93	91	93	88
5	–	–	*93*	–	–	*92*	***94***	–	–	–	–	*92*	*90*	*92*	–

Table 4.24: Berk: Omit (*verb*)

Errors per	Accuracy (%)														
sentence	1R	J48	LMT	REPTree	DS	DT	JRip	PART	BN	NB	Logistic	SL	SGD	SMO	VP
1	52	52	61	54	55	58	60	46	59	58	60	**61**	58	60	60
2	56	67	73	65	63	67	70	62	67	67	73	**73**	71	72	69
3	59	72	80	70	67	74	76	72	72	73	79	**80**	78	79	71
4	63	79	84	76	68	78	80	75	74	74	81	**84**	81	83	73
5	*65*	–	–	–	–	–	*88*	–	–	–	*88*	***88***	*87*	*88*	–

Table 4.25: Berk: Number

Errors per	Accuracy (%)														
sentence	1R	J48	LMT	REPTree	DS	DT	JRip	PART	BN	NB	Logistic	SL	SGD	SMO	VP
1	49	46	52	52	**54**	53	51	31	49	50	46	52	49	49	52
2	53	56	60	57	59	**60**	58	45	57	55	58	60	54	60	59
3	57	56	60	57	60	59	59	52	61	**61**	58	60	56	59	59

Table 4.26: Berk: Object

Errors per	Accuracy (%)														
sentence	1R	J48	LMT	REPTree	DS	DT	JRip	PART	BN	NB	Logistic	SL	SGD	SMO	VP
1	59	61	65	64	**65**	63	63	57	60	60	62	65	59	63	62
2	*64*	***84***	*79*	*79*	*64*	*76*	*83*	*83*	*77*	*75*	*74*	*71*	*72*	*72*	*48*
3	*82*	***93***	*89*	*91*	*71*	*93*	*93*	*93*	*92*	*72*	*89*	*78*	*83*	*82*	*52*

Table 4.27: Berk: Insert

Errors per	Accuracy (%)														
sentence	1R	J48	LMT	REPTree	DS	DT	JRip	PART	BN	NB	Logistic	SL	SGD	SMO	VP
1	51	48	57	47	52	55	55	38	57	52	54	**57**	52	54	56
2	50	57	65	54	58	62	62	52	63	57	64	**65**	62	64	61
3	53	62	71	59	61	63	67	59	66	59	**71**	71	70	71	66

Table 4.28: Berk: Displace (*verb*)

Errors per sentence	Accuracy (%)														
	1R	J48	LMT	REPTree	DS	DT	JRip	PART	BN	NB	Logistic	SL	SGD	SMO	VP
1	52	59	**65**	61	58	63	63	53	63	61	64	65	63	64	63
2	63	69	75	70	65	70	73	65	69	68	74	74	71	**75**	71
3	66	75	**80**	71	69	76	76	72	74	72	78	80	77	79	74
4	71	76	83	74	74	77	77	74	77	75	80	**83**	80	80	74
5	*75*	–	–	–	–	–	*88*	–	***89***	*85*	*87*	*87*	*85*	*86*	–

Table 4.29: Berk: Number (*noun, pronoun*)

Errors per sentence	Accuracy (%)														
	1R	J48	LMT	REPTree	DS	DT	JRip	PART	BN	NB	Logistic	SL	SGD	SMO	VP
1	51	55	60	57	**60**	59	57	40	54	52	55	59	58	58	58
2	**74**	65	70	64	58	74	71	65	74	63	70	70	67	70	61
3	*58*	*76*	–	*72*	*60*	*67*	*75*	***78***	–	*67*	*71*	*67*	*69*	*69*	*46*
4	*70*	*86*	–	*85*	*63*	*84*	*86*	*87*	***88***	*72*	*80*	*71*	*76*	*75*	*45*
5	*83*	*93*	*89*	*93*	*66*	*92*	*93*	***94***	*94*	*78*	*89*	*75*	*84*	*83*	*47*

4.1.5 MASC

The *MASC* dataset (3.2.4) is a subset the OANC; it contains 2,902 sentences that span a variety of domains and topics featured in the larger OANC [29]. Its use within this research is supplementary; however, notably it contains a dataset that has been generated using mixed (*any*) error types. The results for isolated errors are given in Figure 4.12, Figure 4.13, Figure 4.14; shown in Figure 4.15 are the classification accuracies achieved (61% - 84%) when any type of error may be present in a sentence.

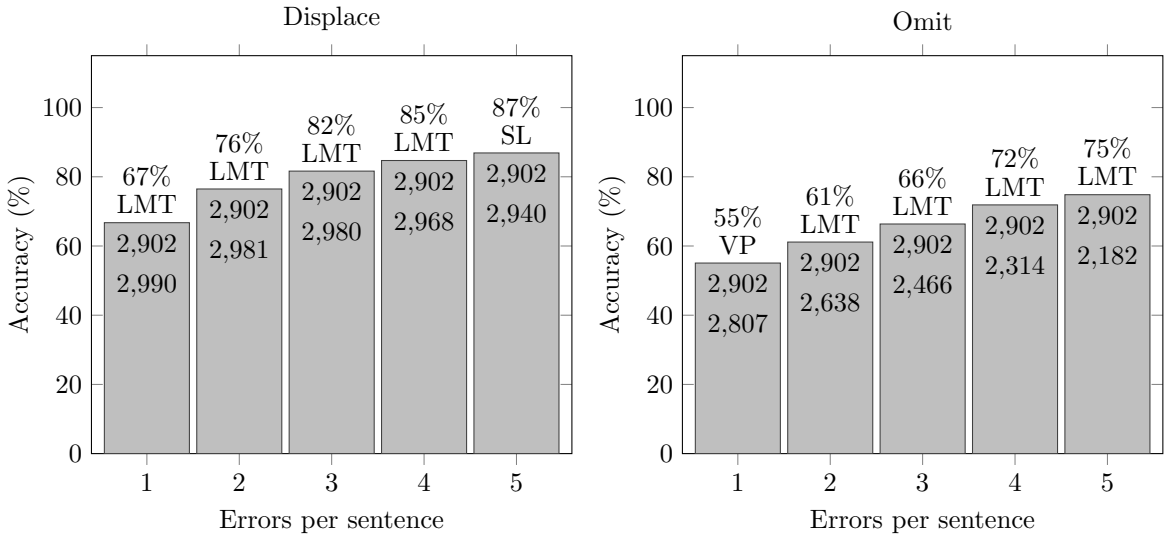


Figure 4.12: MASC: Displace, Omit

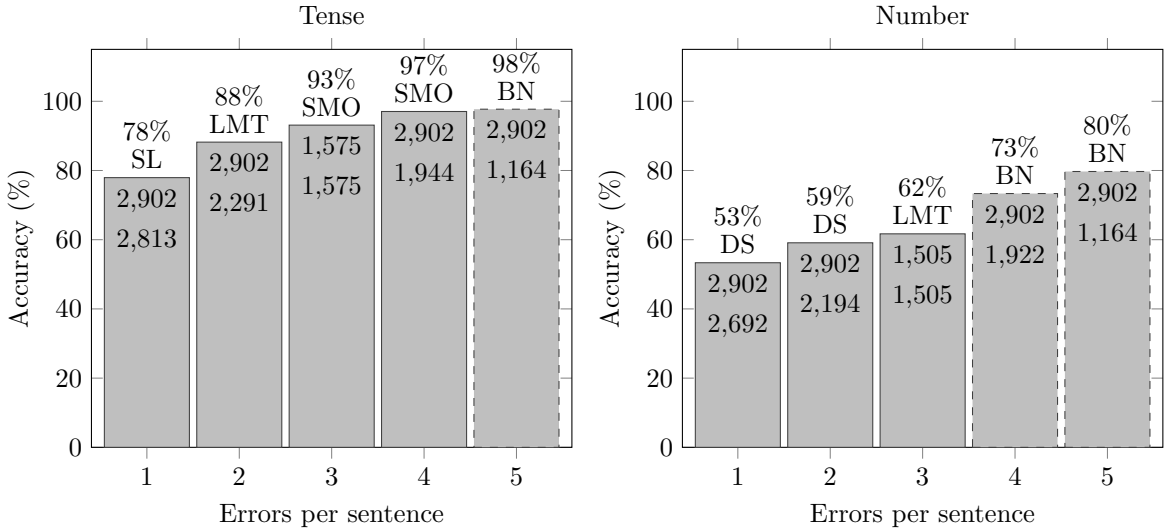


Figure 4.13: MASC: Tense, Number

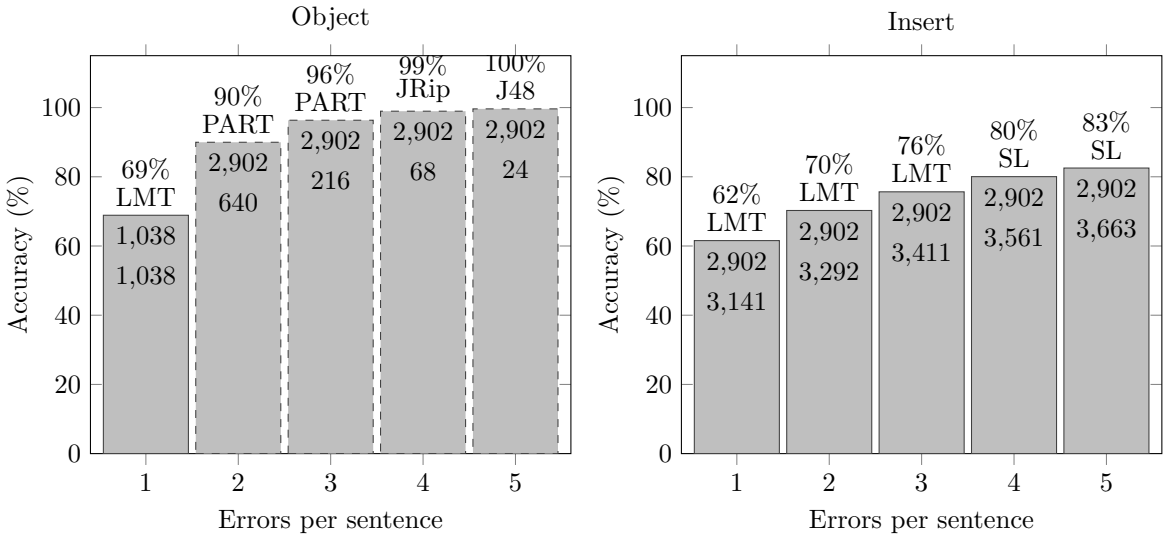


Figure 4.14: MASC: Object, Insert

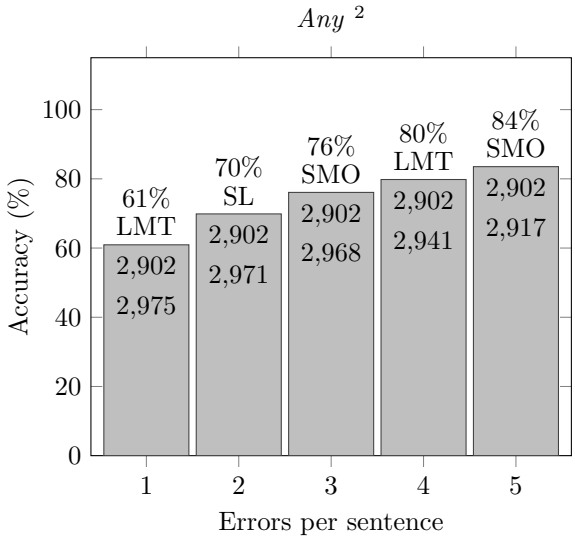


Figure 4.15: MASC: *Any*

[2]this dataset may contain any and all errors.

Table 4.30: Displace

Errors per	Accuracy (%)														
sentence	1R	J48	LMT	REPTree	DS	DT	JRip	PART	BN	NB	Logistic	SL	SGD	SMO	VP
1	53	63	**67**	63	59	65	66	55	64	61	66	66	64	66	64
2	62	72	**76**	70	64	72	73	66	70	68	75	76	74	76	72
3	64	77	**82**	73	67	75	77	73	74	71	81	82	79	81	77
4	69	79	**85**	75	69	77	80	78	76	74	84	84	83	84	80
5	71	81	87	79	73	81	83	81	78	76	86	**87**	86	87	83

Table 4.31: MASC: Omit

Errors per	Accuracy (%)														
sentence	1R	J48	LMT	REPTree	DS	DT	JRip	PART	BN	NB	Logistic	SL	SGD	SMO	VP
1	49	45	54	47	52	54	53	38	54	54	51	54	50	52	**55**
2	51	53	61	53	56	58	59	48	60	58	60	**61**	57	61	60
3	54	61	**66**	57	59	61	63	55	63	61	66	66	63	66	65
4	57	64	72	61	61	67	68	61	66	64	72	**72**	66	72	68
5	59	68	75	63	62	69	71	65	68	66	74	**75**	73	74	70

Table 4.32: MASC: *Any*

Errors per	Accuracy (%)														
sentence	1R	J48	LMT	REPTree	DS	DT	JRip	PART	BN	NB	Logistic	SL	SGD	SMO	VP
1	51	56	61	55	54	58	59	46	59	58	60	**61**	58	60	60
2	56	65	70	64	62	66	67	60	66	64	69	**70**	67	69	67
3	63	72	76	69	67	70	73	69	71	68	75	76	72	**76**	72
4	65	73	80	73	68	73	76	72	74	71	79	**80**	76	80	75
5	69	79	83	76	71	77	78	77	77	74	83	83	82	**84**	79

Table 4.33: MASC: Tense

Errors per	Accuracy (%)														
sentence	1R	J48	LMT	REPTree	DS	DT	JRip	PART	BN	NB	Logistic	SL	SGD	SMO	VP
1	73	75	78	76	74	76	77	70	73	72	77	**78**	76	77	73
2	83	86	88	85	85	86	86	85	83	81	87	**88**	86	88	83
3	88	90	93	90	89	91	91	90	88	88	91	93	92	**93**	87
4	92	96	97	95	93	95	96	96	94	92	95	97	97	**97**	92
5	*93*	*98*	–	*97*	*95*	*96*	*98*	*97*	***98***	*96*	–	*97*	*96*	*97*	*86*

Table 4.34: MASC: Number

Errors per	Accuracy (%)														
sentence	1R	J48	LMT	REPTree	DS	DT	JRip	PART	BN	NB	Logistic	SL	SGD	SMO	VP
1	49	51	53	51	53	51	50	31	50	51	47	53	48	49	53
2	55	53	58	58	**59**	58	56	43	55	56	57	58	56	58	58
3	55	54	**62**	56	59	60	60	53	61	61	59	62	57	60	59
4	*57*	*70*	–	*67*	*65*	*59*	*71*	*72*	***73***	*65*	–	*68*	*69*	*69*	*53*
5	*64*	*79*	–	*75*	*65*	*69*	*78*	*79*	***80***	*66*	–	*71*	*71*	*72*	*54*

Table 4.35: MASC: Object

Errors per	Accuracy (%)														
sentence	1R	J48	LMT	REPTree	DS	DT	JRip	PART	BN	NB	Logistic	SL	SGD	SMO	VP
1	61	65	69	67	65	67	67	61	64	63	65	**69**	66	68	63
2	*70*	*89*	–	*86*	*67*	*85*	*90*	***90***	*90*	*72*	–	*78*	*80*	*80*	*57*
3	*86*	*96*	–	*95*	*83*	*95*	*96*	***96***	*96*	*85*	–	*84*	*91*	*91*	*59*
4	*97*	*99*	–	*98*	*90*	*98*	***99***	*98*	*98*	*93*	–	*89*	*96*	*96*	*65*
5	*99*	*100*	*99*	*99*	*98*	*99*	***100***	*100*	*99*	*99*	–	*95*	*99*	*99*	*81*

Table 4.36: MASC: Insert

Errors per	Accuracy (%)														
sentence	1R	J48	LMT	REPTree	DS	DT	JRip	PART	BN	NB	Logistic	SL	SGD	SMO	VP
1	52	56	62	51	55	59	59	43	59	55	60	**62**	57	60	60
2	56	64	70	61	59	64	66	58	67	59	70	**70**	69	70	67
3	58	69	**76**	65	61	69	71	66	71	63	76	75	74	76	72
4	59	74	80	69	64	71	75	70	74	66	79	**80**	79	80	74
5	62	77	82	73	67	74	79	75	75	67	82	**83**	80	82	77

4.2 Discussion of classifier performance

A readily observable tendency is for classification accuracy to improve as the number of grammatical errors increase within a sentence. This appears to be true for all error types and is independent of the corpus used. The improvements in accuracy in many cases are modest (between 3% and 10%); in some cases, such as *tense*, *displace*, and *insert* errors, this effect becomes less pronounced towards 5 errors per sentence, while in other cases, such as *number* and *omit* errors, the gains are incremental. Accuracy thus appears to in some cases follow logarithmic-like increases while in others behave more linearly as the quantity of errors per sentence is increased. For example, one may predict that adding a sixth *tense* error

to a sentence may increase detection of grammatical incorrectness by only a single or less percentage, whereas adding a sixth *number* error may increase accuracy by 2% to 3%.

Some noticeable irregularities occur from imbalanced data, such as in Figure 4.3 (plotted with a dashed outline). Imbalance occurs as a result of under-representation of grammatically incorrect sentences, as explained in subsection 3.4.2. Though techniques were used to mitigate underrepresentation, bias is still the likely cause for outlying results, such as from classifying *object* errors.

Constraining error types to a subset of word types appears to improve classification of grammatical incorrectness. For example, the average classification accuracy across all other corpora for *insert* errors is 75%, whereas the classification accuracy for *insert* errors in the *Berk* corpus, constrained to *verbs*, is 85% – a gain of 10%, and similar gains for errors in quantities 1 through 5. Likewise, accuracy is improved for *omit* and *displace* errors when constrained to *verbs*, and for *number* errors when constrained to *nouns* and *pronouns*. Thus classification accuracy improves for sentences containing errors occurring on specific word types.

From the results in Section 4.1, it is evident that several classifiers routinely achieve best performance, such as *SimpleLogistic* (SL) (Section 3.4.1) and *Logistic Model Trees* (LMT) (Section 3.4.1). Independent of corpora, error type, or quantity, the best-performing classifiers tend to perform comparably well in several different contexts. The results data in tabular form for each corpus and error type confirm that several of the top-performing classifiers perform equally or near-equally well. In many cases, the top-three or -five classifiers achieve accuracies within a single or less percentage point of each other. Figure 4.16, Figure 4.17, and Figure 4.18 highlight the similarities in classification accuracies and compare training time across the top-five classifiers for *tense* errors occurring in the Verbatim corpus. The classifiers and corresponding accuracies are plotted from first-best (leftmost) to fifth-best (rightmost); the inset number gives the time in seconds taken to train a single model on the dataset [3].

[3] 10-fold stratified cross-validation is used for training and testing, thus the total time when using this method can be expected to take approximately 10 times the duration for training a single model

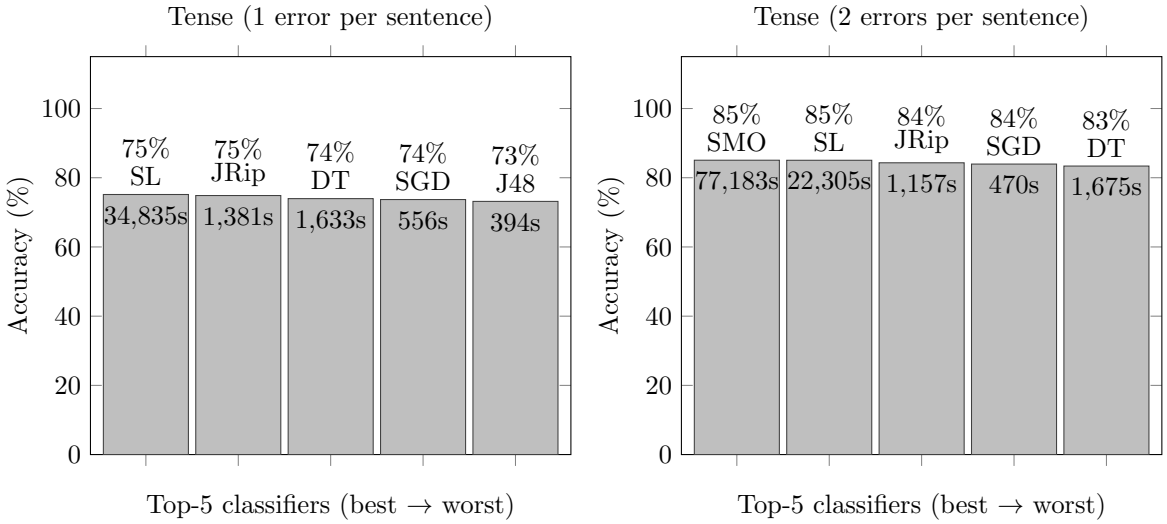


Figure 4.16: Verbatim: Tense (1, 2 errors) comparison

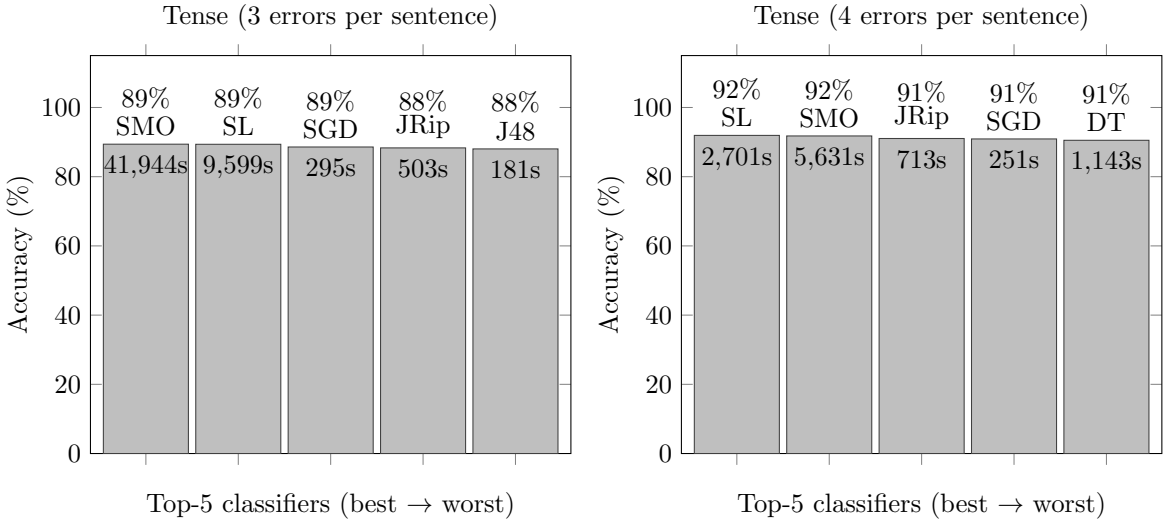


Figure 4.17: Verbatim: Tense (3, 4 errors) comparison

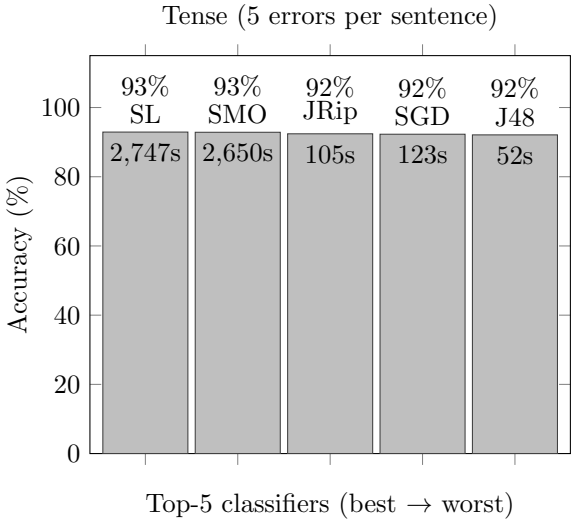


Figure 4.18: Verbatim: Tense (5 errors) comparison

In Figure 4.16, comparing the top-five classifiers for a single *tense* error, the best-performing classifier is SimpleLogistic (Section 3.4.1); it achieves an accuracy of 75%, while the fifth-best classifier (J48) achieves still nearly that, at 73%. The range in classification accuracies is thus just 2% among the top-five classifiers for this error type and quantity. A similar pattern emerges from the remaining comparisons – the variance in accuracy among the top-five best-performing classifiers is almost negligible (within 2%) for any error quantity; this appears to hold across corpora, and error type.

When considering training time as a performance metric, the differences in classification algorithms becomes more apparent. For example, in Figure 4.17, while both the SMO and SimpleLogistic classifiers achieve an accuracy of 89% in classifying sentences containing *tense* errors, the former algorithm requires nearly 42,000 seconds (11.7 hours) to train a single model [4], whereas the latter takes only approximately 9,600 seconds (2.7 hours) – nearly five times less. The next-best algorithms perform similarly (within a single percentage) yet require only a fraction of the time for training. For example, the J48 classifier requires only 181 seconds to train a single model on the same dataset for which SimpleLogistic requires nearly 9,600 seconds – a significant difference for only small gains in accuracy. While this research primarily focuses on classification accuracy, it is evident that training time is also a worthy consideration when working with large datasets.

Consistency in classifier performance is another interesting metric: while some algorithms may perform better in the context of a specific error type, it is worth comparing performance amortized across different contexts. For example, when considering performance on *tense* errors, the J48 classifier appears to perform similarly-well to SimpleLogistic; however when considering performance across all error types, SimpleLogistic appears more often as a top-performing classifier, often exceeding the accuracy of other, faster classifiers (such as J48) by several percentage points or more. Considering overall per-

[4] The typical computer used for this task was equipped with an *Intel i7* quad-core processor, and 8 gigabytes of main memory.

formance, and especially consistency in classification accuracy, it is observed that LMT (Section 3.4.1), SimpleLogistic (Section 3.4.1), Logistic (Section 3.4.1), and SMO (Section 3.4.1) are consistently among the best-performing classification algorithms across error type, quantity, and corpora.

Analyzing the results from the largest and best-represented dataset (Verbatim), summarily this research achieves between 75% and 93% accuracy in determining the grammatical correctness of sentences in which *tense* errors occur (ranging between 1 and 5 errors per sentence); between 64% and 81% for *displace* errors; 60% to 76% for *insert* errors; 55% to 70% for *omit* errors; 54% to 72% for *number* errors; and between 67% and 99% for *object* errors [1]. Averaging the results across all corpora, the research achieves between 76% and 95% accuracy for *tense* errors (ranging between 1 and 5 errors per sentence); between 66% and 85% for *displace* errors; 60% to 76% for *insert* errors; 55% to 73% for *omit* errors; 53% to 75% for *number* errors; and between 70% and 99% for *object* errors [1]. Thus this research finds that sentences exhibiting *tense* and *displace* errors are the most accurately detected for grammatical incorrectness, while sentences containing *omit* and *number* errors are more difficult to detect; and that in general, grammatical incorrectness becomes more reliably predictable (beyond 70%) as the number of errors per sentence increases.

4.3 Comparison with similar methods

Grammar verification is an open-ended challenge, and varying approaches can assume different goals in the types of errors they purport to detect. While Section 2.3 reports in general on the results of various grammar checking approaches, perhaps the closest comparative findings are from the research outlined in Section 2.3.2. The results from Andersen's work [1] present classification accuracies categorized by error type, which are largely congruent with the results and types of errors presented in this chapter. Comparing Figure 2.4 from Section 2.3.2 with the results presented throughout Section 4.1, some direct and proximate comparisons can be derived: for *Missing word* errors, detected with 62% accuracy in the compared research, this research achieves between 55% and 70% accuracy for comparatively similar *omit* errors, for a similarly sized corpus (Verbatim); where *Unnecessary word* errors are detected with 60% accuracy in the compared research, this research achieves between 60% and 76% for similar *insert* errors; for *Incorrect tense* errors detected with 58% accuracy in the compared research, this research achieves between 75% and 93% accuracy for sentences containing *tense* errors; where *Word order* errors are detected with 61% in the compared research, this research achieves between 64% and 81% in the case of similar *Displace* errors; finally, where the overlapping error types *Wrong form*, *Agreement*, *Derivation*, and *Incorrect inflection* are detected with between 68% and 77% accuracy (collectively) in the compared research, this research achieves between 54% and approximately 80% accuracy [5] (collectively) among similar *number* and *object* errors. However, it is important to note that the compared research does not specify or constrain the number and type of errors per sentence for the accuracies reported previously [1]; hence a range is given (representing error quantities 1 through 5) when comparing with the results from this research, which are categorized by error type and quantity.

[5] The upper-bound is corrected for any potential bias introduced by underrepresentation in the Verbatim dataset for *object* errors; it disregards the outlying value of 99%.

Comparing classification accuracy by error quantity, Andersen's work detects single errors with an accuracy of 49%, and towards 5 errors, an accuracy of 72% research [1]; the results in this research achieve an accuracy of 63% in detecting grammatical correctness for sentences containing a single error [6] , and up to approximately 81% for five errors. However, it is important to note that this comparison is made from a single classification model in the compared research, in contrast with the average of several models for different error types in this research. While the results and comparisons in this section and chapter do not have exact parity with other research, we believe this research achieves some noted improvements in accuracy over the nearest methods.

[6] This is obtained by averaging the classification accuracies for a single error across all error types from the Verbatim corpus.

Chapter 5

Conclusion & Future Work

The grammatical structure of human natural language uniquely shapes the understanding and exchange of information, especially in the digital and written form, and its common misuse warrants methods for automatically detecting mistakes in grammatical syntax. The work accomplished in this research seeks to address this challenge, and in doing so, defines and implements a unique approach that combines machine-learning and statistical natural language processing. Several core methods are established: (1) the automatic generation of grammatical errors, and the formation of grammatically parallel corpora (Section 3.2); (2) the definition and extraction of over 150 features from a sentence; and (3) the application of extracted feature data to be used in machine-learning classification.

Chapter 3 describes the theory and implementation of the error generation, feature extraction, and machine-learning processes, along with supporting software tools. The experimental results presented in Chapter 4 describe the classification accuracies of the machine-learning algorithms on the various corpora and grammatical error types explored by this research.

5.1 Findings & contributions

Three important methods are established in addressing the challenge of grammatical verification, particularly from an NLP and machine-learning approach. The first is a supervised method of generating and collecting example data for grammatical correctness and incorrectness (forming a grammatically parallel corpora). The latter especially addresses the need for, and limited existence of, publicly-available *error corpora* that contain purely grammatically incorrect sentences. The theory and implementation of this are detailed in Section 3.2; the publicly-available open-source software, *CorpusTools* [13], allows control over the error-generation process and the construction of grammatically erroneous corpora from an original input corpus.

The second and third methods established in this research consists of the definition and extraction of over 150 sentence features, and the application of the extracted feature data with various machine-learning algorithms in order to learn and predict grammatical correctness. Many of the defined features

are derived using current methods in statistical natural language processing. The feature extraction and machine-learning processes are described in Section 3.3 and Section 3.4, along with the publicly-available open-source implementation through *GrammarTools* [13].

Using these methods, this research finds it is possible to predict grammatical incorrectness in sentences containing various types of grammatical errors (defined in Section 3.2.3), and reports the accuracy of classification for each error type, quantity, and corpus. Summarily, the research finds that sentences containing *tense* errors are classified with 76% to 95% accuracy (ranging between 1 and 5 errors per sentence); between 66% and 85% for *displace* errors; 60% to 76% for *insert* errors; 55% to 73% for *omit* errors; 53% to 75% for *number* errors; and 70% to above 80% for *object* errors. *Tense* and *displace* errors are the most reliably detected, while *omit* and *number* errors are more difficult to detect. One corollary is that, regardless of error type or corpus, all errors become increasingly easier to detect (above 70% accuracy) as the error quantity approaches or exceeds five errors per sentence; a second corollary is that accuracy is improved for classifiers that are trained on more specific types of errors, such as when constraining the types of words to which an error is applied.

5.2 Limitations

Some boundaries and caveats exist in the theoretical and experimental findings in this research. First, the research is conducted with a set of discretely defined types of grammatical errors (Section 3.2.3), and while these errors aim to represent many of those that occur commonly in English language misuse, it is important to note that they are only a subset of what may render a sentence grammatically incorrect. It is also mentioned in Section 3.2 that the sources of example data (correct and incorrect) contain some degree of noise that may result in misrepresentation; subsection 3.4.2 explains how for some of the smaller data sets, under-representation may occur during the machine-learning phase.

It is important to note that the results presented in this research are on a per-corpus and per-error basis: there is little reporting on the performance of a single model applied to the classification of all error types across all corpora. Finally, it remains to be measured how well models trained on one corpus perform on another corpus of a different topic, or on free-form sentences provided by a user in real time.

5.3 Future Work

The challenge of grammatical verification is still an open-ended question and certainly the methods presented in this research can be further refined and elaborated on. Particularly, in the area of feature definition and extraction, new features may be defined to improve data separation and classifier performance. For example, as defined in Appendix 1, there are several features currently describing and tracking the use (or misuse) of verbs in a sentence. These features are included to bolster the detection of verb-related errors, such as *tense* errors, and indeed such errors are some of the most reliably detected in this research. Expectedly, if similar and more features are added to describe the use and misuse of other types of words (such as adjectives or adverbs), detection of the related error types may

similarly improve. Other useful features may be defined to capture the word complexities or frequencies for a sentence, informing the training process about the impact of rare or complex words on perceived grammatical correctness. Still more features can be added to address sentences on a semantic level, such as recording information about the actors, subjects, objects, and actions in a sentence, or even perhaps features that span multiple sentences or document boundaries.

Feature optimization is another consideration; while this research aims to propound many features in the hopes that they are useful, it is likely that not all features contribute meaningfully or significantly in the machine-learning process and may be optimized away; future research can employ *feature selection* strategies to this end. Further, a numerical class feature is implemented, though unused in this research, to express the number or severity of grammatical errors present in the sentence – this feature can be incorporated into future experiments. Additionally, since the features and NLP techniques used in this research are statistical and evidence-based, it is possible to apply them to other languages. An area of future interest then may be to adapt the features used here for English sentences to sentences of other languages, especially those which are structurally similar to English.

The CorpusTools software can be extended further, such as to include more types of errors and greater specificity over the generation of errors. *Stochastic universal sampling* [72] can be adapted to ensure that the statistical distribution of errors in the generated corpus is exactly consistent with the assigned weights. To further reduce any noise in the generated corpora, it is possible to perform manual filtering and preprocessing with the aid of CorpusTools and task-distribution or crowd-sourcing platforms, such as *Amazon Mechanical Turk* [73].

Also worth further consideration is the inter-domain and cross-domain applicability of trained models, which involves determining how well a model trained on a corpus of one topic may be applied to detect errors in sentences of a similar topic, different topic, or more generally, of any other topic. Additionally, a full performance comparison of the various classification algorithms may be useful towards recommending classifiers, or ensembles of classifiers, that are best-suited for the grammar verification methods presented in this research. Since models are generated per grammatical error type and quantity, it is useful to explore the predictive capability of a *collection* of models, such as through voting or ensemble strategies, to make an overall prediction about grammatical correctness.

Finally, an overarching goal for ongoing development is to refine the results of this and future research into a fully-realized and usable grammar verification system. This naturally requires robust and reliable error detection, along with a system for offering corrections and suggestions. Any such system should aim to be adaptable through retraining or updating of its models in response to user feedback. The pinnacle achievement for a grammar verification system should be its practical, reliable, and wide-spread use among natural language users.

Appendix 1

Sentence features

nTokens
number of tokens
the number of tokens (after tokenization) contained in the sentence, including punctuation tokens.

nVerbs
number of verbs
the number of verbs contained in the sentence; a word is considered to be a verb if it is tagged as such by either the part-of-speech tagger or parser of the OpenNLP library.

opennlpParseProb
OpenNLP Parse Probability
the probability of the parse as returned by the OpenNLP parser; the probability is formed from the log of the product of probabilities for the constituents that form the parse tree.

stanfordParseProb
Stanford NLP Parse Probability
the probability of the parse as returned by the Stanford NLP parser, in log-probability.

opennlpDeltaParseProb
OpenNLP delta parse probability
the difference in probability between the best parse and the next-best parse, as given by the OpenNLP parser; if only a single parse exists, the value is 0.

stanfordDeltaParseProb
Stanford NLP delta parse probability
similar to feature *opennlpDeltaParseProb*, except using the parse returned by the Stanford NLP parser.

opennlpTagMismatchTotal
OpenNLP tag mismatch total
the number of part-of-speech tags that differ between the sequence of tags returned by the OpenNLP tagger and those returned by the parser.

stanfordTagMismatchTotal
Stanford NLP tag mismatch total
similar to feature *opennlpTagMismatchTotal*, except using the tags returned by the Stanford NLP tagger and parser.

opennlpTagMismatchRatio
OpenNLP tag mismatch ratio
the ratio of part-of-speech tags that differ between the sequence of tags returned by the OpenNLP tagger and those returned by the parser, to the total number of tags in the sentence. Taken by dividing the value of feature *opennlpTagMismatchTotal* by the value of feature *nTokens*.

stanfordTagMismatchRatio
Stanford NLP tag mismatch ratio
similar to feature *opennlpTagMismatchRatio*, except using the tags returned by the Stanford NLP tagger and parser.

opennlpStanfordCrossDeltaParseProb
OpenNLP-Stanford NLP cross-delta parse probability
the difference between the OpenNLP parse probability and the Stanford NLP parse probability for a sentence.

opennlpStanfordCrossTagMismatchTotal
OpenNLP-Stanford NLP cross-tag mismatch total
the number of part-of-speech tags that differ between the sequence of tags returned by the OpenNLP tagger and those returned by the Stanford NLP tagger.

opennlpStanfordCrossTagMismatchRatio
OpenNLP-Stanford NLP cross-tag mismatch ratio
the ratio of part-of-speech tags that differ between the sequence of tags returned by the OpenNLP tagger and those returned by the Stanford NLP tagger, to the total number of tags in the sentence. Taken by dividing the value of feature *opennlpStanfordCrossTagMismatchTotal* by the value of feature *nTokens*.

opennlpStanfordCrossPTagMismatchTotal
OpenNLP-Stanford NLP cross-parser-tag mismatch total

similar to feature *opennlpStanfordCrossTagMismatchTotal*, except using tags returned by the respective parsers.

opennlpStanfordCrossPTagMismatchRatio
OpenNLP-Stanford NLP cross-parser-tag mismatch ratio
similar to feature *opennlpStanfordCrossTagMismatchRatio*, except using tags returned by the respective parsers.

opennlpStanfordCrossParseMismatchTotal
OpenNLP-Stanford NLP cross-parse mismatch total
the number of parse constituents that differ between the parse tree returned by the OpenNLP parser and the tree returned by the Stanford NLP parser.

opennlpStanfordCrossParseMismatchRatio
OpenNLP-Stanford NLP cross-parse mismatch ratio
the ratio of parse constituents that differ between the parse tree returned by the OpenNLP parser and the tree returned by the Stanford NLP parser, to the total number of compared constituents between the parse trees.

tagSeqProb
tag sequence probability
the probability associated with the sequence of part-of-speech tags as returned by the OpenNLP parser.

deltaTagSeqProb
delta tag sequence probability
the difference between the tag sequence probabilities (feature *tagSeqProb*) of the best and next-best parse, as returned by the OpenNLP parser; if only a single parse exists, the value is 0.

firstTagProb
first tag probability
the probability of the first part-of-speech tag as returned by the OpenNLP tagger.

minTagProb
minimum tag probability
the minimum probability of all POS tags returned by the OpenNLP tagger.

maxTagProb
maximum tag probability
the maximum probability of all POS tags returned by the OpenNLP tagger.

deltaMinTagProb
delta minimum tag probability
the difference between the probability of the tag with minimum probability in the best tagging, and the probability of the tag sharing the same position in the next-best tagging, as returned by the OpenNLP tagger; if only a single tagging exists, the value is 0.

deltaMaxTagProb
delta maximum tag probability
similar to feature *deltaMinTagProb*, except using the tag with maximum probability.

rangeTagProb
range of tag probabilities
the range of tag probabilities: the difference between the maximum and minimum tag probabilities.

firstPTagProb
first parser tag probability
similar to feature *firstTagProb*, except using tags returned by the OpenNLP parser.

minPTagProb
minimum parser tag probability
similar to feature *minTagProb*, except using tags returned by the OpenNLP parser.

maxPTagProb
maximum parser tag probability
similar to feature *maxTagProb*, except using tags returned by the OpenNLP parser.

deltaMinPTagProb
delta minimum parser tag probability
similar to feature *deltaMinTagProb*, except using tags returned by the OpenNLP parser.

deltaMaxPTagProb
delta maximum parser tag probability
similar to feature *deltaMaxTagProb*, except using tags returned by the OpenNLP parser.

rangePTagProb
range of parser tag probabilities
similar to feature *rangeTagProb*, except using tags returned by the OpenNLP parser.

minDeltaTagPTagProb
minimum delta tag-parser-tag probability
the minimum difference between the tag probability returned by the OpenNLP parser and, for the same token, the tag probability returned by the OpenNLP tagger, across all tokens in a sentence.

maxDeltaTagPTagProb
maximum delta tag-parser-tag probability
similar to feature *minDeltaTagPTagProb*, except using the maximum difference.

totDeltaTagPTagProb
total delta tag-parser-tag probability
the total difference between the tag probability returned by the OpenNLP parser and, for the same token, the tag probability returned by the OpenNLP tagger, summed across all tokens in a sentence.

$$\sum_{i}^{n} p(ptag_i) - p(tag_i) \tag{1.1}$$

avgDeltaTagPTagProb
average delta tag-parser-tag probability
the average difference, across all tags in a sentence, between the tag probability returned by OpenNLP's parser and (for the same word) the tag probability returned by OpenNLP's tagger. Taken by dividing the value of feature *totDeltaTagPTagProb* by the value of feature *nTokens*.

$$\sum_{i}^{n} \frac{p(ptag_i) - p(tag_i)}{n} \tag{1.2}$$

minDeltaVerbChangeTagProb
minimum delta verb change tag probability
the minimum difference, across all verbs in a sentence, between the tag probability of a verb and the tag probability of the altered form of the verb that yields the highest tag probability.

maxDeltaVerbChangeTagProb
maximum delta verb change tag probability
similar to feature *minDeltaVerbChangeTagProb*, except using the maximum difference.

totDeltaVerbChangeTagProb
total delta verb change tag probability
the total of the differences, summed over all verbs in a sentence, between the tag probability of a verb and the tag probability of the altered form of the verb that yields the highest tag probability. Equivalently, the sum of the value of feature *maxDeltaVerbChangeTagProb* across all verbs in a sentence.

avgDeltaVerbChangeTagProb
average delta verb change tag probability
the average difference, across all verbs in a sentence, between the tag probability of a verb and the tag probability of the altered form of the verb that yields the highest tag probability. Equivalently, the average value of feature *maxDeltaVerbChangeTagProb* for all verbs in a sentence, taken by dividing the value of feature *totDeltaVerbChangeTagProb* by the value of *nVerbs*.

minDeltaVerbChangePTagProb
minimum delta verb change tag probability
similar to the value of feature *minDeltaVerbChangeTagProb*, except using tags returned by the OpenNLP parser.

maxDeltaVerbChangePTagProb
maximum delta verb change tag probability
similar to the feature *maxDeltaVerbChangeTagProb*, except using tags returned by the OpenNLP parser.

totDeltaVerbChangePTagProb
total delta verb change tag probability
similar to the feature *totDeltaVerbChangeTagProb*, except using tags returned by the OpenNLP parser.

avgDeltaVerbChangePTagProb
average delta verb change tag probability
similar to the feature *avgDeltaVerbChangeTagProb*, except using tags returned by the OpenNLP parser.

minDeltaVerbChangeTagSeqProb
minimum delta verb change tag sequence probability
the minimum difference for all verb changes across a sentence, between the tag sequence probability of the original sentence and the tag sequence probability of the sentence resulting from the altered form of the verb that yields the highest tag sequence probability for the entire sentence.

maxDeltaVerbChangeTagSeqProb
maximum delta verb change tag sequence probability
the maximum difference for all verb changes across a sentence, between the tag sequence probability of the original sentence and the tag sequence probability of the sentence resulting from the altered form of the verb that yields the highest tag sequence probability for the entire sentence.

totDeltaVerbChangeTagSeqProb
total delta verb change tag sequence probability
the total difference of all verb changes across a sentence, between the tag sequence probability of the

original sentence and the tag sequence probability of the sentence resulting from the altered form of the verb that yields the highest tag sequence probability for the entire sentence.

avgDeltaVerbChangeTagSeqProb
average delta verb change tag sequence probability
the average difference of all verb changes across a sentence, between the tag sequence probability of the original sentence and the tag sequence probability of the sentence resulting from the altered form of the verb that yields the highest tag sequence probability for the entire sentence.

minDeltaVerbChangeOpennlpParseProb
minimum delta verb change OpenNLP parse probability
the minimum difference for all verb changes across a sentence, between the parse probability (as returned by the OpenNLP parser) of the original sentence and the parse probability of the sentence resulting from the altered form of the verb that yields the highest parse probability for the entire sentence.

maxDeltaVerbChangeOpennlpParseProb
maximum delta verb change OpenNLP parse probability
the maximum difference for all verb changes across a sentence, between the parse probability (as returned by the OpenNLP parser) of the original sentence and the parse probability of the sentence resulting from the altered form of the verb that yields the highest parse probability for the entire sentence.

totDeltaVerbChangeOpennlpParseProb
total delta verb change OpenNLP parse probability
the total difference of all verb changes across a sentence, between the parse probability (as returned by the OpenNLP parser) of the original sentence and the parse probability of the sentence resulting from the altered form of the verb that yields the highest parse probability for the entire sentence.

avgDeltaVerbChangeOpennlpParseProb
average delta verb change OpenNLP parse probability
the average difference of all verb changes across a sentence, between the parse probability (as returned by the OpenNLP parser) of the original sentence and the parse probability of the sentence resulting from the altered form of the verb that yields the highest parse probability for the entire sentence.

$$\frac{\sum_i^n p(parse(s)) - p(parse(modify(s, v_i)))}{n} \tag{1.3}$$

where n is the number of verbs in the sentence s, and v_i is the verb currently being altered.

minDeltaVerbChangeStanfordParseProb
minimum delta verb change Stanford NLP parse probability

similar to the feature *minDeltaVerbChangeOpennlpParseProb*, except using the parse probability returned from the Stanford NLP parser.

maxDeltaVerbChangeStanfordParseProb
maximum delta verb change Stanford NLP parse probability
similar to the feature *maxDeltaVerbChangeOpennlpParseProb*, except using the parse probability returned from the Stanford NLP parser.

totDeltaVerbChangeStanfordParseProb
total delta verb change Stanford NLP parse probability
similar to the feature *totDeltaVerbChangeOpennlpParseProb*, except using the parse probability returned from the Stanford NLP parser.

avgDeltaVerbChangeStanfordParseProb
average delta verb change Stanford NLP parse probability
similar to the feature *avgDeltaVerbChangeOpennlpParseProb*, except using the parse probability returned from Stanford NLP parser.

totVerbChangeTagImprove
total verb change tag improvements
the total number of times, across all verb changes in a sentence, that altering the form of a verb resulted in an improvement (increase) in its tag probability.

avgVerbChangeTagImprove
average verb change tag improvements
the average of the total number of tag improvements across all verb changes in a sentence, and the total number of verbs for that sentence.

totVerbChangePTagImprove
total verb change tag improvements
similar to the feature *totVerbChangeTagImprove*, except using tags returned by the OpenNLP parser.

avgVerbChangePTagImprove
average verb change tag improvements
similar to the feature *avgVerbChangeTagImprove*, except using tags returned by the OpenNLP parser.

totVerbChangeTagSeqImprove
total verb change tag sequence improvements
the total number of times, across all verb changes in a sentence, that altering the form of a verb resulted in an improvement (increase) in the tag sequence probability of the entire sentence.

avgVerbChangeTagSeqImprove
average verb change tag sequence improvements
the average of the total number of tag sequence improvements across all verb changes in a sentence, and the total number of verbs for that sentence.

totVerbChangeOpennlpParseImprove
total verb change parse improvements
the total number of times, across all verb changes in a sentence, that altering the form of a verb resulted in an improvement (increase) in the parse probability of the entire sentence.

avgVerbChangeOpennlpParseImprove
average verb change parse improvements
the average of the total number of parse improvements across all verb changes in a sentence, and the total number of verbs for that sentence.

totVerbChangeStanfordParseImprove
total verb change parse improvements
similar to the feature *totVerbChangeOpennlpParseImprove*, except using the parse probability returned from the Stanford NLP parser.

avgVerbChangeStanfordParseImprove
average verb change parse improvements
similar to the feature *avgVerbChangeOpennlpParseImprove*, except using the parse probability returned from the Stanford NLP parser.

minVerbChangeImproveRatio
minimum verb change improvement ratio
the minimum ratio, across all verb changes, of the number of improvements (to tag, parser tag, tag sequence, OpenNLP parse, or Stanford NLP parse probabilities) that result from a verb change, divided by the number of improvements possible (a total of five).

maxVerbChangeImproveRatio
maximum verb change improvement ratio
the maximum ratio, across all verb changes, of the number of improvements (to tag, parser tag, tag sequence, OpenNLP parse, or Stanford NLP parse probabilities) that result from a verb change, divided by the number of improvements possible (a total of five).

totVerbChangeImproveRatio
total verb change improvement ratio
the sum of ratios over all verb changes of the number of improvements (to tag, parser tag, tag sequence,

OpenNLP parse, or Stanford NLP parse probabilities) that result from a verb change, divided by the number of improvements possible (a total of five).

avgVerbChangeImproveRatio
average verb change improvement ratio
the average of the ratios, across all verb changes, of the number of improvements (to tag, parser tag, tag sequence, OpenNLP parse, or Stanford NLP parse probabilities) that result from a verb change, divided by the number of improvements possible (a total of five).

deltaMinChangeTagSeqProb
delta minimum change tag sequence probability
the difference between the tag sequence probability of the sentence and the tag sequence probability of the sentence resulting from the altered form of the token with mininum tag probability, that yields the highest tag sequence probability for the entire sentence.

deltaMinChangeOpennlpParseProb
delta minimum change OpenNLP parse probability
the difference between the parse probability (as returned by the OpenNLP parser) of the sentence and the parse probability of the sentence derived from the altered form of the token with minimum tag probability, that yields the highest parse probability for the entire sentence.

deltaMinChangeStanfordParseProb
delta minimum change Stanford NLP parse probability
similar to the feature *deltaMinChangeOpennlpParseProb*, except using the parse probability returned from the Stanford NLP parser.

opennlpDeltaParseProbSwapMinLeft
OpenNLP delta parse probability swap minimum left
the difference between the parse probability (as returned by the OpenNLP parser) of the sentence and the parse probability of the sentence derived from swapping the token with minimum tag probability with the token adjacent to its left.

opennlpDeltaParseProbSwapMinRight
OpenNLP delta parse probability swap minimum right
the difference between the parse probability (as returned by the OpenNLP parser) of the sentence and the parse probability of the sentence derived from swapping the token with minimum tag probability with the token adjacent to its right.

opennlpDeltaParseProbOmitMin
OpenNLP delta parse probability omit minimum

the difference between the parse probability (as returned by the OpenNLP parser) of the sentence and the parse probability of the sentence derived from omitting the token with minimum tag probability.

opennlpDeltaParseProbOmitMinLeft
OpenNLP delta parse probability omit minimum left
the difference between the parse probability (as returned by the OpenNLP parser) of the sentence and the parse probability of the sentence derived from omitting the token adjacent left of the token with minimum tag probability.

opennlpDeltaParseProbOmitMinRight
OpenNLP delta parse probability omit minimum right
the difference between the parse probability (as returned by the OpenNLP parser) of the sentence and the parse probability of the sentence derived from omitting the token adjacent right of the token with minimum tag probability.

stanfordDeltaParseProbSwapMinLeft
Stanford NLP delta parse probability swap minimum left
similar to the feature *opennlpDeltaParseProbSwapMinLeft*, except using the parse probabilities returned from the Stanford NLP parser.

stanfordDeltaParseProbSwapMinRight
Stanford NLP delta parse probability swap minimum right
similar to the feature *opennlpDeltaParseProbSwapMinRight*, except using the parse probabilities obtained from the Stanford NLP parser.

stanfordDeltaParseProbOmitMin
Stanford NLP delta parse probability omit minimum
similar to the feature *opennlpDeltaParseProbOmitMin*, except using the parse probabilities obtained from the Stanford NLP parser.

stanfordDeltaParseProbOmitMinLeft
Stanford NLP delta parse probability omit minimum left
similar to the feature *opennlpDeltaParseProbOmitMinLeft*, except using the parse probabilities obtained from the Stanford NLP parser.

stanfordDeltaParseProbOmitMinRight
Stanford NLP delta parse probability omit minimum right
similar to the feature *opennlpDeltaParseProbOmitMinRight*, except using the parse probabilities obtained from the Stanford NLP parser.

firstTag
first tag
the part-of-speech tag of the first token occurring in the sentence.

minTag
minimum tag
the part-of-speech tag having the lowest probability of all tags for the sentence.

maxTag
maximum tag
the part-of-speech tag having the highest probability of all tags for the sentence.

firstPTag
first parser tag
similar to feature *firstTag*, except using tags returned by the OpenNLP parser.

minPTag
minimum parser tag
similar to feature *minTag*, except using tags returned by the OpenNLP parser.

maxPTag
maximum parser tag
similar to feature *maxTag*, except using tags returned by the OpenNLP parser.

isFirstTagMismatch
is first tag mismatched
whether the tag of the first token occurring in the sentence (produced by the OpenNLP tagger) differs from the tag for that same token, produced by the OpenNLP parser.

isMinTagMismatch
is minimum tag mismatched
whether the tag having lowest probability (from the OpenNLP tagger) differs from the tag having lowest probability as produced by the OpenNLP parser.

isMaxTagMismatch
is maximum tag mismatched
whether the tag having highest probability (from the OpenNLP tagger) differs from the tag having highest probability as produced by the OpenNLP parser.

isOpennlpClause
is OpenNLP clausal
whether the sentence can be parsed into a parse tree (by the OpenNLP parser) containing a clause as a constituent of its root.

isStanfordClause
is Stanford NLP clausal
similar to feature *isOpennlpClause*, except using the Stanford NLP parser.

isDCGParsable
is DCG parsable
whether the sentence can be parsed by a user-specified definite clause grammar (DCG).

s-nClauses
number of simplified clauses
the number of simplified clauses extracted from the sentence.
Sentences are reduced to simplified clauses containing just subjects and objects (determiners, nouns and pronouns) and actions (verbs), from which further features are extracted. These features are prefixed with an *"s-"* in their names, further listed in this section.

s-minOpennlpParseProb
simplified: minimum OpenNLP parse probability
the lowest parse probability (as returned by the OpenNLP parser) of all clauses extracted from the sentence.

s-maxOpennlpParseProb
simplified: maximum OpenNLP parse probability
the highest parse probability (as returned by the OpenNLP parser) of all clauses extracted from the sentence.

s-totOpennlpParseProb
simplified: total OpenNLP parse probability
the sum of parse probabilities (as returned by the OpenNLP parser) of all clauses extracted from the sentence.

s-avgOpennlpParseProb
simplified: average OpenNLP parse probability
the average parse probability (as returned by the OpenNLP parser) across all clauses extracted from the sentence.

s-minStanfordParseProb
simplified: minimum Stanford NLP parse probability
similar to the *s-minOpennlpParseProb*, except using probabilities returned by the Stanford NLP parser.

s-maxStanfordParseProb
simplified: maximum Stanford NLP parse probability
similar to the *s-maxOpennlpParseProb*, except using probabilities returned by the Stanford NLP parser.

s-totStanfordParseProb
simplified: total Stanford NLP parse probability
similar to the *s-totOpennlpParseProb*, except using probabilities returned by the Stanford NLP parser.

s-avgStanfordParseProb
simplified: average Stanford NLP parse probability
similar to the *s-avgOpennlpParseProb*, except using probabilities returned by the Stanford NLP parser.

s-minDeltaVerbChangeTagProb
simplified: minimum delta verb change tag probability
similar to the *minDeltaVerbChangeTagProb* feature, except the minimum is calculated across all clauses.

s-maxDeltaVerbChangeTagProb
simplified: minimum delta verb change tag probability
similar to the *maxDeltaVerbChangeTagProb* feature, except the maximum is calculated across all clauses.

s-totDeltaVerbChangeTagProb
simplified: total delta verb change tag probability
similar to the *totDeltaVerbChangeTagProb* feature, except the total is summed across all clauses.

s-avgDeltaVerbChangeTagProb
simplified: average delta verb change tag probability
similar to the *avgDeltaVerbChangeTagProb* feature, except the average is calculated across all clauses.

s-minDeltaVerbChangePTagProb
simplified: minimum delta verb change parser tag probability
similar to the *minDeltaVerbChangePTagProb* feature, except the minimum is calculated across all clauses.

s-maxDeltaVerbChangePTagProb
simplified: maximum delta verb change parser tag probability
similar to the *maxDeltaVerbChangePTagProb* feature, except the maximum is calculated across all clauses.

s-totDeltaVerbChangePTagProb
simplified: total delta verb change parser tag probability
similar to the *totDeltaVerbChangePTagProb* feature, except the total is summed across all clauses.

s-avgDeltaVerbChangePTagProb
simplified: average delta verb change parser tag probability
similar to the *avgDeltaVerbChangePTagProb* feature, except the average is calculated across all clauses.

s-minDeltaVerbChangeTagSeqProb
simplified: minimum delta verb change tag sequence probability
similar to the *minDeltaVerbChangeTagSeqProb* feature, except the minimum is calculated across all clauses.

s-maxDeltaVerbChangeTagSeqProb
simplified: maximum delta verb change tag sequence probability
similar to the *maxDeltaVerbChangeTagSeqProb* feature, except the maximum is calculated across all clauses.

s-totDeltaVerbChangeTagSeqProb
simplified: total delta verb change tag sequence probability
similar to the *totDeltaVerbChangeTagSeqProb* feature, except the total is summed across all clauses.

s-avgDeltaVerbChangeTagSeqProb
simplified: average delta verb change tag sequence probability
similar to the *avgDeltaVerbChangeTagSeqProb* feature, except the average is calculated across all clauses.

s-minDeltaVerbChangeOpennlpParseProb
simplified: minimum delta verb change OpenNLP parse probability
similar to the *minDeltaVerbChangeOpennlpParseProb* feature, except the minimum is calculated across all clauses.

s-maxDeltaVerbChangeOpennlpParseProb
simplified: maximum delta verb change OpenNLP parse probability
similar to the *maxDeltaVerbChangeOpennlpParseProb* feature, except the maximum is calculated across all clauses.

s-totDeltaVerbChangeOpennlpParseProb
simplified: total delta verb change OpenNLP parse probability
similar to the *totDeltaVerbChangeOpennlpParseProb* feature, except the total is summed across all clauses.

s-avgDeltaVerbChangeOpennlpParseProb
simplified: average delta verb change OpenNLP parse probability
similar to the *avgDeltaVerbChangeOpennlpParseProb* feature, except the average is calculated across all clauses.

s-minDeltaVerbChangeStanfordParseProb
simplified: minimum delta verb change Stanford NLP parse probability
similar to the *minDeltaVerbChangeStanfordParseProb* feature, except the minimum is calculated across all clauses.

s-maxDeltaVerbChangeStanfordParseProb
simplified: maximum delta verb change Stanford NLP parse probability
similar to the *maxDeltaVerbChangeStanfordParseProb* feature, except the maximum is calculated across all clauses.

s-totDeltaVerbChangeStanfordParseProb
simplified: total delta verb change Stanford NLP parse probability
similar to the *totDeltaVerbChangeStanfordParseProb* feature, except the total is summed across all clauses.

s-avgDeltaVerbChangeStanfordParseProb
simplified: average delta verb change Stanford NLP parse probability
similar to the *avgDeltaVerbChangeStanfordParseProb* feature, except the average is calculated across all clauses.

s-totVerbChangeTagImprove
simplified: total verb change tag improvements
similar to the *totVerbChangeTagImprove* feature, except the total is summed across all clauses.

s-avgVerbChangeTagImprove
simplified: average verb change tag improvements
similar to the *avgVerbChangeTagImprove* feature, except the average is calculated across all clauses.

s-totVerbChangePTagImprove
simplified: total verb change parser tag improvements
similar to the *totVerbChangePTagImprove* feature, except the total is summed across all clauses.

s-avgVerbChangePTagImprove
simplified: average verb change parser tag improvements
similar to the *avgVerbChangePTagImprove* feature, except the average is calculated across all clauses.

s-totVerbChangeTagSeqImprove
simplified: total verb change tag sequence improvements
similar to the *totVerbChangeTagSeqImprove* feature, except the total is summed across all clauses.

s-avgVerbChangeTagSeqImprove
simplified: average verb change tag sequence improvements
similar to the *avgVerbChangeTagSeqImprove* feature, except the average is calculated across all clauses.

s-totVerbChangeOpennlpParseImprove
simplified: total verb change OpenNLP parse improvements
similar to the *totVerbChangeOpennlpParseImprove* feature, except the total is summed across all clauses.

s-avgVerbChangeOpennlpParseImprove
simplified: average verb change OpenNLP parse improvements
similar to the *avgVerbChangeOpennlpParseImprove* feature, except the average is calculated across all clauses.

s-totVerbChangeStanfordParseImprove
simplified: total verb change Stanford NLP parse improvements
similar to the *totVerbChangeStanfordParseImprove* feature, except the total is summed across all clauses.

s-avgVerbChangeStanfordParseImprove
simplified: average verb change Stanford NLP parse improvements
similar to the *avgVerbChangeStanfordParseImprove* feature, except the average is calculated across all clauses.

s-minVerbChangeImproveRatio
simplified: minimum verb change improvement ratio
similar to the *minVerbChangeImproveRatio* feature, except the minimum is calculated across all clauses.

s-maxVerbChangeImproveRatio
simplified: maximum verb change improvement ratio
similar to the *maxVerbChangeImproveRatio* feature, except the maximum is calculated across all clauses.

s-totVerbChangeImproveRatio
simplified: total verb change improvement ratio
similar to the *totVerbChangeImproveRatio* feature, except the total is summed across all clauses.

s-avgVerbChangeImproveRatio
simplified: average verb change improvement ratio
similar to the *avgVerbChangeImproveRatio* feature, except the average is calculated across all clauses.

s-minDeltaMinChangeTagSeqProb
simplified: minimum delta minimum change tag sequence probability
the lowest *deltaMinChangeTagSeqProb* of all clauses extracted from the sentence.

s-maxDeltaMinChangeTagSeqProb
simplified: maximum delta minimum change tag sequence probability
the highest *deltaMinChangeTagSeqProb* of all clauses extracted from the sentence.

s-totDeltaMinChangeTagSeqProb
simplified: total delta minimum change tag sequence probability
the total *deltaMinChangeTagSeqProb* across all clauses extracted from the sentence.

s-avgDeltaMinChangeTagSeqProb
simplified: average delta minimum change tag sequence probability
the average *deltaMinChangeTagSeqProb* of all clauses extracted from the sentence.

s-minDeltaMinChangeOpennlpParseProb
simplified: minimum delta minimum change OpenNLP parse probability
the lowest *deltaMinChangeOpennlpParseProb* of all clauses extracted from the sentence.

s-maxDeltaMinChangeOpennlpParseProb
simplified: maximum delta minimum change OpenNLP parse probability
the highest *deltaMinChangeOpennlpParseProb* of all clauses extracted from the sentence.

s-totDeltaMinChangeOpennlpParseProb
simplified: total delta minimum change OpenNLP parse probability
the total *deltaMinChangeOpennlpParseProb* across all clauses extracted from the sentence.

s-avgDeltaMinChangeOpennlpParseProb
simplified: average delta minimum change OpenNLP parse probability
the average *deltaMinChangeOpennlpParseProb* of all clauses extracted from the sentence.

s-minDeltaMinChangeStanfordParseProb
simplified: minimum delta minimum change Stanford NLP parse probability
the lowest *deltaMinChangeStanfordParseProb* of all clauses extracted from the sentence.

s-maxDeltaMinChangeStanfordParseProb
simplified: maximum delta minimum change Stanford NLP parse probability
the highest *deltaMinChangeStanfordParseProb* of all clauses extracted from the sentence.

s-totDeltaMinChangeStanfordParseProb
simplified: minimum delta minimum change Stanford NLP parse probability
the total *deltaMinChangeStanfordParseProb* across all clauses extracted from the sentence.

s-avgDeltaMinChangeStanfordParseProb
simplified: average delta minimum change Stanford NLP parse probability
the average *deltaMinChangeStanfordParseProb* of all clauses extracted from the sentence.

funcValue
function value
the value of a user-defined function of all sentence features defined in this section (excluding this feature itself).

isGrammatical
is grammatical
the boolean class value [1] indicating whether a sentence is grammatically correct (*true*) or incorrect (*false*). The value of this attribute is typically provided by the researcher for training instances, or is predicted by the trained model for new, unknown instances.

[1] a numeric class attribute is also defined by the implementation, allowing the researcher to indicate the number of errors present in a sentence, or alternatively indicate an arbitrary measure of incorrectness or *badness* of a sentence. While some experimentation was done with this numeric class value, it remains largely untreated here, and is instead left for future research.

Appendix 2

Sample corpus

The following (tokenized) sentences have been randomly sampled from the *Verbatim* corpus (of the OANC, explained in subsection 3.2.4), containing approximately 25,000 sentences. Of the 100 sentences sampled, 81 are deemed grammatically correct, followed by 19 grammatically incorrect sentences that may constitute *noise*, including fragments of text that may be incoherent (such as from titles, headings, figures, dates, etc.).

2.1 Grammatically correct sentences

Wisdom on the scone -pronunciation issue was shown by Sir Ernest Gowers in his 1965 revision of H.W. Fowler's A Dictionary of Modern English Usage .
How reassuring to be told by one's smiling, hooded, middleaged woman caddy that one's drive is "safe-o"—that is, not unfindable; or that one's next shot will require a "nine-o" or an "eight-o" or a "wedgie" (if one is in the "sand-o").
Thomas Hardy described an extreme case in Jude the Obscure .
As a noun it was first used of trade dealings between people of different localities.
The looking-glass image of utopia–dystopia–has provided more numerous additions to the English lexicon.
If our press would attain true excellence in selecting English expressions to correspond to Russian originals, it should exercise more care about designating the Soviet Union as "Russia" and Soviets as "Russians."
As a result, many Americans innocently mispronounce the name of the great operatic bass and his descendants.
For one thing, you cannot simply coin a word and lay claim to it.
Since his product works differently from yours, you cannot get him for violating patent law, so you haul him into court and sue the pants off him for infringing on the implied trademark you have established with your Widget Terminator.

To one another?
Still, let the term stand.
Thou preparest a table before me in the presence of mine enemies: Thou anointest my head with oil; my cup runneth over.
My advice is–give it up!
"How did you get here," I was often asked.
Or maybe analog computer used to be simply computer –we live in an age of technological complexity, I concluded, as I bicycled home that day from work.
Rules or lists may change, thus rendering the mnemonics obsolete.
One young girl wore a shirt with "MILK MILK" printed on it corresponding to the appropriate part of her anatomy beneath the shirt.
(He would only be asking to hold the baby.)
It may be argued that more diseases have been discovered than cures (It could hardly be the other way around, could it?)
Hard to say what that means, except stinging eyes, and besides, using half of each of two clichés is simply spilt milk off a duck's back on troubled waters: isn't it "spit and polish" and "piss and vinegar?"
The curious point is that this trend seems to go against Zipf's Law, which notes that syllables get trimmed off words over the years.
Sate and Satiate are perhaps the most blatant instance: the two are synonymous in most dictionaries, the form from the Latin satiare , 'to satisfy.'
Each spelled sound is discussed at the beginning of its alphabetic section, digraphs being treated at the beginnings of the sections for their first letter.
In a very general sense the Puritans can be said to have added a few names to the stock which modern parents draw upon, but the author obviously did not mean that.
I am beginning to find it irksome to have to point out every few years that Thomas Crapper did indeed live and that the company he founded lasted until about 1928.
Naturally, nothing is more reassuring to a reader.
The first 40 or so pages are devoted to an excellent essay, titled "Historical Sketch."
I had it now, of course.
In a context that was appropriate but not worth describing, a refined American lady was heard to exclaim with some surprise, "I didn't know they had prostitutes in Poland!"
It turns out to have been Sir Thomas Henry Hall Caine, Stoker's friend; although the significance of that revelation eludes me completely, it might well have won some bookworm a beer.
Could your contributor, notwithstanding his correct identification of the background to Three Men on the Bummel , be confusing it with the better-known Three Men in a Boat , where the scenery is decidedly fluvial?
Visitors from the North always demanded a visit to the nearby tourist attraction, a lush, tropical paradise populated by trained parrots and alligators and colorful insects and such.
Touring the tourist attraction's aviary–in which the birds flew free while visitors strolled in a glass tunnel–I hit my stride.

The Omni Gazetteer is available in two editions, as eleven bound volumes and on CD-ROM.
There is no word for argument in Spanish, either, in that sense of the word.
One wonders whether the author is the Bill Bryson who wrote The Lost Continent–Travels in Small Town America .
Equally famous (though as an Irish bull) is his question of a former student shortly after World War I: "Was it you or your brother who was killed in the war?" Spooner admitted "occasional infelicities in verbal diction" but became openly irritated when his name was associated with oral transpositions.
The Solomonic decision, to award custody to the woman who was willing to give the child to her rival rather than see it killed by being split between them was just the opposite of splitting the baby .
Hanks and Hodges and de Felice say that most of these nicknames could also be bestowed ironically.
There followed numerous recyclings of this sentence with an intertextual reference that was less and less to the Fry play and more and more to the occasion of the conference speech.
At any rate, he never seemed to bother about brows, lids, and lashes, although his attitude was sometimes supercilious.
I do not think Malcolm has ever seen a small child break the single-word language barrier and get hold of syntax, a fundamental breakthrough in language learning.
The wicked old witch at last is dead!
Of course, she never quite conceded defeat: hope springs eternal in the human breast.
In an earlier issue of VERBATIM writers have commented on what have come to be called bacronyms, names of companies, products, charities, and other institutions that have been chosen solely because their acronyms yield words that are pronounceable, meaningful, or both.
F.R. captures the youthful slang of that innocent era when life consisted of countless exclamation marks.
Recently, the head of the California Bar Association delivered an address at the annual meeting decrying jokes about lawyers, suggesting that a man who had raided a law office and killed some people in it had been inspired or spurred on by the derisive attitude toward lawyers that "lawyer jokes" fostered.
Names of apples like cleopatra and democrat, of potatoes like bintje, black derwent, and kennebec, carry a Tasmanian stamp (in Australia at any rate), as does the mutton bird (the shearwater, Puffinus tenuirostris)—which has given Tasmanian English the verb mutton bird or in its abbreviated form bird, as well as lexical oddities like dizz ('cook a mutton bird') for those foolhardy enough to contemplate such a feat.
The indigenous flora and fauna likewise have been carefully marketed to tourists as different : the Tasmanian tiger (now probably extinct) lives in legend and forlorn reaches of the remotest parts of the island, the Tasmanian devil (a carnivorous marsupial of undeniably fierce appearance) in tired zoos and nature parks where nonexclusive animals like the kangaroo and koala share the honors with a remarkable range of diminutive marsupials genuinely peculiar to Tassie, which have added value in an age that cultivates the notion of wilderness.
The fourth volume listed hyphenated forms in the same way.
True, there is not much subtlety in calling a gun a cannon , a ship a canoe , or an incompetent prizefighter a canvasback ; but that a canoe inspector is a gynecologist is not immediately obvious, that Cement City might be thought picturesquely oblique for a cemetery, and that chain lightning is certainly as

descriptive as rotgut for cheap potent liquor cannot escape anyone's notice.
The leader shouts "Taking in" when he reaches the top of a pitch and begins gathering in the slack.
Zane Grey (1875-1939) and James Frank Dobie (1888-1964) might have used the phrase in their writings, but evidence is lacking.
I'm beginning to believe it.
Since almost any grammatical rule that can be broken in any language is sometimes broken, I cannot say that McCarthy has never heard a Mexican say Para todo, meaning for the whole [purchase]?
An impressive and thoroughly engaging addition to Shakespeare scholarship, Professor Marvin Spevack's A Shakespeare Thesaurus deserves to take its place in the canon of eminently useful books about the world's greatest dramatist.
At that moment I would have bet my very soul that he hailed from The Land Of Rabbie Burns, and the first words he uttered showed the intuition inherited from my mystic forebears to be firing on all cylinders.
We watch out for the S.Q.
Catting was a word for chasing after the female sex, i.e., a man out on the tiles–like a tomcat?
How can Dan Rather or Peter Jennings let his audience know when a celebrity suffers eponymously?
The speaker, the scriptwriter, and their audience might have thought that President and Mrs. Bush were ill with Grave disease.
It was the pervasive Anglo-Saxon who really set the foundation for the dialect spoken by the modern descendants of Wigtown Man.
The roads were a sea of mud and now, with night approaching, we were stuck, finally and irrevocably, in the middle of the rainforest.
(Ajax is a cleanser.)
Thus I had him off for HBI.
Mr. Champlin [On Good Terms, XXII, 1, 11] might be interested to know that in this part of the world, South of Manchester, in the market town of Altrincham, a custom called Beating the Bounds takes place every year.
In Quebec, it is taken for granted that English affects French.
Terms like Monomantic Syntoraxis and Tempofluxion Dogma reflect his droll skepticism toward religion.
All these slogans are wrought in attractive designs and lettering and are not intended to be read, so I would like to express my thanks to those puzzled Japanese parents who allowed me to do so.
In any event, Ventris did, finally, decode Linear B, but died (at thirty-four) before arriving at a translation of Linear A. If susceptible teenagers watched that program, perhaps they will be inspired to work on Linear A, still an enigma, owing largely to the paucity of corpus.
In addition, the central issues and tenets of lexical phonology are set out and critically evaluated.
Each of his punstruments creatively cannibalizes and recycles preexisting materials, all that comes to hand.
So salve your conscience, and save your soul, by buying it.
The territory that preceded the state of Utah (named after the Ute Indians) was called the Territory of Desert .

In a further feature on The Malaysian Affair (March 13, 1994), the allegation was repeated and the article further suggested that he might have had a corrupt connection with a $600m banking fraud in Hong Kong, the death of a bank auditor and the false imprisonment of a Malaysian businessman as part of a cover-up.
For the Fon people of Benin the appropriate phrase is Night has fallen or The King has returned to Allada .
Place names and celestial bodies have been a fruitful source of names for both natural and synthetic elements.
Judging by the quality of the 1,002 QRs archived in Madison, we—INFs and FWs alike—must have compensated for our limitations pretty well.
One fellow always greeted me with, Arkansas!
Cassidy offered an intriguing explanation: the metaphor probably originated in Caribbean folklore, which has it that sea turtles (the referent for caouane in standard French dictionaries) copulate for a hundred days at a time.
If it comes to that, even uptown and downtown aren't quite equivalent because downtown in many towns is only a few blocks.
Repeat was broken into re- and -peat and -peat was combined with three.

2.2 Grammatically incorrect sentences

Favorite Grammatical Game: Puzzling Pronouns Here is a game just made to while away the hours on a commuter train with your favorite author, a perfect place to hunt for Puzzling Pronouns .
– Trying once more to attract the ear of Marvin the daydreamer, his teacher says, "Earth to Marvin. What does that word should mean?
[Headline in the Lompoc (California) Record, .
Submitted by] The Communication Ravine For nearly eight years my husband and I worked in Papua New Guinea as teachers in a church school.
Specialties A.G.
Submitted by] "There is no residency requirement for US Senate other than that the candidate be a resident of the state he is running from at the time of his election."
offer Seperate offer (DeMars v. LaOur, 123 Wis.
petticoat (n .)
Curse John - In 1774 Philip Fithian was reminiscing in his diary about his undergraduate days at Princeton when "they often practised mischief by parading bad women and burning Curse John."
Dictionaries for Advanced Learners and Users of Foreign Languages It is widely acknowledged that complaints about missing items in the range of dictionaries required by advanced learners and users of foreign languages are justified to a very considerable extent.
Malcolm My friend Malcolm is a born disciplinarian, and language annoys him no end.
Riak means 'to call.' cheque spring bounced cheque.

Submitted by .]
= euphemism; Sl. = slang; Obs.
borrowing esprit d'escalier estuary, an arm of the sea; a only to translate it into the vapour bath.
or smash a rag? Suppose you duff? or nose and lag?
mump and gag–grimace.
(from British and US trade names.)

Appendix 3

Sample error corpus

The following (tokenized) sentences have been randomly sampled from an automatically generated error corpus containing 3 *tense* errors (described in Section 3.2.3) per sentence. The sentences have been generated by CorpusTools (subsection 3.2.3), using the *Verbatim* corpus (of the OANC, subsection 3.2.4) as input. Of the 20 sentences sampled, all are deemed grammatically incorrect or incoherent.

3.1 Generated error sentences

It would be a shame to losing the etymology burying in the original spelling , though it would being a boon for easy pronunciation .
[From The Times , 15 January 1988] Letter to the Editor of The Times Penny Perrick regrets (January 11) that “There be no word in English to describing that particular , special sort of pride that one feel in the achievements of one ’s children . ”
Otto Rotcod was bearing to becoming the palindrome make flesh .
Whereas a document be reading , a speech be heard .
But that do not meaning that such a square cannot existing .
Through facilities making available by AT&T Bell Laboratories of Murray Hill , New Jersy , all pronunciations be actually hearing using speech-synthesis technology .
Maledicta 9 , Since 1977 , Dr. Reinhold Aman have being publish Maledicta , subtitled The International Journal of Verbal Aggression , from his bivouac in Waukesha , Wisconsin .
Several reliable sources claim familiarity to the 1960s and 1970s , and we be able to confirming this with a 1974 citation from the musician David Bowie .
Before your tour you get into your uniform , include your bonnet (helmet–but never the flat cap used on motorized patrol) , your stick (truncheon , including the new American-style nightsticks) , and your uniform jacket (not strictly a jargon term , but another case of police-public divide , in that the public nearly always calling it a tunic , which is never using by the police themselves) .
People without lexicographic training who undertake to compiling a dictionary running the risk of com-

mitting innumerable blunders of the kind that only the seasoned professional know how to avoid .
The only thing I know about Gilbert be that I finding him listing as co-author of Muskrat Ramble .
The entry for golden-syrup giving a cross-reference to syrup , where it is not mentioned ; as I understanding the style , this means that the two are to be construeding as synonyms , which is not the case .
A deep reclining chair we buy at an estate sale his children hold after an elderly gent 's death is calling Dead Man 's Gulch .
Imagining my frisson when , after only a few pages into Chapter 1 , I come upon this exchange between characters : What 's happen ?
It takes no more than a preliminary skim to suspecting what is soon substantiated in the ample sources and usages that accompanying most entries—that it is within the military services of the United States and Great Britain , from the times of the World Wars , that the F-words having their truest homes .
A recent movie make the name seeming ominous but the Indians probably consider the priests more ridiculous than sinister .
Growing up in the rural Ozarks , I learned early that good country people expecting visitors to sit and visiting a spell before bring up their business .
Random House dictionaries , from the ACD onward , having always been knowing for their ease of use and the clarity and understandability of the information presenting .
I might adding artificial intelligence just to making it harder , but why making the issue more confused than it is already ?
Celebrants begin plan next year 's parade the day after this year 's be over .

References

[1] Øistein E Andersen. Grammatical error prediction. *Technical Report, University of Cambridge, Computer Laboratory*, (794):153, 2011.

[2] James L Peterson. Computer programs for detecting and correcting spelling errors. *Communications of the ACM*, 23(12):676–687, 1980.

[3] Daniel Kies. Evaluating grammar checkers: A comparative ten-year study. In *6th International Conference on Education and Information Systems, Technologies and Applications: EISTA*, 2008.

[4] Tim McGee and Patricia Ericsson. The politics of the program: Ms word as the invisible grammarian. *Computers and Composition*, 19(4):453–470, 2002.

[5] Daniel Naber. A rule-based style and grammar checker. 2003.

[6] Johan Carlberger, Rickard Domeij, Viggo Kann, and Ola Knutsson. The development and performance of a grammar checker for swedish: A language engineering perspective. *Natural language engineering*, 1(1):000–000, 2004.

[7] Jenny R Saffran. Statistical language learning mechanisms and constraints. *Current directions in psychological science*, 12(4):110–114, 2003.

[8] Martin A Nowak, Natalia L Komarova, and Partha Niyogi. Evolution of universal grammar. *Science*, 291(5501):114–118, 2001.

[9] Ian H Witten and Eibe Frank. *Data Mining: Practical machine learning tools and techniques*. Morgan Kaufmann, 2005.

[10] Pierre M Nugues. *An introduction to language processing with perl and prolog*. Springer, 2006.

[11] Julie Franck, Gabriella Vigliocco, and Janet Nicol. Subject-verb agreement errors in french and english: The role of syntactic hierarchy. *Language and Cognitive Processes*, 17(4):371–404, 2002.

[12] Sylviane Granger, Estelle Dagneaux, Fanny Meunier, and Magali Paquot. International corpus of learner english v2. 2009.

[13] Anthony Penniston. Grammartools. `http://grammar-tools.com`, 2013.

[14] Anthony Penniston and Eric Harley. Attempts to verify written english. In *Proceedings of The Fourth International C* Conference on Computer Science and Software Engineering*, pages 121–128. ACM, 2011.

[15] Christopher D Manning and Hinrich Schütze. *Foundations of statistical natural language processing*. MIT press, 1999.

[16] Mitchell P Marcus, Mary Ann Marcinkiewicz, and Beatrice Santorini. Building a large annotated corpus of english: The penn treebank. *Computational linguistics*, 19(2):313–330, 1993.

[17] Eric Brill. Transformation-based error-driven learning and natural language processing: A case study in part-of-speech tagging. *Computational linguistics*, 21(4):543–565, 1995.

[18] Adam L Berger, Vincent J Della Pietra, and Stephen A Della Pietra. A maximum entropy approach to natural language processing. *Computational linguistics*, 22(1):39–71, 1996.

[19] Adwait Ratnaparkhi. A maximum entropy model for part-of-speech tagging. In *Proceedings of the conference on empirical methods in natural language processing*, volume 1, pages 133–142. Philadelphia, PA, 1996.

[20] Adwait Ratnaparkhi. *Maximum entropy models for natural language ambiguity resolution*. PhD thesis, University of Pennsylvania, 1998.

[21] Eugene Charniak. Statistical parsing with a context-free grammar and word statistics. *AAAI/IAAI*, 2005:598–603, 1997.

[22] Dan Klein and Christopher D Manning. Accurate unlexicalized parsing. In *Proceedings of the 41st Annual Meeting on Association for Computational Linguistics-Volume 1*, pages 423–430. Association for Computational Linguistics, 2003.

[23] Noam Chomsky. *Syntactic structures*. Walter de Gruyter, 2002.

[24] Antje Helfrich and Bradley Music. Design and evaluation of grammar checkers in multiple languages. In *Proceedings of the 18th conference on Computational linguistics-Volume 2*, pages 1036–1040. Association for Computational Linguistics, 2000.

[25] WhiteSmoke. Whitesmoke. `http://www.whitesmoke.com`, 2013.

[26] Ginger. Ginger Grammar Checker. `http://www.gingersoftware.com/grammarcheck`, 2013.

[27] Diane Nicholls. The cambridge learner corpus: Error coding and analysis for lexicography and elt. In *Proceedings of the Corpus Linguistics 2003 conference*, pages 572–581, 2003.

[28] Nancy Ide and Catherine Macleod. The american national corpus: A standardized resource of american english. In *Proceedings of Corpus Linguistics 2001*, volume 3, 2001.

[29] The ANC Project. The Open ANC. `http://www.anc.org/data/oanc`, 2013.

[30] Norma A Pravec. Survey of learner corpora. *ICAME journal*, 26(81):114, 2002.

[31] Sylviane Granger. A bird's-eye view of learner corpus research. *Computer learner corpora, second language acquisition and foreign language teaching*, pages 3–33, 2002.

[32] The Apache OpenNLP Project. The Apache OpenNLP Library. `http://opennlp.apache.org`, 2013.

[33] The Stanford NLP Group. Stanford NLP Software. `nlp.stanford.edu/software`, 2013.

[34] Kristina Toutanova, Dan Klein, Christopher D Manning, and Yoram Singer. Feature-rich part-of-speech tagging with a cyclic dependency network. In *Proceedings of the 2003 Conference of the North American Chapter of the Association for Computational Linguistics on Human Language Technology-Volume 1*, pages 173–180. Association for Computational Linguistics, 2003.

[35] James Gosling. *The Java language specification*. Addison-Wesley Professional, 2000.

[36] Ehud Reiter, Robert Dale, and Zhiwei Feng. *Building natural language generation systems*, volume 33. MIT Press, 2000.

[37] Albert Gatt and Ehud Reiter. Simplenlg: A realisation engine for practical applications. In *Proceedings of the 12th European Workshop on Natural Language Generation*, pages 90–93. Association for Computational Linguistics, 2009.

[38] Eugene Charniak, Don Blaheta, Niyu Ge, Keith Hall, John Hale, and Mark Johnson. Bllip 1987-89 wsj corpus release 1. *Linguistic Data Consortium, Philadelphia*, 2000.

[39] Guy Aston and Lou Burnard. *The BNC handbook*. Edinburgh University Press Edinburgh, 1998.

[40] Nancy Ide and Keith Suderman. The american national corpus first release. In *LREC*. Citeseer, 2004.

[41] W Nelson Francis and Henry Kucera. Brown corpus manual. *Brown University Department of Linguistics*, 1979.

[42] Nancy Ide, Collin Baker, Christiane Fellbaum, and Charles Fillmore. Masc: The manually annotated sub-corpus of american english. 2008.

[43] Rebecca J Passonneau, Collin Baker, Christiane Fellbaum, and Nancy Ide. The masc word sense sentence corpus. In *Proceedings of LREC*, 2012.

[44] The ANC Project. The Manually Annotated Sub-Corpus. `http://www.anc.org/data/masc`, 2013.

[45] Kathryn Bock and Kathleen M Eberhard. Meaning, sound and syntax in english number agreement. *Language and Cognitive Processes*, 8(1):57–99, 1993.

[46] F Yates and PM Grundy. Selection without replacement from within strata with probability proportional to size. *Journal of the Royal Statistical Society. Series B (Methodological)*, pages 253–261, 1953.

[47] Nancy Ide, Christiane Fellbaum, Collin Baker, and Rebecca Passonneau. The manually annotated sub-corpus: a community resource for and by the people. In *Proceedings of the ACL 2010 Conference Short Papers*, pages 68–73. Association for Computational Linguistics, 2010.

[48] Mark Hall, Eibe Frank, Geoffrey Holmes, Bernhard Pfahringer, Peter Reutemann, and Ian H Witten. The weka data mining software: an update. *ACM SIGKDD explorations newsletter*, 11(1):10–18, 2009.

[49] Tim Bray, Jean Paoli, C Michael Sperberg-McQueen, Eve Maler, and François Yergeau. Extensible markup language (xml). *World Wide Web Consortium Recommendation REC-xml-19980210. http://www. w3. org/TR/1998/REC-xml-19980210*, 1998.

[50] William F Clocksin, Christopher S Mellish, and WF Clocksin. *Programming in PROLOG*, volume 4. Springer, 1987.

[51] P Moura et al. Iso/iec dtr 13211 3: 2006. *Definite Clause Grammar Rules, Draft, April*, 1:2010, 2010.

[52] Enrico Denti, Andrea Omicini, and Alessandro Ricci. tuprolog: A light-weight prolog for internet applications and infrastructures. In *Practical Aspects of Declarative Languages*, pages 184–198. Springer, 2001.

[53] European Computer Manufacturers Association and others. Ecmascript language specification, 1999.

[54] Norris Boyd et al. Rhino: Javascript for java. *Mozilla Foundation*, 2007.

[55] Robert C Holte. Very simple classification rules perform well on most commonly used datasets. *Machine learning*, 11(1):63–90, 1993.

[56] John Ross Quinlan. *C4. 5: programs for machine learning*, volume 1. Morgan kaufmann, 1993.

[57] Niels Landwehr, Mark Hall, and Eibe Frank. Logistic model trees. *Machine Learning*, 59(1-2):161–205, 2005.

[58] Wayne Iba and Pat Langley. Induction of one-level decision trees. In *ML*, pages 233–240. Citeseer, 1992.

[59] Ron Kohavi. The power of decision tables. In *Machine Learning: ECML-95*, pages 174–189. Springer, 1995.

[60] William Cohen et al. Fast effective rule induction. In *Twelfth International Conference on Machine Learning*, pages 115–123, 1995.

[61] Eibe Frank and Ian H Witten. Generating accurate rule sets without global optimization. 1998.

[62] Remco R Bouckaert. *Bayesian network classifiers in weka*. Department of Computer Science, University of Waikato, 2004.

[63] George H John and Pat Langley. Estimating continuous distributions in bayesian classifiers. In *Proceedings of the Eleventh conference on Uncertainty in artificial intelligence*, pages 338–345. Morgan Kaufmann Publishers Inc., 1995.

[64] Saskia Le Cessie and JC Van Houwelingen. Ridge estimators in logistic regression. *Applied statistics*, pages 191–201, 1992.

[65] Marc Sumner, Eibe Frank, and Mark Hall. Speeding up logistic model tree induction. In *Knowledge Discovery in Databases: PKDD 2005*, pages 675–683. Springer, 2005.

[66] Léon Bottou. Large-scale machine learning with stochastic gradient descent. In *Proceedings of COMPSTAT'2010*, pages 177–186. Springer, 2010.

[67] Corinna Cortes and Vladimir Vapnik. Support-vector networks. *Machine learning*, 20(3):273–297, 1995.

[68] John Platt et al. Sequential minimal optimization: A fast algorithm for training support vector machines. 1998.

[69] Yoav Freund and Robert E Schapire. Large margin classification using the perceptron algorithm. *Machine learning*, 37(3):277–296, 1999.

[70] Nitesh V. Chawla, Kevin W. Bowyer, Lawrence O. Hall, and W. Philip Kegelmeyer. Smote: synthetic minority over-sampling technique. *arXiv preprint arXiv:1106.1813*, 2011.

[71] Wei Fan, Salvatore J Stolfo, Junxin Zhang, and Philip K Chan. Adacost: misclassification cost-sensitive boosting. In *ICML*, pages 97–105. Citeseer, 1999.

[72] James E Baker. Reducing bias and inefficiency in the selection algorithm. In *Proceedings of the second international conference on genetic algorithms*, pages 14–21, 1987.

[73] Amazon.com. Mechanical Turk. `http://www.mturk.com`, 2013.

Glossary

1R One Rule. 42, 44, 45, 48, 49, 51–56, 59, 60

American National Corpus . 24, 31

BN BayesNet. 42, 44, 45, 47–49, 51–57, 59, 60

Cambridge Learner Corpus . 17

corpus . 3, 5, 17, 22–25, 31, 32, 67

CorpusTools . 3, 25, 26, 29, 30, 32, 67, 69, 97

definite clause grammar . 34, 83

DS DecisionStump. 42, 44, 45, 47–49, 51–57, 59, 60

DT DecisionTable. 42–45, 48, 49, 51, 52, 54–56, 59, 60, 62

GrammarTools . 3, 4, 33, 35, 40, 68

grammatically parallel corpus . 3, 25, 31, 32, 67

International Corpus of Learner English . 24

J48 . 42, 44, 45, 48, 49, 51, 52, 54–56, 58–60, 62, 63

JRip . 42, 44, 45, 47–56, 58–60, 62, 63

LMT Logistic Model Tree. 42–49, 51–60

Logistic . 42, 44, 45, 47–49, 51, 52, 54–56, 59, 60

Manually Annotated Sub-Corpus . 24, 31, 32

natural language generation . 23, 25, 26

natural language processing . v, 2–5, 8, 10, 14, 19, 21, 22, 33, 35, 36, 67, 68

NB Naive Bayes. 42, 44, 45, 48, 49, 51, 52, 54–56, 59, 60

Open American National Corpus . 19, 24, 31

OpenNLP . 22, 23, 26, 33, 71

PART Partial Decision Trees. 42, 44, 45, 48, 49, 51–56, 58–60

part of speech . 2, 6

REPTree Reduced-Error Pruning Tree. 42, 44, 45, 47–49, 51, 52, 54–56, 59, 60

SGD Stochastic Gradient Descent. 42, 44, 45, 48, 49, 51, 52, 54–56, 59, 60, 62, 63

SL SimpleLogistic. 42–60, 62, 63

SMO Sequential Minimal Optimization. 42–45, 47–60, 62, 63

Stanford NLP . 22, 23, 26, 71

VP Voted Perceptron. 42, 44–46, 48, 49, 51, 52, 54–57, 59, 60

Acronyms

ANC American National Corpus. 24, 31

CLC Cambridge Learner Corpus. 17, 22, 24

DCG definite clause grammar. 33, 34, 83

GPC grammatically parallel corpus. 25

ICLE International Corpus of Learner English. 24

MASC Manually Annotated Sub-Corpus. 24, 31, 32, 36, 40, 56

NLG natural language generation. 23, 25, 26

NLP natural language processing. v, 2–5, 8–10, 14, 19, 21–23, 33, 35, 36, 67–69

OANC Open American National Corpus. 19, 24, 25, 31, 32, 36, 40, 42, 46, 49, 56, 91, 97

PCFG probabilistic context-free grammar. 10–12, 23

POS part-of-speech. 5, 22

POS part of speech. 2, 6, 7, 12, 13, 15, 16, 19, 23, 26, 27, 32