UNIVERSITY OF
CAMBRIDGE

Automatic annotation of error types for grammatical error correction

Christopher Jack Bryant

Churchill College

This dissertation is submitted in December 2018 for the degree of Doctor of Philosophy

DECLARATION

This dissertation is the result of my own work and includes nothing which is the outcome of work done in collaboration except as declared in the Preface and specified in the text. It is not substantially the same as any that I have submitted, or am concurrently submitting, for a degree or diploma or other qualification at the University of Cambridge or any other University or similar institution except as declared in the Preface and specified in the text. I further state that no substantial part of my dissertation has already been submitted, or is being concurrently submitted, for any such degree, diploma or other qualification at the University of Cambridge or any other University or similar institution except as declared in the Preface and specified in the text. This dissertation does not exceed the prescribed limit of 60 000 words.

Christopher Jack Bryant
19th December, 2018

SUMMARY

Automatic annotation of error types for grammatical error correction

Christopher Jack Bryant

Grammatical Error Correction (GEC) is the task of automatically detecting and correcting grammatical errors in text. Although previous work has focused on developing systems that target specific error types, the current state of the art uses machine translation to correct all error types simultaneously. A significant disadvantage of this approach is that machine translation does not produce annotated output and so error type information is lost. This means we can only evaluate a system in terms of overall performance and cannot carry out a more detailed analysis of different aspects of system performance.

In this thesis, I develop a system to automatically annotate parallel original and corrected sentence pairs with explicit edits and error types. In particular, I first extend the Damerau-Levenshtein alignment algorithm to make use of linguistic information when aligning parallel sentences, and supplement this alignment with a set of merging rules to handle multi-token edits. The output from this algorithm surpasses other edit extraction approaches in terms of approximating human edit annotations and is the current state of the art. Having extracted the edits, I next classify them according to a new rule-based error type framework that depends only on automatically obtained linguistic properties of the data, such as part-of-speech tags. This framework was inspired by existing frameworks, and human judges rated the appropriateness of the predicted error types as 'Good' (85%) or 'Acceptable' (10%) in a random sample of 200 edits. The whole system is called the ERRor ANnotation Toolkit (ERRANT) and is the first toolkit capable of automatically annotating parallel sentences with error types.

I demonstrate the value of ERRANT by applying it to the system output produced by the participants of the CoNLL-2014 shared task, and carry out a detailed error type analysis of system performance for the first time. I also develop a simple language model based approach to GEC, that does not require annotated training data, and show how it can be improved using ERRANT error types.

ACKNOWLEDGEMENTS

First and foremost, this thesis is dedicated to my grandfather, Jack Bryant, who sadly passed away in 2017 before I could submit. Although he never obtained a degree of his own, he nevertheless published several academic papers during his career (Bryant, 1962, 1963; Bryant et al., 1967), and was always amused to receive letters addressed to 'Dr. Bryant' despite his lack of formal qualification. Although I may be the first official 'Dr. Bryant' in the family, I very much consider it to be in his honour.

I am thus grateful to my supervisor, Ted Briscoe, for not only giving me the opportunity to pursue this doctorate, but also for providing a relaxed atmosphere in which to do it. I genuinely always felt completely free to explore my own ideas, yet was also confident he would keep me on track. I wish him fair winds and following seas!

I would similarly like to thank Paula Buttery, Jennifer Foster and Ann Copestake for their various advice and suggestions during my first year report and viva, and also Hwee Tou Ng for originally getting me interested in grammatical error correction. I also acknowledge the Institute for Automated Language Teaching and Assessment (ALTA), the Cambridge University Computer Laboratory, and Churchill College for their generosity and willingness to fund me over the past three and a half years.

This thesis would not have been possible without the support of my many friends and colleagues throughout Cambridge. I would particularly like to thank Øistein Andersen, Theresa Biberauer, Youmna Farag, Mariano Felice, Jack Hodkinson, Sri Jagannathan, Ekaterina Kochmar, Russell Moore, Rupesh Patil, Marek Rei, Yiannos Stathopoulos, Menglin Xia, Helen Yannakoudakis, Zheng Yuan, Ahmed Zaidi, and Meng Zhang, for their conversations and various company during conferences, cinema trips, game nights, and in general. They each made my PhD journey a little more pleasant. Special thanks go to Mariano Felice, who additionally had to put up with my endless Skype messages at all hours and whom I consider a close friend.

I also feel very fortunate to have met my partner, Li Nguyen, during my time in Cambridge. Her constant love and Choco Pies have been an endless source of encouragement that unquestionably made the writing-up process more enjoyable.

I finally want to thank my parents, Cliff and Wendy, for teaching me the importance of education (and so much more), and also my brother, Paul, for the games and gallivants.

Thank you all.

CONTENTS

List of Abbreviations

BLEU	BiLingual Evaluation Understudy
CLC	Cambridge Learner Corpus
CoNLL	Conference on Natural Language Learning
EFL	English as a Foreign Language
ERRANT	ERRor ANnotation Toolkit
F	F-score
FCE	First Certificate in English
FN	False negative
FP	False positive
GEC	Grammatical Error Correction
GLEU	Generalised Language Evaluation Understanding
HOO	Helping Our Own
JFLEG	Johns Hopkins FLuency-Extended GUG
L1	First language
L2	Second language
LM	Language Model
M2	MaxMatch scorer/format
MT	Machine Translation
NLP	Natural Language Processing
NMT	Neural Machine Translation
NUCLE	National University of Singapore Corpus of Learner English
P	Precision
POS	Part of speech
R	Recall
SMT	Statistical Machine Translation
SVA	Subject-verb agreement
TN	True negative
TP	True positive

CHAPTER 1

INTRODUCTION

Grammatical Error Correction (GEC) is the task of automatically detecting and correcting grammatical errors in text. For example, given the sentence 'This informations were very useful', the goal of a correction system would be to output the correct sentence 'This information was very useful'. The main application of a correction system is thus to assist humans with their writing. In particular, native (L1) and non-native (L2) speakers both sometimes make mistakes, and so a GEC system can help automatically detect and correct these mistakes.

This thesis primarily explores GEC in the context of non-native speakers of English. This is because approximately 744 million people speak English as second language (compared to 378 million native speakers) (Simons and Fennig, 2018), and some sources even predict up to 2 billion speakers of English 'at a useful level' by 2020 (Howson, 2013). There is consequently a growing need to develop new pedagogical technologies that can assist both students and teachers. A GEC system could thus not only provide students with instant feedback without having to wait for a teacher, but also reduce teacher workload and keep track of student progress.

1.1 Motivations

Motivated by this need, GEC has gained increased attention in recent years, thanks mainly to a series of shared tasks between 2011 and 2014 that encouraged researchers to build and evaluate their own correction systems on a common dataset (Dale and Kilgarriff, 2011; Dale et al., 2012; Ng et al., 2013, 2014). Although this undoubtedly led to significant progress in the field, the shared tasks also revealed weaknesses in the way systems were evaluated, which in turn resulted in the development of new metrics (Felice and Briscoe, 2015; Napoles et al., 2015).

Despite these developments however, the new metrics were only able to evaluate a system in terms of overall performance and could not produce more detailed scores for different aspects of performance. This is significant because a robust specialised system may actually be more desirable than a mediocre general system; for example a system that scores 90% on determiner

errors but 5% on all other error types may actually be more useful than a system that scores 40% on all error types. Without an error type analysis however, this information is completely unknown.

The main reason a detailed error type analysis has not been done before in GEC is because system output is usually unannotated. In particular, there is a mismatch between an unannotated hypothesis and an annotated reference. Manual annotation is a slow, laborious and expensive process however, and so it is impractical to manually annotate a hypothesis at every stage of system development. Automatic annotation, on the other hand, is a much more attractive prospect, and is hence the main topic of this thesis.

In addition to facilitating detailed error type evaluation, automatic annotation also has several other advantages. For example, it can be used to simplify the annotation of new data and even standardise existing corpora. In particular, human annotators will no longer need to learn the intricacies of complicated error type frameworks when annotating data and can instead focus entirely on just correcting the text. Similarly, existing corpora that were annotated under different conditions using different frameworks can also all be automatically standardised, facilitating cross-corpora comparison for the first time.

Automatic error types can also be useful in teaching, where a teacher might first correct a student essay and then receive automatic feedback concerning the types and quantity of errors that were made. This information could then be used to track individual student progress or even influence future lesson plans. On a larger scale, researchers will also be able to more systematically analyse the errors made by different L1 learners and adapt their systems or lesson plans as appropriate (cf. Shatz, 2017).

Finally, another advantage of automatic error type annotation is that error types can be used as parameters in GEC systems. For example, words that are identified as morphology errors can be restricted such that a correction system only proposes morphological variants as corrections and not other possibilities such as synonym changes or deletions. Since the majority of recent work in GEC has focused on developing *supervised* machine learning systems however, I instead investigate the contribution of error types in the context of *unsupervised* correction systems.[1] Since it is unlikely that annotated data will ever contain all possible corrections for all error types, it is still important to develop methods that make use of more abundant quantities of unannotated data.

1.2 Aims

Having introduced the potential benefits of automatic annotation, the main aims of this thesis are as follows:

[1] Error types can likely be used in both supervised and unsupervised systems.

1. Develop a method capable of automatically extracting edits from parallel data and annotating them with error types. This involves:

   (a) Preprocessing and standardising the most commonly used GEC corpora. In particular, character edits in untokenized essays must be converted to token edits in tokenized sentences.

   (b) Developing a novel alignment algorithm to align parallel sentences in as linguistically intuitive a way as possible. The quality of the alignment will directly affect the quality of the automatically extracted edits.

   (c) Designing a new rule-based error type framework based solely on automatically obtainable properties of the data; e.g. part-of-speech tags and parse information. This framework will be used to classify the automatically extracted edits.

2. Apply the above method to the output produced by the teams in the CoNLL-2014 shared tasks to carry out a detailed evaluation of system performance for the first time. This will help identify the strengths and weaknesses of particular systems which may be informative for future research.

3. Build a baseline language model based correction system without using annotated training data. The emphasis on supervised machine learning systems in current GEC research means unsupervised approaches have been largely neglected.

4. Investigate how to incorporate error type information into the baseline language model system to improve performance. This will demonstrate that error types are not only useful for annotation and evaluation, but can also be exploited in a correction model.

1.3 Structure

After this introduction, Chapter 2 first provides an overview of GEC as a field. Specifically, it introduces the corpora most commonly used to train and test GEC systems, and also outlines the evolution of different approaches and their strengths and weaknesses. It next describes the three most popular ways to evaluate correction systems, before finally summarising the results of the recent shared tasks that popularised GEC. It concludes with a review of state-of-the-art developments since the end of the shared tasks.

Chapter 3, Chapter 4 and Chapter 5 are the core chapters of this thesis that mainly describe a method to automatically annotate parallel data with error types. Specifically, Chapter 3 opens with a discussion of the different formats of various corpora and introduces a method to standardise them, Chapter 4 describes a linguistically enhanced alignment algorithm that automatically extracts edits from parallel sentences in as human-like a way as possible, and Chapter 5 finally outlines how the extracted edits can be classified according to a new dataset-agnostic rule-based

classifier. The final system, called the ERRor ANnotation Toolkit (ERRANT), is then applied to the system output produced by the teams in the most recent CoNLL-2014 shared task; Chapter 6 presents a detailed evaluation of system error type performance for the first time in GEC.

Having demonstrated one application of ERRANT, Chapter 7 next introduces a baseline unsupervised language model based approach to GEC and describes how it can be improved using ERRANT error types. Chapter 8 finally concludes the thesis and outlines future work.

CHAPTER 2

BACKGROUND

This chapter provides a general overview of GEC as a field and introduces the previous work that is most pertinent to this thesis. Specifically:

- Section 2.1 describes the most commonly used datasets and error type frameworks.
- Section 2.2 summarises the main approaches to GEC.
- Section 2.3 shows how GEC systems can be evaluated.
- Section 2.4 outlines the GEC shared tasks and more recent work.

2.1 Datasets

Although annotated learner corpora are not a prerequisite for GEC research, they nevertheless provide some valuable insights into non-native errors. In particular, there is no better way to calculate robust error type statistics than to analyse genuine errors made by real learners. Alternative ways of computing error type statistics, such as from native or artificially generated data, are generally not as informative (Foster and Andersen, 2009; Felice and Yuan, 2014; Felice, 2016), although recent work has made some progress in this area (Rei et al., 2017; Ge et al., 2018b).

In this section, I will hence introduce several of the most commonly used datasets in GEC and describe their different characteristics. For example, some corpora are annotated with detailed error type information and other metadata, while others consist only of parallel original and corrected sentence pairs. The differences between datasets are also one of the main motivations for developing an automatic error type classifier capable of standardising disparate datasets in this thesis.

Exam Statistics	
Scripts	131,777
Answers	206,476
Exams	23
L1s	130

Test Year	# Scripts
1990-1995	9,016
1996-2000	13,491
2001-2005	41,965
2006-2010	63,454
2011+	3,851

Learner Statistics	
Gender	# Scripts
Male	63,660
Female	65,599

Age	# Scripts
0-10	1,367
11-15	53,803
16-20	28,727
21-30	33,682
31-50	12,394
51+	761

Table 2.1: Various statistics about the data in the CLC. A script may contain more than one answer depending on the exam. Each exam may have multiple sittings in each test year. Learner metadata is occasionally missing from some scripts.

2.1.1 The Cambridge Learner Corpus

The Cambridge Learner Corpus (CLC) is a proprietary collection of over 130 thousand English language exam scripts collected by Cambridge Assessment English[1] in collaboration with Cambridge University Press[2] (Nicholls, 2003). It contains over 200 thousand student answers to questions from 23 different Cambridge exams (some exams require students to answer more than one question) and comprises roughly 29 million words. The majority of answers were written by learners aged 11-30 between 2000 and 2010, but in total, learners from 130 different first language backgrounds (L1s) are represented in the corpus. The essay scores and grades awarded to each learner are also provided with each answer. See Table 2.1 for more details about the composition of the corpus.

In addition to metadata, each answer in each script has also been annotated with corrections and error types in the following format:

```
This <e type="AGV"><i>are</i><c>is</c></e> an annotated sentence.
```

In this XML structure, the error annotation <e> contains both the original incorrect text <i> and the annotator's proposed correction <c>. In this example, the error is also typed AGV which indicates a verb agreement error.[3] Since one of the main aims of this thesis is to develop a new automatic error type annotation framework, it is worth introducing the CLC framework in more detail. Table 2.2 hence shows all the error codes in the full CLC along with their meanings and frequencies.

[1] http://www.cambridgeenglish.org/

[2] http://www.cambridge.org/gb/cambridgeenglish

[3] See Section 3.1 for more information about the CLC data format

Code	Meaning	Freq.
AG	Agreement	1,645
AGA	Pronoun agreement	11,537
AGD	Determiner agreement	8,813
AGN	Noun agreement	35,605
AGQ	Quantifier agreement	1,427
AGV	Verb agreement	56,747
AS	Argument structure	7,366
CD	Determiner countability	115
CE	Compound error	12,207
CL	Collocation	1,203
CN	Noun countability	9,495
CQ	Quantifier countability	3,688
DA	Pronoun derivation	8,612
DC	Conjunction derivation	974
DD	Determiner derivation	5,888
DI	Determiner inflection	577
DJ	Adjective derivation	37,728
DN	Noun derivation	35,091
DQ	Quantifier derivation	734
DT	Preposition derivation	1,627
DV	Verb derivation	10,508
DY	Adverb derivation	21,613
FA	Pronoun form	1,109
FD	Determiner form	7,060
FJ	Adjective form	3,612
FN	Noun form	61,148
FQ	Quantifier form	776
FV	Verb form	88,343
FY	Adverb form	907
IA	Pronoun inflection	454
ID	Idiom	10,564
IJ	Adjective inflection	5,502
IN	Noun inflection	10,566
IQ	Quantifier inflection	1,091
IV	Verb inflection	21,106
IY	Adverb inflection	174
L	Register	18,874
M	Missing	28,290
MA	Missing pronoun	56,081
MC	Missing conjunction	26,522
MD	Missing determiner	176,782
MJ	Missing adjective	4,702
MN	Missing noun	20,286
MP	Missing punctuation	235,877
MQ	Missing quantifier	8,656
MT	Missing preposition	81,749
MV	Missing verb	47,377
MY	Missing adverb	14,392
QL	Question prompt error	368
R	Replacement	78,546
RA	Replacement pronoun	33,753
RC	Replacement conjunction	12,785
RD	Replacement determiner	38,716
RJ	Replacement adjective	49,657
RN	Replacement noun	112,763
RP	Replacement punctuation	285,323
RQ	Replacement quantifier	8,966
RT	Replacement preposition	195,008
RV	Replacement verb	172,070
RY	Replacement adverb	46,115
S	Spelling (non-word)	264,538
SA	American spelling	15,241
SX	Spelling (real word)	48,307
TV	Verb tense	183,310
U	Unnecessary	13,450
UA	Unnecessary pronoun	21,739
UC	Unnecessary conjunction	11,888
UD	Unnecessary determiner	82,371
UJ	Unnecessary adjective	3,682
UN	Unnecessary noun	11,130
UP	Unnecessary punctuation	115,214
UQ	Unnecessary quantifier	2,876
UT	Unnecessary preposition	56,246
UV	Unnecessary verb	24,002
UY	Unnecessary adverb	16,998
W	Word order	76,586
X	Negation	4,180
	Total	**3,191,028**

Table 2.2: The error codes, meanings and frequencies of all 77 error types in the CLC.

In general, the CLC error typology is fairly modular. 10 part-of-speech (POS) based codes for pronouns (A), conjunctions (C), determiners (D), adjectives (J), nouns (N), punctuation (P), quantifiers (Q), prepositions (T), verbs (V) and adverbs (Y) can all be prefixed by 3 edit operation codes for missing (M), replacement (R) and unnecessary (U) or 5 morphological codes for agreement (AG), countability (C), derivation (D), form (F) and inflection (I) where applicable. If none of these codes are appropriate however, a further 12 separate codes are available for argument structure (AS), compound errors (CE), collocations (CL), idioms (ID), register (L), question prompt errors (QL), spelling (S, SA and SX), verb tense (TV), word order (W) and negation (X) errors.

The modularity of this system is one of the main strengths of the CLC as it is very easy to extract examples of specific error types based only on the error code. For example, all preposition errors can be extracted by searching for codes that end with T. It is worth noting that this system is not perfect however, and there are also some inconsistencies in the labelling. For example, we might expect ID to denote a determiner inflection error (based on the prefix + POS pattern), but it instead denotes an idiom error, while DI denotes a determiner inflection error. Similarly, codes C and D have both been used twice to denote conjunction and countability errors, and determiner and derivation errors respectively.

The main weakness of the CLC framework is its size. Although more error types tend to mean greater expressiveness, the trade-off is that this also increases the complexity of the system and results in rarer categories. For example, 50 of the 77 categories each account for less than 1% of the total corpus. This not only means human annotators must learn the precise definitions of increasingly subtle error type distinctions before they can start annotating, but also that machine learning algorithms may struggle to differentiate between larger numbers of rarer categories that become under-represented in training data. A better framework would hence be one that compromises between expressiveness and complexity.

2.1.2 Public First Certificate in English

Although the main CLC is private, a small subsection of it is publicly available online for non-commercial purposes. Specifically, Yannakoudakis et al. (2011) released 1,244 scripts containing 2,488 answers to First Certificate in English (FCE) exams from 2000 and 2001.[4] This subsection comprises roughly 530 thousand words and was similarly annotated with metadata, corrections and error type information. The only difference between the format of the public FCE and the full CLC is that the former a) does not include the gender of the learner, b) explicitly splits the text into paragraphs on a `<p>` tag, and c) uses `<NS>` rather than `<e>` to denote an error. In all other ways, annotations in the public FCE are identical to the CLC.

[4] `https://ilexir.co.uk/datasets/index.html`

Code	Meaning	Freq.
ArtOrDet	Article or determiner	6,637
Cit	Citation	547
Mec	Mechanical (orthography)	3,028
Nn	Noun number	3,764
Npos	Noun possessive	239
Others	Others	1,458
Pform	Pronoun form	185
Pref	Pronoun reference	925
Prep	Preposition	2,413
Rloc-	Local redundancy	4,680
SVA	Subject-verb agreement	1,523
Sfrag	Sentence fragment	186
Smod	Dangling modifier	47
Spar	Sentence parallelism	518
Srun	Run-on sentence	866

Code	Meaning	Freq.
Ssub	Subordinate clause	356
Trans	Transitions	1,352
Um	Unclear meaning	1,141
V0	Missing verb	413
Vform	Verb form	1,443
Vm	Modal verb	431
Vt	Verb tense	3,199
WOadv	Adj/Adv word order	347
WOinc	Word order	696
Wa	Acronyms	48
Wci	Word choice or idiom	5,300
Wform	Word form	2,159
Wtone	Tone or register	581
	Total	**44,482**

Table 2.3: The error codes, meanings and frequencies of all 28 error types in NUCLE v3.2.

2.1.3 NUCLE

The first version of the National University of Singapore Corpus of Learner English (NUCLE) was a public collection of 1,414 student essays written by non-native (mainly South-East Asian) undergraduates at the National University of Singapore (Dahlmeier et al., 2013). It was collected in collaboration with the Center for English Language Communication at the same university and comprised roughly 1.2 million words. Each essay was also annotated with corrections and error types, but unlike the CLC, did not include metadata about the student or exam. The latest version of NUCLE (v3.2)[5] differs from the original in that it is not only slightly smaller (1,397 essays), but also contains 28 rather than 27 error types.[6] These error types, their meanings and frequencies are all shown in Table 2.3.

Compared to the CLC, one advantage of the NUCLE framework is that it is a lot smaller; 28 error types are a lot more manageable than 77. In fact NUCLE error types are also a lot more productive, and only 9 out of the 28 categories each account for less than 1% of the corpus. That said, a significant drawback to the NUCLE framework is that it is a lot less modular than the CLC and error categories are sometimes inconsistent in scope.

For example, while it is straightforward to extract all missing word errors from the CLC by searching for error codes prefixed by M, the same is not possible in NUCLE. Instead, there is only one category that exclusively contains missing words (missing verbs: V0) and all other missing words are variously subsumed under other categories, such as prepositions (Prep) or others (Others). Similarly, although the local redundancy category (Rloc-) is roughly analogous

[5] `http://www.comp.nus.edu.sg/~nlp/corpora.html`

[6] Preposition errors were originally classified under word choice errors (Wcip) but were separated out and given their own category.

to a general unnecessary word category, certain unnecessary words are again subsumed under other categories such as article or determiner (ArtOrDet) or preposition (Prep). This ultimately means it is very difficult to extract errors with similar properties from NUCLE and different error types are unpredictably more expressive than others.

In fact some of the error types themselves are also fairly questionable. For example, citation errors (Cit) only really apply to formal university essays, and so are not technically grammatical errors, while dangling modifiers (Smod) and acronym errors (Wa) are extremely specific categories given their infrequency. A better framework would hence be one that only contains categories that are both frequent and generalisable; this is one of the core principles behind the new framework I developed.

2.1.4 CoNLL-2013 and CoNLL-2014

Since the National University of Singapore also hosted the Conference on Natural Language Learning (CoNLL) GEC shared tasks of 2013 and 2014 (Ng et al., 2013, 2014), it was also responsible for creating new test data (see Section 2.4.2). The CoNLL-2013 and CoNLL-2014 test sets were thus annotated in roughly the same conditions as NUCLE and respectively consist of 50 annotated essays of about 600 words each.

The CoNLL-2014 test set is particularly noteworthy because it was also the first dataset to be doubly annotated by two different annotators, and in fact Bryant and Ng (2015) and Sakaguchi et al. (2016) both subsequently annotated it a further 8 times each for a total of 18 sets of overlapping annotations. This test set is currently one of the most popular benchmarks for GEC systems.

2.1.5 Lang-8

Lang-8 is an online language learning social networking service that enables non-native learners to post text in the language they are trying to learn in the hope that native users will post replies with corrections.[7] The website was crawled in December 2010 by Mizumoto et al. (2012), who subsequently made two different versions of the data available online as research corpora.[8]

Specifically, the Lang-8 Learner Corpus v2.0 is the full multilingual version of the corpus that has not been preprocessed in any way, while the Lang-8 Corpus of Learner English v1.0 (Tajiri et al., 2012) is an English-only subset that has been cleaned using various preprocessing techniques (Mizumoto et al., 2011). The full corpus contains approximately 8.2 million sentences in many different L2 languages, while the English-only corpus contains roughly 1 million sentences of L2 English alone. Neither corpus has been explicitly annotated with error type annotations and instead both consist of only parallel original and corrected sentence pairs. It

[7] `http://lang-8.com/`
[8] `http://cl.naist.jp/nldata/lang-8/`

is worth noting that in addition to the official corpora, different research groups have also independently scraped the Lang-8 website and created their own non-public Lang-8 corpora (Junczys-Dowmunt and Grundkiewicz, 2016; Ge et al., 2018a).

Finally, it is also worth mentioning that preprocessing is especially important in a corpus like Lang-8 because it has not been annotated to the same standards as NUCLE or the CLC. Instead, Lang-8 users were free to post anything in response to learners' queries, and so "corrections" often include in-line comments, suggestions, emoticons and other artefacts that are not strictly part of the sentence. Consequently, the main advantage of Lang-8 is that it provides a large quantity of lower quality parallel data that can bolster the small amount of higher quality annotated data used in certain GEC paradigms.

2.1.6 JFLEG

The GUG corpus is a small collection of 3,129 learner sentences annotated for grammaticality on an ordinal scale by Heilman et al. (2014). Napoles et al. (2017) subsequently reannotated 1,501 of these sentences[9] with four sets of overlapping corrections to create the Johns Hopkins University Fluency-Extended GUG corpus (JFLEG). The corpus was then split roughly in half to create a development and test set of 754 and 747 sentences respectively, both of which are currently used as popular benchmarks. Like Lang-8 however, the corpus only consists of parallel sentence pairs and does not contain any explicit annotations or metadata.

The main difference between JFLEG and other GEC corpora is that JFLEG advocates fluency rather than minimally corrected sentences. Specifically, Sakaguchi et al. (2016) argued that forcing annotators to make minimal edits within the confines of an error type framework often led to unidiomatic sentences, and that it was instead more natural to allow annotators to completely rewrite sentences if necessary. JFLEG was thus an attempt to encourage researchers to build systems that were capable of producing idiomatic rather than just grammatical output.

The fluency vs. minimal edit debate still continues today, but it is likely that the level of feedback a GEC system provides ultimately depends on the target user. For example, although native or advanced learners might benefit from fluency corrections, beginner or intermediate learners may be disheartened if a fluency system continually suggests they rewrite everything using more complicated grammatical constructions.

2.2 Approaches to GEC

As GEC has evolved, so too have the systems and approaches designed to tackle it. In this section I will hence introduce the most popular paradigms and describe their various strengths and weaknesses.

[9]The paper says 1,511 sentences but the data itself is 1,501

2.2.1 Rules

The oldest and simplest GEC systems depended entirely on hand-coded rules. For example, sentences in English typically begin with a capital letter and end with a full stop, so it is straightforward to write a rule that enforces these constraints. Rules based on string matching alone are rather limited however, and so most rule-based systems additionally make use of automatically obtained part-of-speech (POS) tags and parse information to detect and correct errors (McCoy et al., 1996; Park et al., 1997; Schneider and McCoy, 1998; Michaud et al., 2000; Dodigovic, 2007).

To show how this information can be useful, consider the case of subject-verb agreement (SVA), where the grammatical number of a subject must agree with the grammatical number of the verb it is governed by.[10]

- The cat often chases the mouse.
- *The cats often chases the mouse.
- *The cat often chase the mouse.
- The cats often chase the mouse.

To detect the erroneous combinations, we hence need to determine 1) the subject of the verb, 2) the number of the subject, and 3) the number of the verb. Although this information is not explicit in raw text, it can be obtained from a parser and a POS tagger. Specifically, a parser annotates the grammatical relations between words at arbitrary levels of distance (here, the subject and verb are separated by 'often'), while a POS tagger assigns part-of-speech tags that often encode grammatical number (Figure 2.1). Examples of open-source software that can do this include: RASP[11] (Briscoe et al., 2006), NLTK[12] (Bird et al., 2009), CoreNLP[13] (Manning et al., 2014), and spaCy[14] (Honnibal and Johnson, 2015). Note that each of these toolkits might also assign labels at different levels of granularity; e.g. there are 36 main POS tags in the English Penn Treebank tagset (Marcus et al., 1993), but 12 in the Google Universal Tagset (Petrov et al., 2012).

For rule-based GEC, it then becomes a case of i) extracting the subject and verb from the parse and ii) checking whether their POS tags have the same number agreement or not. Rules for other error types can similarly be constructed in an analogous way.

The main advantage of rule-based methods over other approaches is that they are often easy to implement and usually very precise. Since rules are defined manually by humans, humans have direct control over when they activate. This can be a double-edged sword however, and

[10] It is linguistic convention to mark ungrammatical text with an asterisk.
[11] `https://www.ilexir.co.uk/rasp/index.html`
[12] `https://www.nltk.org/`
[13] `https://stanfordnlp.github.io/CoreNLP/`
[14] `https://spacy.io/`

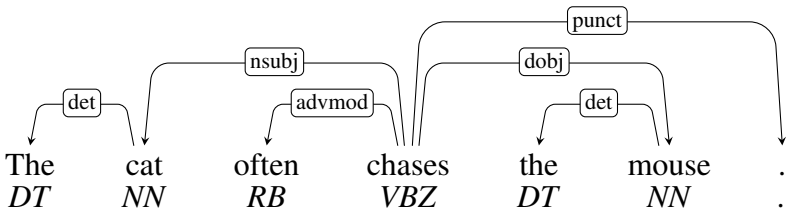

Figure 2.1: The parse reveals "cat" is the nominal subject (nsubj) of "chases" and the POS tags show that both words are singular (*NN* and *VBZ* rather than *NNS* and *VBP*).

some rules may become so complicated that they are laborious to create and maintain. There is hence a trade-off between rule precision and rule complexity.

Another advantage of rule-based methods is that they do not rely on training data and so can be applied to any language. For example, in addition to English, rule-based grammar checkers have also been developed for German (Weischedel et al., 1978), Spanish and Greek (Bustamante and Leon, 1996), Swedish (Arppe, 2000; Birn, 2000; Domeij et al., 2000), Punjabi (Gill and Lehal, 2008), Persian (Ehsan and Faili, 2010), and Mandarin Chinese (Wu et al., 2015), amongst others. The disadvantage of this approach, however, is that new rules had to be written for each new language, and so rules are typically not very generalisable.

2.2.2 N-gram language models

The intuition behind using language models (LMs) in GEC is that lower probability sequences are more likely to be ungrammatical than higher probability sequences. For example, it is intuitive that 'I eated the cake' is less likely than 'I ate the cake' because the former contains an error. In order to quantify this difference however, we need a language model.

The most common way to build a language model is to compute the probabilities of short windows of text (**n-grams**) from a large corpus of monolingual text, such as Wikipedia[15] (Pasternack and Roth, 2008) or Common Crawl[16] (Buck et al., 2014). For example, the 4-gram probability of 'I ate the cake' can be computed as the number of times 'I ate the cake' appeared in a corpus divided by the number of times 'I ate the' appeared in the same corpus. This is also the probability of the word 'cake' given 'I ate the' (Equation 2.1).

$$\begin{aligned} P(I\,ate\,the\,cake) &= P(cake \mid I\,ate\,the) \\ &= \frac{Count(I\,ate\,the\,cake)}{Count(I\,ate\,the)} \end{aligned} \tag{2.1}$$

If we instead compute the probability of this sequence using bigrams ($n = 2$) however, we must apply the **chain rule of probability** to obtain the overall probability of the sequence (Equation 2.2).

[15] `https://dumps.wikimedia.org/`
[16] `http://commoncrawl.org/`

$$
\begin{aligned}
P(I\,ate\,the\,cake) &= P(ate\,|\,I) \times P(the\,|\,ate) \times P(cake\,|\,the) \\
&= \frac{C(I\,ate)}{C(I)} \times \frac{C(ate\,the)}{C(ate)} \times \frac{C(the\,cake)}{C(the)}
\end{aligned} \tag{2.2}
$$

In the general case, the probability of any window of N words (from word w_{n-N+1} to word w_n) can be expressed as Equation 2.3 and the chain rule can be applied to calculate probabilities of sequences longer than N. We refer the reader to Jurafsky and Martin (2009, Ch. 4) and Chelba et al. (2014) for a more comprehensive introduction to language modelling.

$$
P(w_n|w_{n-N+1}^{n-1}) = \frac{C(w_{n-N+1}^{n-1} w_n)}{C(w_{n-N+1}^{n-1})} \tag{2.3}
$$

In relation to GEC, the most common application of a LM is to detect improbable sequences of words and score alternative suggestions (Turner and Charniak, 2007; Gamon et al., 2008; Bergsma et al., 2009; Heilman et al., 2012; Lee et al., 2014; Yeh et al., 2017; Zhao et al., 2017; Bryant and Briscoe, 2018; Lin and Chen, 2018). Specifically, having detected a low probability sequence, a correction system might propose an alternative that maximally increases the overall probability above some threshold. A significant advantage of this approach is that it is theoretically able to handle every error type provided a suitable correction can be generated.

One disadvantage, however, is that probability is not always a good proxy for grammaticality. For example, consider the following joke:

> Student: I is ...
> Teacher: No, you must always say 'I am'.
> Student: Ok. I am the ninth letter of the alphabet.

This example shows one of the very few cases where it is grammatical to say 'I is', so researchers should be aware that LMs may fail when handling rare words or constructions. Despite this weakness, LMs are prolific in GEC because they i) are fairly simple to implement, ii) only require native training data and iii) are highly versatile. It is not uncommon therefore to find LMs used in combination with other approaches.

2.2.3 Statistical classifiers

In machine learning, a classifier is an algorithm that attempts to classify an input $x \in \mathbb{X}$ into a category $y \in \mathbb{Y}$. Since the size of all possible corrections $\mathbb{Y}$ is extremely large in error correction however, most GEC classifiers instead focus only on specific error types where $\mathbb{Y}$ is a lot smaller. In particular, there has been a lot of work on article, preposition, and verb errors:

- Article classifiers: Han et al. (2006); De Felice (2008); De Felice and Pulman (2008); Gamon et al. (2008); Rozovskaya and Roth (2010b); Dahlmeier and Ng (2011); Seo et al. (2012); Rozovskaya and Roth (2013); Xiang et al. (2013); Rozovskaya and Roth (2014).

- Preposition classifiers: Chodorow et al. (2007); De Felice and Pulman (2007); De Felice (2008); De Felice and Pulman (2008); Gamon et al. (2008); Tetreault and Chodorow (2008); Tetreault et al. (2010); Rozovskaya and Roth (2010a); Dahlmeier and Ng (2011); Cahill et al. (2013); Rozovskaya and Roth (2013, 2014).

- Verb classifiers: Gamon et al. (2008); Lee and Seneff (2008); Tajiri et al. (2012); Rozovskaya and Roth (2013, 2014); Rozovskaya et al. (2014b).

More specifically, article classifiers attempt to predict one of $\{\varepsilon, a, the\}$ in front of noun phrases (ε is a null article), preposition classifiers attempt to predict one of the top n most frequent prepositions (n ranges between 5 and 36 depending on the study), and verb classifiers attempt to predict the form or tense of a verb (e.g. $\{eat, eats, ate, eating, eaten\}$). The input to all these classifiers is hence the word or context in which each of these error types occurs.

Unlike a deterministic rule-based system which requires explicit instructions concerning how the properties of input x map to output y however, a classifier rather learns a function f that maps x to y automatically. In order to do this however, the properties of x must first be converted into a machine readable format in a process known as **feature extraction**. Specifically, all the properties of x that humans think will help the classifier, such as the current word, its POS tag, syntactic position, etc., must be encoded as a vector.

It is then up to the classifier to determine how useful each encoded feature is and hence weight its importance. This process is called **training** and such weights are learnt from a large number of examples x_{train} for which we know the correct answer y_{train}. The details of training vary depending upon the classification algorithm, but popular examples include naive Bayes, logistic regression, decision trees (Quinlan, 1986), support-vector machines (Cortes and Vapnik, 1995), maximum entropy models (Berger et al., 1996; Ratnaparkhi, 1996), the averaged perceptron (Freund and Schapire, 1999), and conditional random fields (Lafferty et al., 2001).

These algorithms also sometimes have additional configurational parameters, called **hyper-parameters**, which control variables such as model complexity and maximum training time. These parameters are external to the training process and must be optimised separately on a small set of held-out data called the **development set**. The predictions $\hat{y}_{dev}$ are thus compared against the correct answers y_{dev} using some metric (Section 2.3), and the classifier iteratively updates until the parameters converge on an optimum solution. The system is finally evaluated on another set of held-out data, the **test set**, which is entirely independent from both the training and development set, and this score is treated as the true performance of the classifier.

One advantage of classifiers over rules is that humans do not need to explicitly define the relationship between features and output, and can instead rely on machine learning to determine

the importance of each feature. This somewhat overcomes the problem of having to maintain a large number of rules with increasing complexity. That said, humans still need to decide which features to include in their system (**feature engineering**) which may not be trivial. This is especially significant given that some, or too many, features may actually harm classifier performance.

Finally, as mentioned at the start of this section, another limitation of classifiers is that they only target very specific error types with small confusion sets. They hence do not extend well to other error types, such as synonym errors, or a more general solution. In fact if we were to build multiple classifiers for multiple error types, a related issue is that classifier order matters, and results will vary depending on whether you apply, e.g., the article classifier before the preposition classifier or vice versa. Given the 28 error types in NUCLE, let alone the 77 in the CLC, classifiers quickly become impractical.

2.2.4 Statistical machine translation

The traditional goal of statistical machine translation (SMT) is to automatically translate one language to another; e.g. English to French. When applied to GEC however, the goal is to translate "bad" English to "good" English. Although these two tasks may seem somewhat different, they can both be viewed in terms of the **noisy channel model** (Shannon, 1948). Specifically, we can imagine a correct signal C as having passed through some noisy channel to produce an erroneous signal E, and the goal is to reconstruct the correct signal given the erroneous signal (Figure 2.2).

This can be formulated mathematically using Bayes' rule, where $\hat{C}$ is the most likely prediction of what the original signal C was given the noisy signal E (Equation 2.4):

$$\hat{C} = \arg\max_C P(C|E) = \arg\max_C \frac{P(E|C)P(C)}{P(E)} = \arg\max_C P(E|C)P(C) \tag{2.4}$$

The core components of a SMT system are thus a language model that computes $P(C)$, and a translation model that computes $P(E|C)$. Since LMs were already introduced in Section 2.2.2, I will only describe the translation model here.

Fundamentally, the translation model is a model that predicts the probability that a word or phrase maps to another word or phrase. For example, the word *pamplemousse* in French is more likely to map to *grapefruit* than *banana* in English because that is what it means, while the phrase *discuss about* in "bad English" is more likely to map to *talk about* than *walk about* in "good English" because that is a more likely correction. Since these mappings are rarely deterministic however, for example *discuss about* might also map to *discuss*, they are instead learnt automatically from parallel data. Exactly how these mappings are learnt is beyond the scope of this thesis, but the summary is that the Expectation-Maximisation algorithm (Dempster et al., 1977) finds the most probable alignment between words and phrases in parallel sentences

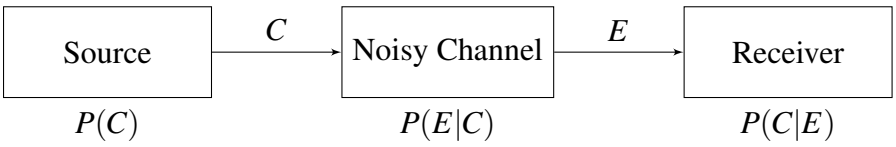

Figure 2.2: The noisy channel model (Shannon, 1948).

based on IBM models 1-5 (Brown et al., 1993) and a more sophisticated phrasal alignment (Koehn et al., 2003). See Koehn (2010) for more information on this and SMT in general.

In relation to GEC, the earliest work using SMT was highly varied, with one system correcting only a specific error type (Brockett et al., 2006) and another correcting errors via round-trip translation (Madnani et al., 2012). In fact Dahlmeier and Ng (2012a) even advocated against using SMT in GEC after obtaining poor results, and so it was only much later when larger quantities of training data became available that SMT became a more dominant approach (Mizumoto et al., 2012; Chang et al., 2013; Wu et al., 2013; Susanto et al., 2014; Junczys-Dowmunt and Grundkiewicz, 2016; Rozovskaya and Roth, 2016; Chollampatt et al., 2016a; Chollampatt and Ng, 2017; Napoles and Callison-Burch, 2017; Grundkiewicz and Junczys-Dowmunt, 2018). It is worth mentioning, however, that this reliance on parallel training data is one of the main weaknesses of SMT in GEC, as annotated correction corpora are typically a lot smaller and harder to obtain than multilingual translation corpora.

Nevertheless, the main advantage of SMT over other approaches is that it can theoretically correct all error types without expert knowledge or feature engineering. Furthermore, it can also handle interacting errors and even produce an **n-best list** of alternative corrections for a single sentence. This has led to much work on n-best list re-ranking, which aims to determine whether the best correction for a sentence is not the single most likely candidate (i.e. n=1), but is rather somewhere further down in the top n most likely candidates (Hoang et al., 2016; Mizumoto and Matsumoto, 2016; Yuan et al., 2016; Yannakoudakis et al., 2017).

Finally, although SMT generalises well to all error types, one of its biggest weaknesses is that it is not very customisable. In particular, while classifiers can be improved by simply adding new features, it is a lot harder to add constraints that target specific error types in SMT systems.

2.2.5 Neural machine translation

Neural networks are a general class of model that learn how to map an input x to an output y via a number of hidden states h. In GEC, the most common type of neural model uses the **encoder-decoder** framework (Xie et al., 2016; Yuan and Briscoe, 2016; Ji et al., 2017; Sakaguchi et al., 2017b; Chollampatt and Ng, 2018a; Ge et al., 2018a; Grundkiewicz and Junczys-Dowmunt, 2018; Junczys-Dowmunt et al., 2018), but other types of model have also been used (Chollampatt et al., 2016a,b; Chollampatt and Ng, 2017; Yannakoudakis et al., 2017). The encoder-decoder framework was originally developed for neural machine translation (NMT) (Cho et al., 2014;

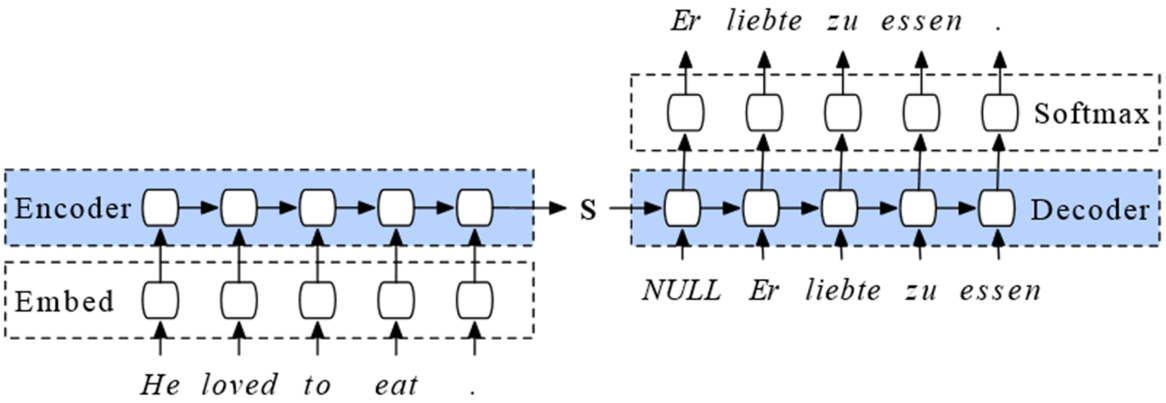

Figure 2.3: An overview of the encoder-decoder framework used in neural machine translation. An input sequence is encoded as a vector s which a decoder then translates based on this vector and all previously predicted words. The same principle applies to GEC.

Sutskever et al., 2014), but like SMT, it can also be adapted for GEC.

In NMT, an **encoder** first encodes a sequence of words $x = (x_1, x_2, ..., x_T)$ as a vector $s = q(h_1, h_2, ..., h_T)$, where q is a non-linear function, and h_t is an activation function $f(x_t, h_{t-1})$ at time t. The activation function f is traditionally just a sigmoid function, but other more sophisticated functions can also be used; e.g. the Long-Short Term Memory (Hochreiter and Schmidhuber, 1997) and Gated Recurrent Units (Cho et al., 2014). Having encoded an input sentence, the **decoder** next tries to predict the output sequence $y = (y_1, y_2, ..., y_T)$ based on both the encoded input sequence and all previously predicted words $\{y_1, y_2, ..., y_{t-1}\}$ (Equation 2.5). An overview of this process is shown in Figure 2.3.

$$p(y) = \prod_{t=1}^{T'} p(y_t | y_1, y_2, ..., y_{t-1}, s) \tag{2.5}$$

Like SMT, one of the main advantages of neural approaches to GEC is that NMT models can theoretically handle all error types. They also contain many more parameters than regular SMT models, and so can typically make more nuanced decisions. It is primarily for this reason that the encoder-decoder approach is currently the state of the art in GEC, but it is worth mentioning that this has only been made possible because of greater access to larger corpora. Without sufficient parallel data, SMT systems typically outperform NMT systems (Junczys-Dowmunt et al., 2018).

One of the main disadvantages of the NMT approach, however, is that models are not particularly transparent and do not generalise well to unseen data. In particular, neural models abstract away from linguistically motivated features, and instead learn their own feature representations which are not interpretable by humans. This makes it very difficult to diagnose why a model made a mistake when it encounters something it has not seen before. Nevertheless, NMT is one of the most popular paradigms in current GEC research, with the highest reported results using a combination of both NMT and SMT (Grundkiewicz and Junczys-Dowmunt, 2018).

2.3 Evaluation metrics

Regardless of the approach, evaluation is an important part of GEC system development. Specifically, without a reliable means of evaluation, it is impossible to quantify system performance.

Although the most intuitive way to do this might be to manually inspect system output and keep track of how many times a system was right and wrong, this approach quickly becomes impractical for large datasets. Specifically, manual inspection takes a lot of time and effort, and must also be carried out on a large enough sample of data to overcome statistical bias. Instead, it is much more practical to rely on automatic metrics that can estimate performance a lot more quickly and cheaply. In this section, I will hence introduce the three most popular metrics in GEC, although note that other metrics are also available (Bryant and Ng, 2015; Grundkiewicz et al., 2015; Napoles et al., 2016c; Asano et al., 2017; Choshen and Abend, 2018b).

2.3.1 MaxMatch

The MaxMatch (M2) scorer (Dahlmeier and Ng, 2012b) is the current *de facto* evaluation metric in GEC. It was developed in response to shortcomings in the scorer used in the Helping Our Own shared tasks (Section 2.4.1), and subsequently became the official metric in the Conference on Natural Language Learning shared tasks (Section 2.4.2). It is based on the F-score (van Rijsbergen, 1979).

To explain the F-score, consider the following aligned original, hypothesis and reference sentences:

Original	Let	's	discuss	about	the	**softwares**	.
Hypothesis	Let	's	discuss	ε	the	software	.
Reference	Let	's	**talk**	about	the	software	.

Starting from left to right, the first difference between the hypothesis and the reference is the word 'discuss' (in red). Specifically, the hypothesis kept the word 'discuss' while the reference changed it to 'talk'. This mismatch is called a **false negative** (FN) because the system did not change something it should have. In contrast, the next difference shows a case where the system changed something it should not have. Specifically, the system deleted 'about' but the reference kept it (in purple). This is called a **false positive** (FP). Finally, since both the hypothesis and reference agreed that 'softwares' should be changed to 'software' (in green), this is called a **true positive** (TP).

Having totalled the TP, FP and FN, we can then calculate precision (P) and recall (R), where precision measures the proportion of hypothesis edits that were correct, while recall measures the proportion of reference edits that were missed. They are calculated as follows:

$$P = \frac{TP}{TP+FP} \tag{2.6}$$

$$R = \frac{TP}{TP+FN} \tag{2.7}$$

We can then use P and R to calculate the F_β score (Equation 2.8). The value of β controls the relationship between P and R where $\beta < 1$ weights towards precision while $\beta > 1$ weights towards recall. $\beta = 1$ is the harmonic mean of precision and recall and weights them equally. Values for P, R and F_β all fall within the range 0 and 1.

$$F_\beta = (1+\beta^2) \times \frac{P \times R}{(\beta^2 \times P) + R} \tag{2.8}$$

One shortcoming of measuring performance in terms of edit overlap between the hypothesis and the reference however, is that there is often more than one way to define an edit span. For example, the edit [*has eating* $\rightarrow$ *was eaten*] can also be realised as [*has* $\rightarrow$ *was*] and [*eating* $\rightarrow$ *eaten*]. If the hypothesis combines them, but the reference does not, the edit will not be counted as a TP even though it produces the same valid correction. A similar thing happens when reference annotations include words that do not change; e.g. [*the cat* $\rightarrow$ *the dog*] vs. [*cat* $\rightarrow$ *dog*]. This ultimately results in underestimated system performance.

Dahlmeier and Ng (2012b) hence designed the M2 scorer to overcome this problem by dynamically calculating the different ways of combining edits based on a Levenshtein alignment (Levenshtein, 1966). They also introduced a parameter u, which controls the maximum number of unchanged words allowed in an edit, to handle cases where reference edits contain unchanged words (this parameter was arbitrarily set to 2). This gives the M2 scorer the ability to *maximally match* the intersection between the hypothesis edits $\{e_1,...,e_n\}$ and the gold reference edits $\{g_1,...,g_n\}$, and hence calculate P and R as follows:

$$P = \frac{\sum_{i=1}^{n} \mid e_i \cap g_i \mid}{\sum_{i=1}^{n} \mid e_i \mid} \tag{2.9}$$

$$R = \frac{\sum_{i=1}^{n} \mid e_i \cap g_i \mid}{\sum_{i=1}^{n} \mid g_i \mid} \tag{2.10}$$

These equations are consistent with Equation 2.6 and 2.7 and can also be used to calculate the F-score (Equation 2.8).

2.3.2 I-measure

Although the M2 scorer overcame some of the problems of previous scorers, it also suffered from a number of other limitations; for example:

Tokens			Classification	
Original	**Hypothesis**	**Reference**	**Detection**	**Correction**
a	a	a	TN	TN
a	a	b	FN	FN
a	a	-	FN	FN
a	b	a	FP	FP
a	b	b	TP	TP
a	b	c	TP	FP, FN, FPN
a	b	-	TP	FP, FN, FPN
a	-	a	FP	FP
a	-	b	TP	FP, FN, FPN
a	-	-	TP	TP
-	a	a	TP	TP
-	a	b	TP	FP, FN, FPN
-	a	-	FP	FP
-	-	a	FN	FN

Table 2.4: How to classify a token in a 3-way alignment based on the extended Writer-Annotator-System scheme.

1. The unchanged word parameter set to 2 is arbitrary yet still affects results.

2. A system that has no TPs means P = 0, R = 0 and F = 0, and is hence indiscriminable from a do-nothing baseline.[17]

3. The lack of a true negative (TN) count means we cannot calculate accuracy, which would help in discriminating against systems where F = 0.

This prompted Felice and Briscoe (2015) to develop the *I*-measure, a metric designed to evaluate a system in terms of overall text *Improvement* rather than edit overlap. They thus carried out a 3-way alignment of the original, hypothesis and reference texts and classified every token according to an extended version of the Writer-Annotator-System (WAS) evaluation scheme by Chodorow et al. (2012) (Table 2.4). Since cases where $original \neq hypothesis \neq reference$ are both a FP and FN in correction, they also introduced a False Positive and Negative term (FPN). This scheme enabled them to calculate accuracy, which they further modified by weighting TPs and FPs to more intuitively reward or punish a system (Equation 2.11). This weight w was set to 2 by default.

$$WAcc = \frac{w \cdot TP + TN}{w \cdot (TP + FP) + TN + FN - (w+1) \cdot \frac{FPN}{2}} \tag{2.11}$$

Having computed system weighted accuracy, they then compared this against baseline weighted accuracy to produce an improvement score I (Equation 2.12). This I score falls within the range -1 and 1, where $I < 0$ indicates text degradation and $I > 0$ indicates text improvement.

[17] Note that if TP = FP = 0, there is a division by 0 error and P = 1 by definition.

$$I = \begin{cases} \lfloor WAcc_{sys} \rfloor & \text{if } WAcc_{sys} = WAcc_{base} \\ \dfrac{WAcc_{sys} - WAcc_{base}}{1 - WAcc_{base}} & \text{if } WAcc_{sys} > WAcc_{base} \\ \dfrac{WAcc_{sys}}{WAcc_{base}} - 1 & \text{otherwise} \end{cases} \tag{2.12}$$

Despite overcoming several problems with the M2 scorer, the I-measure has not been widely adopted in GEC. This is because several studies found it negatively correlated with human judgements at the corpus level (Grundkiewicz et al., 2015; Napoles et al., 2015, 2016c; Sakaguchi et al., 2016; Chollampatt and Ng, 2018b), although two of these studies also found it more strongly correlated with human judgements at the sentence level (Napoles et al., 2016c; Chollampatt and Ng, 2018b). This perhaps suggests the default weight parameter must be tuned to more properly reflect human intuition.

2.3.3 GLEU

One limitation of both the M2 scorer and the I-measure is that they both rely on explicit reference annotations. Specifically, a system can only be evaluated if the edits that transform the source sentence to the reference sentence are well-defined. In an effort to overcome this reliance on human edit spans, Napoles et al. (2015) introduced the Generalised Language Evaluation Understanding (GLEU) metric, a variation of the Bilingual Evaluation Understudy (BLEU) metric commonly used in machine translation (Papineni et al., 2002).

Specifically, while BLEU calculates precision p_n by measuring the overlap between hypothesis n-grams $H = \{h_1, ..., h_k\}$ and reference n-grams $R = \{r_1, ..., r_k\}$, GLEU extends the calculation to not only reward hypothesis n-grams that overlap with the reference but not the original ($R \setminus O$), but also penalise hypothesis n-grams that overlap with the original but not the reference ($O \setminus R$). Although the original formulation of GLEU also included a weight λ that required tuning based on the number of references, Napoles et al. (2016b) subsequently released a new version, GLEU$^+$, that removed this weight (Equation 2.13).

$$p_n^* = \frac{\left(\sum_{ngram \in \{H \cap R\}} count_{H,R}(ngram) - \sum_{ngram \in \{H \cap O\}} \max[0, count_{H,O}(ngram) - count_{H,R}(ngram)] \right)}{\sum_{ngram \in \{H\}} count_H(ngram)}$$

$$count_A(ngram) = \#\text{ occurrences of } ngram \text{ in } A$$

$$count_{A,B}(ngram) = \min(\#\text{ occurrences of } ngram \text{ in } A, \#\text{ occurrences of } ngram \text{ in } B) \tag{2.13}$$

$$\text{BP} = \begin{cases} 1 & \text{if } h > r \\ \exp(1 - r/h) & \text{if } h \leq r \end{cases} \tag{2.14}$$

$$\text{GLEU}(O,H,R) = \text{BP} \cdot \exp\left(\frac{1}{N}\sum_{n=1}^{N} \log\, p_n^*\right) \tag{2.15}$$

The final GLEU score for parallel original, hypothesis and reference texts is hence calculated according to Equation 2.15, where the maximum size of the n-gram window N is typically set to 4. Like BLEU, this equation also includes a brevity penalty (BP) which penalises hypotheses that are shorter than the references (Equation 2.14).

In terms of reliability, although several studies found GLEU correlated more strongly with human judgements than the M2 scorer (Napoles et al., 2015; Asano et al., 2017; Choshen and Abend, 2018a), several other studies found lower or mixed correlations (Napoles et al., 2016b; Sakaguchi et al., 2017a; Chollampatt and Ng, 2018b). In fact Choshen and Abend (2018a) also raised several methodological issues concerning how these correlations were calculated, citing large differences between Grundkiewicz et al. (2015) and Napoles et al. (2015) who independently carried out the same correlation experiments using different sets of human judgements.

Ultimately, robust evaluation of GEC systems is still an active area of research.

2.4 Shared tasks and beyond

The series of shared tasks between 2011-2014 arguably spearheaded research into GEC and are largely responsible for the current state of the field today. They not only promoted GEC to a wider audience, but also facilitated the evaluation of many different approaches on a level playing field. This section hence introduces each of the shared tasks and comments on the evolution of approaches to GEC up to the present day. It is also worth mentioning that similar shared tasks have driven the field for Arabic (Mohit et al., 2014; Rozovskaya et al., 2015) and Mandarin Chinese (Yu et al., 2014; Lee et al., 2015, 2016; Rao et al., 2017, 2018).

2.4.1 Helping Our Own

The Helping Our Own (HOO) shared task of 2011 (Dale and Kilgarriff, 2011) was the first to encourage researchers to work together on general purpose error correction. It was motivated by the observation that non-native English speakers tended to have more trouble writing formal academic papers than native speakers, and so HOO-2011 was advertised as an opportunity to build software that could assist non-native researchers with their writing. The organisers hence released roughly 20,000 words of annotated text fragments from 19 academic papers as a development set, and kept a similar amount of data aside as a test set.

HOO-2011

Team	Approach	P	R	F_1	Reference
JU	CL, RB	10.4	3.8	5.5	Bhaskar et al. (2011)
LI	RB	20.9	3.2	5.6	Ivanova et al. (2011)
NU	CL, LM	29.1	7.4	11.8	Dahlmeier et al. (2011)
UD	LM	5.0	2.0	2.8	Zesch (2011)
UI	CL, RB	**50.7**	**13.3**	**21.1**	Rozovskaya et al. (2011)
UT	LM	5.0	4.1	4.5	Boyd and Meurers (2011)

HOO-2012

Team	Approach	P	R	F_1	Reference
CU	CL	**70.00**	4.64	8.70	Kochmar et al. (2012)
ET	CL, LM, RB	-	-	-	Heilman et al. (2012)
JU	CL, RB	2.52	2.65	2.58	Bhaskar et al. (2012)
KU	CL	1.45	15.45	2.65	Lee et al. (2012)
LE	CL, LM, RB	31.15	22.08	25.84	Quan et al. (2012)
NA	CL, LM	29.43	20.53	24.19	Sakaguchi et al. (2012a)
NU	CL, LM	45.45	20.97	**28.70**	Dahlmeier et al. (2012)
TC	CL, LM	2.66	12.80	4.41	Lynch et al. (2012)
TH	CL	9.44	25.61	13.79	Wu et al. (2012)
UD	CL, LM	1.20	4.19	1.87	Zesch and Haase (2012)
UI	CL, RB	26.39	**28.26**	27.29	Rozovskaya et al. (2012)
UT	LM	21.95	15.89	18.44	Boyd et al. (2012)
VA	CL	6.16	7.51	6.77	van den Bosch and Berck (2012)
VT	LM, RB	8.76	4.19	5.67	Daudaravicius (2012)

Table 2.5: Table showing the teams, approaches, and their scores for correction in the HOO-2011 and HOO-2012 shared tasks. CL = classifier, LM = language model, RB = rule-based. ET in HOO-2012 only submitted output for detection and not correction.

Systems were evaluated in terms of *detection*, *recognition*, and *correction*, or rather the extent to which they a) correctly identified a word that needed changing, b) correctly identified the exact span of a word or words that needed changing, and c) correctly identified the exact span of a word or words that needed changing and also edited the text to match the reference. All scores were reported in terms of F_1 (Section 2.3.1).

The task ultimately proved extremely challenging however, and of the 6 teams that took part, the best correction system scored just 21.1 F_1 using a combination of classifiers and rules (Rozovskaya et al., 2011). In contrast, the team that came second scored 11.8 F_1 using classifiers, a language model and spell checking software (Dahlmeier et al., 2011), while all other teams scored less than 6 F_1, similarly using combinations of classifiers, language models and rules (Bhaskar et al., 2011; Ivanova et al., 2011; Zesch, 2011; Boyd and Meurers, 2011).

In light of the difficulty of the full correction task, the organisers of HOO-2012 (Dale et al., 2012) subsequently decided to simplify the second iteration of the task and instead asked participants to only correct determiner and preposition errors. This was because these error types

had previously been extensively studied (cf. Section 2.2.3) and so it was hoped that more teams would participate. Since all the existing systems for these error types were usually evaluated on different datasets however, HOO-2012 also presented a good opportunity to more formally evaluate existing approaches under controlled conditions. To that end, HOO-2012 moved away from the academic domain and the academic papers that were annotated for HOO-2011, and instead used the public FCE (Section 2.1.2) as official training, development and test data. This dataset was considerably larger than that in HOO-2011, and so it was hoped teams could better exploit this new annotated resource to produce better systems.

The second HOO shared task was thus more successful than the first, and of the 26 different teams that registered interest, 14 submitted output. The top team still only achieved a score of 28.7 F_1 for correction, using a combination of classifiers and a language model (Dahlmeier et al., 2012), but the competition was a lot closer, with 3 other teams scoring above 24 F_1 (Quan et al., 2012; Rozovskaya et al., 2012; Sakaguchi et al., 2012a). Despite this improvement however, half of all teams still scored less than 9 F_1 (Bhaskar et al., 2012; Daudaravicius, 2012; Kochmar et al., 2012; Lee et al., 2012; Lynch et al., 2012; van den Bosch and Berck, 2012; Zesch and Haase, 2012).

Across both HOO tasks, classifiers were the most popular approach by far, and almost all teams used at least one in their system. It might therefore be surprising that so many teams achieved such a wide range of results, but this can be accounted for by the fact that both the features chosen and overall system architecture can have a significant effect on system performance. Finally, language models were also reasonably popular, appearing in just over half of all systems, while rules were only used in a quarter of all systems. All teams, their approaches and results for both HOO tasks are shown in Table 2.5.

2.4.2 Conference on Natural Language Learning

Following on from HOO-2012, the organisers of the Conference on Natural Language Learning (CoNLL) 2013 shared task (Ng et al., 2013) decided to increase the difficulty of the task by requiring participants to correct noun number, subject-verb agreement (SVA) and verb form errors in addition to determiner and preposition errors. They also released NUCLE (Section 2.1.3) as the official training data and changed the official metric to the M2 scorer (Section 2.3.1).

Despite the increased difficulty, CoNLL-2013 was also a success, and of the 54 teams that registered interest, 17 ultimately submitted system output. Classifiers were again the most popular approach, with more than half of all teams using at least one in their systems, but four teams also tried applying statistical machine translation SMT for the first time in a shared task (Buys and van der Merwe, 2013; Wilcox-O'Hearn, 2013; Yoshimoto et al., 2013; Yuan and Felice, 2013). A pure classifier approach still came first overall however, achieving a score of 31.2 F_1 (Rozovskaya et al., 2013), while a purely language model based approach came second with 25.01 F_1 (Kao et al., 2013).

CoNLL-2013

Team	Approach	P	R	F_1	Reference
CAMB	MT	39.15	10.10	16.06	Yuan and Felice (2013)
HIT	CL, RB	35.65	16.56	22.61	Xiang et al. (2013)
IITB	CL, RB	28.18	4.99	8.48	Kunchukuttan et al. (2013)
KOR	CL	43.88	3.71	6.85	Yi et al. (2013)
NARA	MT, CL, LM	27.39	18.62	22.17	Yoshimoto et al. (2013)
NTHU	LM	23.80	**26.35**	25.01	Kao et al. (2013)
SAAR	CL, RB	27.69	1.10	2.11	Putra and Szabo (2013)
SJT1	CL, LM	40.18	10.96	17.22	Jia et al. (2013)
SJT2	-	13.33	0.24	0.48	-
STAN	RB	25.50	4.69	7.92	Flickinger and Yu (2013)
STEL	MT	27.00	13.33	17.85	Buys and van der Merwe (2013)
SZEG	CL	5.52	3.16	4.02	Berend et al. (2013)
TILB	CL	6.25	7.24	6.71	van den Bosch and Berck (2013)
TOR	MT	17.67	4.81	7.56	Wilcox-O'Hearn (2013)
UAB	RB	12.42	1.22	2.22	Sidorov et al. (2013)
UIUC	CL	**46.45**	23.49	**31.20**	Rozovskaya et al. (2013)
UMC	CL, LM, RB	28.49	17.53	21.70	Xing et al. (2013)

CoNLL-2014

Team	Approach	P	R	$F_{0.5}$	Reference
AMU	MT, LM	41.62	21.40	35.01	Junczys-Dowmunt and Grundkiewicz (2014)
CAMB	MT, LM, RB	39.71	**30.10**	**37.33**	Felice et al. (2014)
CUUI	CL	41.78	24.88	36.79	Rozovskaya et al. (2014a)
IITB	MT, CL	30.77	1.39	5.90	Kunchukuttan et al. (2014)
IPN	LM, RB	11.28	2.85	7.09	Hernandez and Calvo (2014)
NARA	-	21.57	29.38	22.78	-
NTHU	CL, LM, RB	35.08	18.85	29.92	Wu et al. (2014)
PKU	CL, LM, RB	32.21	13.65	25.32	Zhang and Wang (2014)
POST	LM, RB	34.51	21.73	30.88	Lee and Lee (2014)
RAC	LM, RB	33.14	14.99	26.68	Boroş et al. (2014)
SJTU	CL, RB	30.11	5.10	15.19	Wang et al. (2014a)
UFC	RB	**70.00**	1.72	7.84	Gupta (2014)
UMC	MT	31.27	14.46	25.37	Wang et al. (2014b)

Table 2.6: Table showing the teams, approaches, and their scores for correction in the CoNLL-2013 and CoNLL-2014 shared tasks. MT = machine translation, CL = classifier, LM = language model, RB = rule-based. SJTU2 in CoNLL-2013 and NARA in CoNLL-2014 did not submit system description papers.

Finally, having now given the community several years to prepare, the organisers of CoNLL-2014 (Ng et al., 2014) again decided to increase the difficulty of the task and this time required participants to correct all error types as in HOO-2011. They also changed the official metric from F_1 to $F_{0.5}$, which weights precision twice as much as recall, in an effort to encourage system submissions to be more precise than to necessarily correct every error. This time, participation was a lot higher than in HOO-11, and of the 45 teams that registered interest, 13 submitted output.

Despite their success in previous tasks, classifiers were not as common in CoNLL-2014 as they had been before. This is likely because classifiers do not extend well to general error correction and it is impractical to build lots of separate classifiers for multiple error types. Instead, rule-based and language model approaches were most common, being employed by more than half of all teams.

The most significant result however, was that of the top 3 teams, the teams that came first and third respectively both used SMT in their approach; specifically, they scored 37.33 $F_{0.5}$ (Felice et al., 2014) and 35.01 $F_{0.5}$ (Junczys-Dowmunt and Grundkiewicz, 2014). In contrast, a purely classifier approach came a close second with 36.79 $F_{0.5}$ (Rozovskaya et al., 2014a) and a language model and rule-based approach came a distant fourth with 30.88 $F_{0.5}$ (Lee and Lee, 2014). Although the use of SMT may not be particularly surprising given its ability to correct all error types simultaneously, this result nevertheless paved the way for SMT to become dominant in the years after the shared task. The approaches and results of all the teams are summarised in Table 2.6.

2.4.3 Recent work

Since the end of the shared tasks, almost all new GEC systems have incorporated machine translation into their approach. Whilst the earliest approaches did this by means of system combination with SMT and classifiers (Susanto et al., 2014; Rozovskaya and Roth, 2016), there has also been a lot of work on re-ranking n-best lists (Hoang et al., 2016; Mizumoto and Matsumoto, 2016; Yuan et al., 2016; Yannakoudakis et al., 2017). Although the best system combination approach improved the F-score on the CoNLL-2014 test set by over 10 $F_{0.5}$ (from 37.33 to 47.40), it did so using non-public Lang-8 data so results are incomparable. In contrast, most re-ranking approaches only improved performance by under 5 $F_{0.5}$, although these were also trained on different combinations of public and private datasets. The only exception to the above is Yannakoudakis et al. (2017), who used a neural detection system (Rei and Yannakoudakis, 2016) to re-rank their output instead of a more conventional classifier, and ultimately reported a score of 51.08 $F_{0.5}$.

All the other approaches to GEC using SMT either focus on tuning the system (Junczys-Dowmunt and Grundkiewicz, 2016) or else adding new features (Chollampatt et al., 2016a,b; Chollampatt and Ng, 2017). Specifically, Junczys-Dowmunt and Grundkiewicz (2016) achieved 49.49 $F_{0.5}$ by tuning their system on public data with a large language model, while the current best SMT approach combines SMT with a neural joint model and large language model to reach 53.14 $F_{0.5}$ (Chollampatt and Ng, 2017).

In addition to SMT, there has also been a growing trend towards using neural machine translation (NMT) (or other sequence-to-sequence models) in GEC. Although the first of these systems was purely token-based (Yuan and Briscoe, 2016), subsequent work found that incorporating character-based information into the model tended to improve scores by up to 8 $F_{0.5}$, most likely

CoNLL-2014							
Name	**Approach**	**Data**	**P**	**R**	**$F_{0.5}$**	**GLEU**	**Reference**
Statistical Machine Translation							
$CU16_{smt}$	MT, RR	CLC	-	-	38.08	65.68	Yuan et al. (2016)
NU14	MT, CL	NUCLE, Lang-$8_{en.v1}$, wiki	53.55	19.14	39.39	-	Susanto et al. (2014)
TU16	MT, RR	NUCLE, Lang-8_{v2}, GW	45.80	26.60	40.00	66.10	Mizumoto and Matsumoto (2016)
$NU16_{rank}$	MT, RR	NUCLE, Lang-$8_{en.v1}$, wiki	50.35	23.84	41.19	-	Hoang et al. (2016)
$NU16_{glm}$	MT, NJM, NGLM	NUCLE, Lang-$8_{en.v1}$, wiki	52.34	23.07	41.75	-	Chollampatt et al. (2016b)
$CU17_{CU16.smt}$	MT, NRR	NUCLE, Lang-$8_{en.v1}$, CLC, JFLEG	51.09	25.30	42.44	66.42	Yannakoudakis et al. (2017)
$NU16_{jm}$	MT, NJM	NUCLE, Lang-8_{v2}, FCE, wiki	-	-	44.27	-	Chollampatt et al. (2016a)
VT16	MT, CL	NUCLE, Lang-8_{priv}, wiki, Web1T	60.17	25.64	47.40	-	Rozovskaya and Roth (2016)
$AM16_{pub}$	MT	NUCLE, Lang-$8_{en.v1}$, CC	61.27	27.98	49.49	-	Junczys-Dowmunt and Grundkiewicz (2016)
$CU17_{AM16.pub}$	MT, NRR	NUCLE, Lang-$8_{en.v1}$, CLC, JFLEG	59.88	32.16	51.08	68.69	Yannakoudakis et al. (2017)
$AM16_{priv}$	MT	NUCLE, Lang-8_{priv}, CC	63.52	30.49	52.21	-	Junczys-Dowmunt and Grundkiewicz (2016)
NU17	MT, NJM	NUCLE, Lang-8_{v2}, wiki, CC	62.74	32.96	53.14	-	Chollampatt and Ng (2017)
Neural Machine Translation							
$CU16_{nmt}$	NMT	CLC	-	-	39.90	65.59	Yuan and Briscoe (2016)
SU16	NMT	NUCLE, Lang-$8_{en.v1}$, CC	49.24	23.77	40.56	-	Xie et al. (2016)
HU17	NMT	NUCLE, Lang-8_{v2}	-	-	41.37	-	Schmaltz et al. (2017)
MS17	NMT	NUCLE, Lang-$8_{en.v1}$, CLC, CC	-	-	45.15	-	Ji et al. (2017)
MS18	NMT	NUCLE, Lang-$8_{en.v1}$, Lang-8_{priv}, CLC, wiki	61.24	37.86	54.51	-	Ge et al. (2018a)
NU18	NMT	NUCLE, Lang-8_{v2}, wiki, CC	65.49	33.14	54.79	-	Chollampatt and Ng (2018a)
$AM18_{nmt}$	NMT	NUCLE, Lang-8_{v2}, CC	61.90	40.20	55.80	-	Junczys-Dowmunt et al. (2018)
$AM18_{hyb}$	NMT, MT	NUCLE, Lang-8_{v2}, CC	66.77	34.49	56.25	-	Grundkiewicz and Junczys-Dowmunt (2018)

Table 2.7: A summary of all results on the CoNLL-2014 test set for various groups in various years since the end of the shared task. MT = machine translation, RR = re-ranking, CL = classifier, JM = joint model, GLM = global lexicon model. Any of these prefixed with N = neural. Most datasets are described in Section 2.1. Of those that are not, GW = Gigaword, wiki = English Wikipedia and CC = CommonCrawl. In addition to Web1T, these are all large monolingual English corpora used to create language models.

because this enhanced the model's ability to handle spelling and morphological errors (Xie et al., 2016; Schmaltz et al., 2017; Ji et al., 2017). To further improve upon these systems, most recent work either uses large quantities of non-public data (Ge et al., 2018a) or else highly optimises every component of their system, e.g. word embeddings and language models (Chollampatt and

JFLEG test

Name	Approach	Data	P	R	$F_{0.5}$	GLEU	Reference
$NU16_{jm}$	MT, NJM	NUCLE, Lang-8_{v2}, FCE, wiki	68.08	32.30	55.73	50.13	Chollampatt et al. (2016a)
$CU16_{nmt}$	NMT	CLC	65.66	35.93	56.34	52.05	Yuan and Briscoe (2016)
JH17	NMT	NUCLE, Lang-$8_{en.v1}$, FCE, JFLEG	65.80	40.96	58.68	53.98	Sakaguchi et al. (2017b)
NU17	MT, NJM	NUCLE, Lang-8_{v2}, wiki, CC	-	-	64.25	56.78	Chollampatt and Ng (2017)
NU18	NMT	NUCLE, Lang-8_{v2}, wiki, CC	-	-	66.80	57.47	Chollampatt and Ng (2018a)
MS18	NMT	NUCLE, Lang-$8_{en.v1}$, Lang-8_{priv}, CLC, wiki	-	-	-	57.74	Ge et al. (2018a)
$AM18_{nmt}$	NMT	NUCLE, Lang-8_{v2}, CC	-	-	-	59.90	Junczys-Dowmunt et al. (2018)
$AM18_{hyb}$	MT, NMT	NUCLE, Lang-8_{v2}, CC	-	-	-	61.50	Grundkiewicz and Junczys-Dowmunt (2018)

FCE test

Name	Approach	Data	P	R	$F_{0.5}$	GLEU	Reference
$CU16_{smt}$	MT, RR	CLC	63.27	31.95	52.90	70.15	Yuan et al. (2016)
$CU16_{nmt}$	NMT	CLC	-	-	53.49	71.16	Yuan and Briscoe (2016)
$CU17_{CU16.smt}$	MT, NRR	NUCLE, Lang-$8_{en.v1}$, CLC, JFLEG	64.25	36.13	55.60	71.76	Yannakoudakis et al. (2017)
$CU17_{AM16.pub}$	MT, NRR	NUCLE, Lang-$8_{en.v1}$, CLC, JFLEG	43.34	19.88	35.07	64.78	Yannakoudakis et al. (2017)

Table 2.8: A summary of all results on the JFLEG and FCE test sets for various groups in various years since the end of the shared task. See Table 2.7 for acronym definitions.

Ng, 2018a; Junczys-Dowmunt et al., 2018). The current state-of-the-art score on CoNLL-2014 also does this, ultimately achieving a score of 56.25 $F_{0.5}$ using a combination of NMT and SMT (Grundkiewicz and Junczys-Dowmunt, 2018). The results for all these systems, as well as what data they were trained on, are shown in Table 2.7.

Although most of these systems are typically only evaluated on CoNLL-2014, a handful have also been evaluated on JFLEG and the FCE. These systems are typically identical to the systems tested on CoNLL-2014, except they have been trained and tuned on different combinations of public and private data using different metrics. In particular, since JFLEG was originally released alongside the GLEU metric, it is unsurprising that most systems evaluate on JFLEG using GLEU rather than M2. This ultimately means, however, that researchers may build two different versions of the same system, one tuned for each metric, depending on the target test set. The generalisability of these systems is hence unclear.

Nevertheless, the same NMT and SMT system combination that scored highest on CoNLL-2014 also has the highest GLEU score of 61.50 on JFLEG test (Grundkiewicz and Junczys-Dowmunt, 2018). In contrast, although very few systems are evaluated on the public FCE, the current highest score is 55.60 $F_{0.5}$ and 71.76 GLEU (Yannakoudakis et al., 2017). Table 2.8 shows all the results and datasets used in systems evaluated on both JFLEG and the FCE.

CHAPTER 3

CORPUS PREPROCESSING

One of the main goals of this thesis is to develop a system capable of automatically annotating parallel text with error types. For example, given the following original and corrected sentence pair:

> He only can look at the TV in the night.
> He can only watch TV at night.

We want to extract the edits that transform the former to the latter and classify them:

He	**only can**	**look at**	**the**	TV	**in**	**the**	night	.
He	**can only**	**watch**		TV	**at**		night	.
	R:WO	**R:VERB**	**U:DET**		**R:PREP**	**U:DET**		

Before we can do this however, we must first consider existing datasets and decide how to preprocess them. This is not a trivial task because all the datasets introduced in Section 2.1 are variously available in different file formats that are either tokenized or untokenized, with or without explicit error annotations. Since another goal of this thesis is to automatically standardise different corpora, it is important that we deal with these differences as consistently as possible.

This chapter hence covers topics such as how to extract in-line character edits from XML text, how to convert character edits to token edits, and how to tokenize sentences when annotators change sentence boundaries. Much of the work presented in this chapter has previously been reported in Bryant and Felice (2016).

3.1 The CLC and Public FCE

As the full CLC is not publicly available, this first section mainly describes the format of the public FCE. Both datasets have very similar formats however, and so much of this section also applies to the CLC.

The public FCE is available online in two different formats: one which contains the raw data (Yannakoudakis et al., 2011), and one which was processed for error detection (Rei and Yannakoudakis, 2016).[1] Since the raw data release is not explicitly split into training, development and test sets however, I recovered this split using the file IDs in the error detection release. This means there are 1,061 files for training, 80 for development and 97 for testing (1,238 total).[2]

Each file in the training, development and test sets contains two essays written by the same author on various topics. Each essay is enclosed by <coded_answer> tags which are further subdivided into paragraphs on <p> tags. The CLC differs in this regard in that it a) stores essay text between <text> tags rather than <coded_answer> tags and b) does not split essays into paragraphs.

One feature of the FCE XML structure is that the edit tags are built directly into the essay text:

```
<p>
  This
    <NS type="AGV">
      <i>are</i>
      <c>is</c>
    </NS>
  a paragraph.
</p>
```

Figure 3.1: Example XML paragraph structure in the public FCE.

Here, <NS> denotes the start of an edit (<e> in the CLC), denotes the incorrect original string, and <c> denotes the correction string. This structure hence encodes the edit [*are* → *is*] in the context "This are a paragraph." The error type has also been labelled AGV, which is a verb agreement error (see Table 2.2 for the complete list of error types in the CLC).

In all, there are four basic types of XML edit in any given <NS> tag:

1. No or <c> : <NS>text</NS>
2. Only : <NS>text</NS>
3. Only <c> : <NS><c>text</c></NS>
4. Both and <c> : <NS>text<c>text</c></NS>

Specifically, 1 denotes a mistake that an annotator identified but was unable to correct, while 2, 3 and 4 respectively denote unnecessary, missing and replacement word errors. The CLC and FCE also permit nested edits (i.e. edits within edits), and so it is also possible for other <NS> tags to appear inside any of these four types. An example of a nested edit is shown in Figure 3.2, where the string *entery* is first identified as a spelling error and corrected to *entry* and second identified as a replacement noun error and corrected to *entrance*. This nesting, while fairly rare,

[1] https://ilexir.co.uk/datasets/index.html

[2] The remaining 6 files were only used for validity testing in Yannakoudakis et al. (2011) and are discarded.

```
<p>
  I will wait at the
    <NS type="RN">
      <i>
        <NS type="S">
          <i>entery</i>
          <c>entry</c>
        </NS>
      </i>
      <c>entrance</c>
    </NS>
  .
</p>
```

Figure 3.2: Example nested edit in the public FCE.

	None	<i>	<c>	<i><c>	<NS>	<i><NS>	<c><NS>	<i><c><NS>	Total
CLC	92,306	295,227	618,614	1,883,647	17,007	2,967	4	119,879	3,029,651
FCE train	1,328	4,410	8,224	28,510	245	38	0	1,767	44,522
FCE dev	105	304	761	2,218	20	3	0	126	3,537
FCE test	209	458	907	2,931	51	4	0	216	4,776

Table 3.1: Table showing the counts for XML edit types in both the public FCE and full CLC. For example, there are 304 regular unnecessary word errors in FCE dev (<i>), but also 3 unnecessary word errors that contain a nested edit (<i><NS>).

makes processing this data much more complicated as there is also no limit to the amount of nesting that can occur.

The parallel original and corrected paragraphs in an FCE essay can thus be regenerated as follows:

- Any text inside an <i> tag is saved in the original paragraph alone.
- Any text inside a <c> tag is saved in the corrected paragraph alone.
- All other text inside or outside an <NS> tag is saved on both sides.

The only exception to this is if the edit is a nested edit, in which case only the outermost <i> and <c> text is saved and the intermediate corrections are ignored. In Figure 3.2, this means the outermost edit [*entery* → *entrance*] is saved and the intermediate edit [*entery* → *entry*] is ignored. The character edit spans that map the original to the corrected paragraphs are also saved.

The total counts of all the XML edit types in both the public FCE and full CLC are shown in Table 3.1. Note that this table only counts the outermost <NS> tags so nested edits are not counted twice (or more). In total, roughly 5% of all edits in each dataset contain at least one nested error. While there are no instances of nested edits appearing inside a lone <c> tag anywhere in the public FCE (column <c><NS>), this does happen 4 times in the CLC. It seems unlikely that annotators found errors within their own corrections however, so this is most likely the result of annotator error.

```
<TEXT>
  <P>
    This are a sentence.
  </P>
</TEXT>
<ANNOTATION>
  <MISTAKE start_par="0" start_off="5" end_par="0" end_off="8">
    <TYPE>SVA</TYPE>
    <CORRECTION>is</CORRECTION>
  </MISTAKE>
</ANNOTATION>
```

Figure 3.3: Example SGML paragraph structure

```
S This are a sentence .
A 1 2|||SVA|||is|||REQUIRED|||-NONE-|||0
```

Figure 3.4: Example M2 sentence structure

3.2 NUCLE and CoNLL

The NUCLE corpus is also available online, but can only be obtained after submitting a signed license form agreeing to use it only for research purposes.[3] In contrast, the CoNLL-2013 and CoNLL-2014 test sets are freely available. All these datasets are also available in two different formats: SGML and M2.

In SGML format (Figure 3.3), the original essay text is enclosed by `<TEXT>` tags which are further subdivided by title `<TITLE>` and paragraph `<P>` tags. Unlike the FCE however, edits are not incorporated into the main text and are instead represented as a list of `<MISTAKE>` tags enclosed by `<ANNOTATION>` tags. Each `<MISTAKE>` thus contains both the character and paragraph offsets of the edit, as well as the error type `<TYPE>` and correction `<CORRECTION>`.

In contrast, M2 format (Figure 3.4) is the cleaner, preprocessed version of the SGML where the paragraphs have not only been word and sentence tokenized, but the character edits have also been converted to token edits. An edit line in M2 format thus contains the start and end token offset of the edit, the error type, the tokenized correction string, a flag indicating whether the edit is required or optional, a comment field, and a unique annotator ID.[4] This format was also the official format used in the CoNLL shared tasks.

With respect to edit types, the same four edit operations in the FCE are also available in NUCLE with the following differences:

1. Identified but not corrected edits are labelled Unclear Meaning (`Um`).
2. Missing word edits must span and repeat an adjacent unchanged word in the correction.
3. Nested edits are not allowed.

[3] `https://www.comp.nus.edu.sg/~nlp/conll14st.html`

[4] In practice, the optional edit and comment fields often contain dummy values and are not used.

	SGML Total	M2 Total
NUCLE	44,912	44,482
CoNLL-2013	3,424	3,412
CoNLL-2014.0	2,397	2,391
CoNLL-2014.1	3,331	3,317

Table 3.2: Counts for the total number of edits in NUCLE and CoNLL.

The reason unchanged words must be repeated in missing word edits is because the NUCLE annotation platform did not allow empty strings on the original side of an edit. This meant, for example, that an annotator had to change [*want* → *want to*] or [*go* → *to go*] to fix the missing word error in the sentence "I want go home". One consequence of this, however, is that missing word errors look similar to substitution errors, and so Table 3.2 just reports the total number of edits in NUCLE and CoNLL. Note that CoNLL-2014 features twice in this table because it was doubly annotated by two different annotators, and that the counts respectively represent the total number of SGML and M2 edits before and after official shared task preprocessing.

3.3 From characters to tokens

Although the character edits in NUCLE and CoNLL have already been converted to token edits in M2 format, the same cannot be said of the CLC and FCE. This meant I had to develop my own method to convert character edits to token edits, and so I also took the opportunity to reprocess NUCLE and CoNLL in an effort to standardise corpus preprocessing. I hence first modified or removed several SGML edits that met certain undesirable criteria:

- The correction string of all Unclear Meaning (`Um`) edits was set to be the same as the original string because we do not want to delete text that an annotator was unable to correct.
- All Citation (`Cit`) edits were removed because they are not only extremely rare, but also inconsistent in terms of whether the error is corrected or not. It is also debatable whether citation errors qualify as grammatical errors given that many genres, such as fiction or correspondence, do not require citations.
- Edits that cross paragraph boundaries or select entire paragraphs were also removed because they normally consist of annotator comments that suggest an essay or paragraph be rewritten.
- Edits that contain an ellipsis '...' in the correction were removed because some annotators occasionally used this to denote a long sequence of unchanged text. Such corrections would be interpreted literally however.
- Edits that overlapped with any previously seen edits were removed because overlapping edits were not allowed in NUCLE; these were assumed to be the result of annotator error.

While the majority of the remaining character edits in both NUCLE and the FCE mapped cleanly to token edits, several exceptions made this conversion more complicated. For example, consider the following tokens and character edits:

1. Token: *WORD.* Edit: [. → ,]
2. Token: *Forest'view* Edit: [*Forest'* → *Forest's*]
3. Token: *dancing* Edit: [*ing* → *ed*]
4. Token: *klever* Edit: [*kleve* → *clever*]
5. Token: *To* Edit: [*T* → *to*]

In the first example, the annotator wanted to change a full stop into a comma, but the tokenizer did not separate it from the previous word. This is mostly likely because the previous word is entirely in upper case and so the tokenizer mistook it for an acronym or proper noun, but the result is that the character edit does not cleanly map to a token edit. A similar thing happens in the second example, where the missing whitespace between *Forest'* and *view* causes these two tokens to be erroneously merged. While the annotator was able to mentally tokenize this string when making their correction however, automatic tokenizers are unable to do the same.

Although annotators are typically instructed to only annotate whole words when making corrections, they do not always do this in practice. The third example hence shows a case where an annotator wanted to change the form of a token [*dancing* → *danced*], but did so only by editing its morphology [*ing* → *ed*]. Such edits naturally do not precisely map to tokens.

Finally, the last two cases are examples of annotator error. Specifically, the annotators accidentally omitted a character from their edit spans but still provided a complete correction. Applying the character level edit in the fourth example hence produces the word *cleverr*, while applying the edit in the fifth example produces the word *too*; in other words, the omitted character is duplicated on the end of the correction string. The last case is particularly noteworthy because it also unintentionally produces a valid word.

3.3.1 Word tokenization

In an effort to resolve the above tokenization problems, I tokenized the original text with spaCy[5] v1.9.0, an open source NLP library (Honnibal and Johnson, 2015). One advantage of spaCy over other libraries is that it performs non-destructive tokenization, which means it also keeps track of the character offsets of the tokens it separates; this makes it a lot easier to map characters to tokens.

Before counting the number of mismatches between character spans and token spans in each dataset, I also stripped the leading and trailing whitespace from the character edits to minimise the error rate; in the same way that annotators sometimes omit characters from the end of their

[5] `https://spacy.io/`

	Mismatches	Total Edits	Percent
CLC	22,515	3,029,651	0.74%
FCE train	168	44,522	0.38%
FCE dev	20	3,537	0.57%
FCE test	14	4,776	0.29%
NUCLE	2,538	44,912	5.65%
CoNLL-2013	2	3,424	0.06%
CoNLL-2014.0	8	2,397	0.33%
CoNLL-2014.1	2	3,331	0.06%

Table 3.3: The total number of character-to-token mismatches in relation to the total number of edits in various datasets.

edit spans, they also sometimes add extra characters. The total number of character-to-token mismatches in each annotated dataset is hence shown in Table 3.3.

Although character mismatches typically account for fewer than 1% of all edits in most datasets, the most surprising result is that this figure rises to over 5.5% in NUCLE. Upon closer inspection however, the main reason for this seems to be that NUCLE contains many more morphological edits (e.g. [*ing* $\rightarrow$ *ed*]) than other datasets, which perhaps indicates ambiguity in the annotation guidelines.

3.3.2 Just add whitespace

One solution to the character-to-token alignment problem is to artificially surround each edit with whitespace before tokenization. In particular, if we make the assumption that annotators only edit complete tokens, this implies there is a token boundary before and after each edit. Consequently, artificially surrounding each edit with additional whitespace ensures that the tokenizer will tokenize the text at these positions and that character edit spans will always map to token edit spans.

One disadvantage to this approach, however, is that whilst it correctly handles the *WORD.* and *Forest'view* cases mentioned at the start of this subsection, it fails to handle the other three cases. Specifically, the edits [*dancing* $\rightarrow$ *danced*], [*klever* $\rightarrow$ *clever*] and [*To* $\rightarrow$ *to*] are all respectively realised as [*danc ing* $\rightarrow$ *danc ed*], [*kleve r* $\rightarrow$ *clever r*] and [*T o* $\rightarrow$ *to o*]. Although this may be acceptable given that the last two cases are caused by annotator error anyway, an additional side effect of inserting whitespace around each edit is that it fundamentally changes the original text for each annotator.

To give an example, again consider the string *Forest'view*; while one official CoNLL annotator edited the substring [*Forest'* $\rightarrow$ *Forest's*] as previously discussed, the other edited the whole string [*Forest'view* $\rightarrow$ *Forest's view*]. If we now insert whitespace around each edit however, the original text for the first annotator becomes *Forest'␣view* (i.e. with whitespace), but remains *Forest'view* for the second annotator (i.e. without whitespace). This ultimately means we end up

with two different tokenized versions of the same original text.

This is highly undesirable however, as we do not want the annotations of one annotator to affect the tokenization of all other annotators.

3.3.3 Growing character spans

An alternative to adding whitespace around an edit is to instead grow its character span to align with the nearest token boundary. To give an example, consider the token *dancing*, which an annotator wanted to change into *danced* by means of the correction [*ing* → *ed*]. While the character span of the edit is 4 to 7, the character span of the token is 0 to 7, so we must grow the edit span to agree with the token span.

We can do this by calculating the character difference between the edit start span and the token start span to learn the extent of the mismatch: in this case $4-0=4$. By subtracting this value from the character start span, we hence obtain a new character span that corresponds to a token boundary. While this solves the character alignment problem, the second step is to also update the correction string. Specifically, as we have now increased the size of the edit span by 4 characters, we must also add these 4 characters to the start of the correction string. In this case, this means we add *danc* to both the original and corrected side of the edit to produce [*dancing* → *danced*]. Note that a similar process can be applied to the end of an edit span.

One extremely rare complication to the above concerns a token that has been partially annotated more than once. For example, perhaps the annotator wanted to change [*dancing* → *Danced*] by means of 2 edits: [*d* → *D*] and [*ing* → *ed*]. Since neither of these edits maps to a complete token however, growing the character span will produce 2 edits for the same token: [*dancing* → *Dancing*] and [*dancing* → *danced*]. As it is more complicated to recover the intended edit [*dancing* → *Danced*] however, I arbitrarily chose to just keep the first edit when this happened.

The main advantage of this approach over the whitespace approach is that it correctly handles all the cases where annotators only edited morphology or capitalization. Since this happens most frequently in NUCLE, it should hence be no surprise that this was also the approach adopted in the CoNLL shared tasks where NUCLE was the training data.

The disadvantage of this approach, however, is that character span expansion can also have unintended consequences. For example, growing [*Forest'* → *Forest's*] in the string *Forest'view*, results in the edit [*Forest'view* → *Forest'sview*] which, unlike the whitespace approach, is not what the annotator intended. Having said that, at least this approach does not modify the original text in any way and is hence more compatible with multiple annotators.

Ultimately, although neither option is perfect, the growing character span method has fewer disadvantages than the adding whitespace method and so that is what I used to convert character spans to token spans.

```
S Back to a hundred years ago , water was never a concern .
A 0 2|||Rloc-|||||REQUIRED|||-NONE-|||0
A 2 3|||Mec|||A|||REQUIRED|||-NONE-|||0
A 12 13|||Mec|||,|||REQUIRED|||-NONE-|||0

S While , today it is different .
A 0 1|||Mec|||while|||REQUIRED|||-NONE-|||0
A 1 2|||Mec|||||REQUIRED|||-NONE-|||0
```

Figure 3.5: An example "sentence" pair in NUCLE where the annotator wanted to combine the two original strings as a single sentence.

3.4 From paragraphs to sentences

Having converted character edits to token edits, the next step is to sentence tokenize each paragraph and redefine the edits in terms of sentences. This can be non-trivial however, because annotators can also change sentence boundaries.

To give an example, consider Figure 3.5, which shows two adjacent sentences in NUCLE (M2 format). Although these sentences were separated by the tokenizer, the edits suggest that the annotator actually wanted to combine them. Specifically, the annotator changed the first sentence final full stop into a comma [. $\rightarrow$,], and lower cased the first character of the next sentence [*While* $\rightarrow$ *while*]. We must thus decide whether it is better to separate these sentences, as in the original text, or combine them, as in the corrected text. In fact this is especially important given that most GEC systems are sentence-based, yet we may not want them to learn that it is acceptable to end a sentence with a comma or begin one with a lower case character.

Although it might hence seem more intuitive to merge sentences such that the original and corrected text constitutes at least one full sentence on both sides, it is also worth remembering that this is only possible because we already have access to the corrected text. This is not the case in a realistic grammar checking scenario however, and so it is inadvisable to artificially condition systems to expect full sentences. I consequently decided to sentence tokenize based solely on the original text in order to remain as faithful to the task as possible.

3.4.1 Paragraph edits to sentence edits

Having tokenized the original paragraph, the next step is to update the edit spans. Specifically, there are three different types of scenario when mapping paragraph edits to sentence edits:

1. Edit starts and ends inside a tokenized sentence.
2. Edit starts and ends on a sentence boundary.
3. Edit starts in one sentence and ends in another.

While the majority of edits fall into the first category and hence straightforwardly map to sentence edits, the other two categories are a little more complicated. The second category

in particular is especially problematic given that missing token edits on a sentence boundary ambiguously belong to either the end of one sentence or start of the next. I nevertheless resolved this problem by assigning edits that ended with a punctuation character to the end of the sentence and to the start of the next sentence otherwise. The intuition behind this was that English sentences rarely start with punctuation characters, so missing punctuation is much more likely to occur at the end of a sentence. Finally, I simply ignored edits that crossed sentence boundaries (less than half a percent of all edits); it is very difficult to automatically split an edit such that the span and correction still makes sense on both sides.

This is the final stage of preprocessing, at which point I have now transformed untokenised text with character edits into tokenized sentences with token edits.

3.5 Lang-8 and JFLEG

Although most of this section has focused on how to preprocess untokenized datasets with character edits, not all datasets are available in this format. The Lang-8 Corpus of Learner English v1.0 (Tajiri et al., 2012) is a good example of this, in that it has not only already been word and sentence tokenized, but it also does not contain explicit error annotations of any kind.[6] Instead, the corpus has the following format:

```
<cor_count> <serial_id> <URL> <sent_num> <orig_sent> [<cor_sent1> ...]
```

Specifically, `<cor_count>` represents the total number of alternative corrections for a sentence, `<serial_id>` and `<URL>` identify the serial number and URL of the sentence, `<sent_num>` is the ID of the sentence in the text (0 is the title), `<orig_sent>` is the original sentence itself, and `[<cor_sent1> ...]` is a list of corrected sentences for the original sentence (if any) that total `<cor_count>`. The original and corrected sentences have also already been tokenized by an unknown tokenizer. I consequently do not need to preprocess Lang-8 because I can simply extract the parallel original and corrected sentence pairs.

The JFLEG corpus[7] is similarly straightforward to process in that it already consists of parallel original and corrected text files and does not contain any additional metadata. The text has also already been word and sentence tokenized.

Perhaps the only disadvantage of Lang-8 and JFLEG is that by being tokenized, we cannot properly globally standardise all GEC corpora using the same tokenization. This ultimately means different corpora may have different tokenization, which may have a small effect on system performance.

[6] `http://cl.naist.jp/nldata/lang-8/`
[7] `https://github.com/keisks/jfleg`

3.6 Summary

In this chapter, I described how to preprocess various corpora and keep track of the token edits that map an original sentence to a corrected sentence. Specifically, I discussed how to regenerate parallel original and corrected paragraphs from various file formats and also convert character edit spans into token edit spans. Although character-to-token span mismatches typically occur in less than 1% of all edits, this rises to over 5.5% in NUCLE, which shows it is important to resolve this problem rather than simply ignore the affected edits. Having thus converted character edits to token edits, a second step described how to sentence tokenise the text and recalculate the edit spans even when annotators changed sentence boundaries. The next chapter will describe how to automatically align the preprocessed original and corrected sentences in a linguistically intuitive manner.

CHAPTER 4

LINGUISTICALLY ENHANCED SENTENCE ALIGNMENT

Having preprocessed various datasets, I now have access to all the parallel original and corrected tokenized sentence pairs in all corpora. Although I also have the set of gold edits that explicitly map the former to the latter in the CLC, public FCE, NUCLE and CoNLL, I do not have such a set of edits for Lang-8 and JFLEG. This meant I had to extract these edits automatically from the text. Ultimately, this is an alignment problem (Table 4.1).

We	took	a	**guide**	tour	**on**		**center**	**city**	.
We	took	a	**guided**	tour	**of**	**the**	**city**	**center**	.

Table 4.1: A sample alignment between parallel original and corrected sentences.

While I could use tools such as GIZA++ (Och and Ney, 2003), a sentence alignment toolkit more commonly used in SMT, to align the parallel original and corrected sentences, the problem in GEC is unlike SMT in that it involves the alignment of text in the *same* language. This simplifies the task somewhat, given that the majority of tokens in the parallel GEC sentences remain the same on both sides, and so we can instead use something less sophisticated. This also means we do not have to rely on parallel training data which might inherently learn corpus biases.

This chapter hence introduces a linguistically enhanced sentence alignment algorithm designed to extract edits in as human-like a way as possible. Specifically, the algorithm is an extension of the Levenshtein alignment algorithm (Levenshtein, 1966) that also takes linguistic features such as lemmas and POS tags into account. It is further enhanced by a set of merging rules to capture multi-token edits, and was evaluated on several datasets, achieving 80-85 F_1 when compared against human edits (Felice et al., 2016).[1]

[1]The scores in this thesis are different from the paper because I extended and re-ran the experiments using more up-to-date software.

It is worth mentioning that although the original motivation for this approach was to extract edits from Lang-8 and JFLEG, this is not its only application. For example, automatic annotation is more consistent than human annotation and so can also be used to standardise existing edit spans. In particular, errors such as *has eating* are inconsistently corrected as [*has* → *was*] or [*has eating* → *was eating*] even though they equate to the same thing, while edits such as [*has eating* → *was eaten*] are also variously realised as one or two edits: [*has* → *was*] and [*eating* → *eaten*]. The advantage of automatic alignment is hence that we can regularise whether such edits are split or merged and thus reduce ambiguity in the data.

Another advantage of automatic alignment is that it simplifies the annotation of new data. For instance, Sakaguchi et al. (2016) claimed that forcing annotators to annotate within the confines of an error scheme often led to unnatural sounding sentences and so they instead advocated freer annotation. With automatic alignment, we can hence allow annotators to freely edit the text yet still extract the edits afterwards.

Finally, another application of automatic alignment is to extract the edits from system output. Since it is unlikely that researchers will have the resources to manually annotate the output after every iteration of system development, it is much more efficient to do this automatically instead. This information can then be used to determine how system edits differ from human edits.

4.1 Previous approaches

Although there is very little previous work on automatic sentence alignment in GEC, one of the first attempts was made by Swanson and Yamangil (2012), who built a system to align sentences and classify the extracted edits for the purposes of EFL feedback. In particular, they used the Levenshtein algorithm to align the sentences and then classified the edits according to the CLC error scheme using a maximum entropy classifier.

One limitation of the approach however, as noted by the authors themselves, is that grammatical edits sometimes involve multiple tokens; for example reordering errors (e.g. [*only can* → *can only*]) or phrasal verb errors (e.g. [*look at* → *watch*]) necessarily involve more than one token on at least one side of the edit. The Levenshtein algorithm, however, only aligns individual tokens and so some alignments must be merged in order to obtain multi-token edits. Swanson and Yamangil hence experimented with some basic merging strategies and found that simply merging all adjacent non-match alignments more closely approximated human alignments.

Building on this foundation, Xue and Hwa (2014) analysed Swanson and Yamangil's output and found that approximately 70% of all errors in their error type classifier were caused by bad alignments (merged or otherwise). In order to improve on the simple *all-merge* alignment strategy, they hence trained a binary maximum entropy classifier to predict whether edits should be merged or not. They tested this merging classifier on several datasets, including NUCLE and the FCE, and ultimately reported improvements of between 5-10% for both alignment and classification.

This	**wide**	**spread**	**propaganda**	benefits	**only**	**to**	**the**	**companys**	.
This	**widespread**	**publicity**	**only**	benefits	**their**	**companies**			.

Table 4.2: A linguistically unintuitive Levenshtein alignment.

Despite these improvements however, there is still a considerable performance margin between automatic and human edit annotations. Furthermore, Xue and Hwa's approach also requires training on existing annotations, which may be inconsistent across corpora. A better edit extraction algorithm should hence be independent of any training corpus.

4.2 Automatic alignment

The most common way to align sequences of text in NLP is to use the Levenshtein algorithm, which calculates the cheapest way to transform one string into another by means of insertions, deletions and substitutions. As Levenshtein does not take linguistic information into account however, it often produces alignments that are sub-optimal in terms of human intuition, even if they are optimal in terms of cost (Table 4.2). We hence propose to incorporate linguistic information into the algorithm.

4.2.1 Damerau-Levenshtein

First however, as noted by Xue and Hwa, another limitation of Levenshtein is that it is unable to handle word order errors. For example, [*only can* $\rightarrow$ *can only*] is realised as [*only* $\rightarrow \varepsilon$], [*can* $\rightarrow$ *can*] and [$\varepsilon \rightarrow$ *only*]; i.e. reorderings are treated as deletions followed by insertions of identical tokens. Since we want to preserve word order errors however, we hence propose using Damerau-Levenshtein (Damerau, 1964) rather than Levenshtein, because Damerau-Levenshtein also takes token transpositions into account.

In particular, Damerau-Levenshtein factors the transposition of two adjacent items into the alignment cost, and is hence more appropriate for our task because most word order errors only involve two tokens.[2] That said, we should not ignore the longer word order errors, because the algorithm will otherwise split them into smaller, less meaningful edits that only contribute to the overall system error rate.

To solve this problem, we hence extend the Damerau-Levenshtein distance to allow for transpositions of arbitrary length, as shown in Figure 4.1. This is achieved by traversing a diagonal back from the current cell in the cost matrix and looking for a source sequence that matches the target sequence in any order. More specifically, if source token src_i does not match target token tgt_j at any given point in the matrix, and the previous source token src_{i-1} similarly does not match the previous target token tgt_{j-1}, we next check whether the sorted lower-cased

[2]Specifically, approximately 50-60% of all word order errors involve two tokens in NUCLE and FCE train.

```
function DL_distance_extended(a, b):
  declare d[0..length(a), 0..length(b)]
  for i := 0 to length(a) inclusive do
    d[i, 0] := i
  for j := 0 to length(b) inclusive do
    d[0, j] := j

  for i := 1 to length(a) inclusive do
    for j := 1 to length(b) inclusive do
      if a[i] = b[j] then
        d[i, j] := 0
      else
        d[i, j] := min(d[i-1, j  ] + del_cost(a[i]),
                       d[i  , j-1] + ins_cost(b[j]),
                       d[i-1, j-1] + sub_cost(a[i], b[j]))

      // Damerau-Levenshtein extension for multi-token transpositions
      k = 1
      while i > 1 and j > 1 and (i - k) >= 1 and (j - k) >= 1 and
            d[i-k, j-k] - d[i-k-1, j-k-1] > 0 do
        if sorted(lowercase(a[i-k:i+1])) = sorted(lowercase(b[j-k:j+1])) then
          d[i, j] := min(d[i, j], d[i-k, j-k] + trans_cost(a[i-k:i+1], b[j-k:j+1])
          break
        k += 1

  return d[length(a), length(b)]
```

Figure 4.1: Damerau-Levenshtein distance allowing for transpositions of arbitrary length.

strings of all these tokens, (src_i, src_{i-1}) and (tgt_j, tgt_{j-1}), are the same. If they are, we iteratively increase the size of the sequences $(src_i, src_{i-1}, ..., src_{i-k})$ and $(tgt_j, tgt_{j-1}, ..., tgt_{j-k})$, provided tokens src_{i-k} and tgt_{j-k} still do not match, until either the extended sequences do not match, or at least one of the indices $i-k$ or $j-k$ reaches the first token in the source or target sentence respectively. It is important to make sure tokens src_{i-k} and tgt_{j-k} never match to prevent matched tokens from being included in the transposition. The cost of a matched transposition sequence of length n is finally defined as $n-1$, which is compatible with the original definition.

4.2.2 Linguistic alignment

In an effort to incorporate linguistic information into the alignment, we replaced the substitution cost in Damerau-Levenshtein with the function shown in Figure 4.2. In this function, we set the cost to 0 if the original and corrected tokens differ only in case (e.g. [*the* → *The*]), otherwise, the substitution cost is the sum of sub-costs for lemma, part of speech and character differences. Each of these sub-costs is defined as follows:

```
function substitution(a, b):
   if lowercase(a) = lowercase(b) then
      return 0
   else
      return lemma_cost(a, b) + pos_cost(a, b) + char_cost(a, b)
```

Figure 4.2: Our linguistically motivated token substitution function.

	This	wide	spread	propaganda	benefits	only	to	the	companys	.
(a)	This	widespread	publicity	only	benefits	their	companies			.
(b)	This	widespread		publicity	only	benefits		their	companies	.

Table 4.3: The difference between (a) standard Levenshtein and (b) linguistically-enhanced Damerau-Levenshtein in relation to the original sentence.

lemma cost: 0 if tokens share the same lemma or derivationally related form (e.g. 'met' and 'meeting'), otherwise 0.499.

part-of-speech cost: 0 if tokens share the same part of speech, otherwise 0.25 if both tokens are content words (adjectives, adverbs, nouns or verbs) and 0.5 in all other cases.

character cost: the proportion of character mismatches between 0 and 1, computed as the character-level Damerau-Levenshtein distance between the tokens divided by the length of their alignment.

To increase the likelihood of aligning derivationally related forms, we lemmatise each token several times as if it were an adjective, adverb, noun and verb. We do this because if we only lemmatise for a single part of speech, then we might overlook certain derivationally related words. For example, while the lemma of the verb 'met' is 'meet', the lemma of the noun 'meeting' is 'meeting', which suggests these words are not related. By also lemmatising 'meeting' as a verb however, we find that the two tokens do share a common lemma, 'meet', which instead correctly suggests they are related and should align. Ultimately, we consider two tokens to be derivationally related if their respective sets of lemmas intersect.

The sub-costs are also set in such a way that the overall substitution function always yields values in the $[0, 2)$ range. Keeping the cost asymptotic to 2 is important to enforce a preference for substitutions over insertions and deletions (both cost = 1); this is why we use a lemma cost of 0.499 instead of 0.5. We tried different combinations of these costs, provided they met this condition, but did not find any significant differences in the results.

By incorporating all this additional linguistic information into the cost, we improve the likelihood that tokens with similar etymology, spelling and function align. We hence argue this approach is not only more robust than the simple surface matching alignment calculated by Levenshtein, but also produces more natural, human-like alignments (Table 4.3). The final alignment is retrieved by collecting the operations that make up the optimal path in the cost matrix. Given that the cost is now dependent upon a variable function, it is often the case that there is just a single optimal alignment.

	Sents	Edits	Min. Edits
FCE train	30,200	44,257	42,657
FCE dev	2,371	3,511	3,383
FCE test	2,805	4,748	4,485
NUCLE	59,586	43,854	42,671
CoNLL-2013	1,390	3,411	3,371
CoNLL-2014.0	1,341	2,395	2,382
CoNLL-2014.1	1,341	3,318	3,273

Table 4.4: The number of sentences, edits and minimised edits in various datasets.

Alignment	*Lev*	*Lev*	*DL*
Reference	*Gold*	*Gold-min*	*Gold-min*
FCE train	60.10	64.36	72.22
FCE dev	60.75	64.55	73.36
FCE test	58.25	63.28	73.10
NUCLE	37.26	51.18	54.77
CoNLL-2013	49.03	62.42	70.82
CoNLL-2014.0	50.91	60.29	66.93
CoNLL-2014.1	48.36	63.08	69.49

Table 4.5: Table showing how minimised references edits (Gold-Min) and our linguistically enriched Damerau-Levenshtein algorithm (DL) perform against standard Levenshtein (Lev) and unmodified references (Gold). All scores are F_1.

4.2.3 Alignment experiments

We evaluated our improved alignment algorithm on NUCLE, CoNLL and the public FCE. All datasets were preprocessed and tokenized using the method described in Chapter 3. Tokens were then POS tagged and lemmatised by spaCy v1.9.0. Table 4.4 provides some additional information about the properties of the processed datasets.

Since another aim of this work is to standardise edit annotations, we also minimised the gold edit spans by recursively removing unchanged words from both sides. For example, [*has eating* $\rightarrow$ *was eating*] was reduced to [*has* $\rightarrow$ *was*] and [*look at* $\rightarrow$ *look for*] was reduced to [*at* $\rightarrow$ *for*]. If we did not do this, our automatic edits would never match these human edits because automatic edits can only ever consist of unmatched tokens. This step also removed uncorrected edits from the references and hence reduced the total number of edits in each dataset (Table 4.4).

In order to quantify the effect of edit minimisation, our first experiment simply compared the performance of standard Levenshtein against the unmodified and minimised references. Table 4.5 hence shows that even when the alignment algorithm remains the same, minimised references can have a significant effect on the results, with scores increasing by 4 to 15 F_1 on all datasets. Although the large gains in NUCLE and CoNLL can also be attributed to the fact that their annotation guidelines necessarily required missing word edits to include an unchanged word in their correction (Section 3.2), minimised gold edit spans nevertheless show improvements in the

M	S	D	S	T		D	S	S	M
This	wide	spread	propaganda	benefits	only	to	the	companys	.
This	widespread		publicity	only	benefits		their	companies	.

Table 4.6: Individual operations obtained from automatic alignment: (M)atch, (I)nsertion, (D)eletion, (S)ubstitution and (T)ransposition.

	FCE train		NUCLE	
Edit Size	Cum. Freq	%	Cum. Freq.	%
1 : 1	22,110	51.83%	17,633	41.32%
0 : 1	30,375	71.21%	24,905	58.37%
1 : 0	34,946	81.92%	30,744	72.05%
0-2 : 0-2	40,291	94.45%	38,116	89.33%
0-3+ : 0-3+	42,657	100.00%	42,671	100.00%

Table 4.7: Distribution of edits in minimised gold FCE train and NUCLE in terms of how many tokens are on either side of the edit.

FCE as well. These results hence highlight the importance of edit standardisation in GEC and so we only use minimised references in all subsequent experiments.

In our second experiment, we compared Levenshtein against our own linguistically enhanced Damerau-Levenshtein alignment (Table 4.5). Scores again increased by an average of 7 F_1 across all datasets, with several datasets achieving greater than 70 F_1. These scores are particularly significant given we do not yet merge any edits and so currently fail to match all multi-token edits. This also might explain why the results for NUCLE are significantly lower than the other datasets; we hypothesise the annotators had a preference for longer edits.

4.3 Alignment merging

One limitation of Damerau-Levenshtein is that other than for transpositions, alignments maximally involve only a single token on either side; i.e. insertions are always 0:1, deletions are always 1:0 and substitutions are always 1:1. While this is sufficient in most cases, e.g. [*propaganda* → *publicity*] and [*companys* → *companies*], it is not in others, e.g. [*wide spread* → *widespread*] is split into [*wide* → *widespread*] and [*spread* → ε] (Table 4.6).

Table 4.7 hence shows that 20-30% of all edits in NUCLE and FCE train contain more than one token on at least one side of the edit, and that none of these are currently captured by our approach. This table also shows that NUCLE contains almost 10% more multi-token edits than FCE train, which confirms our theory as to why it performed so poorly in the previous section. Regardless, multi-token edits are still an important class of learner errors that we should attempt to handle.

4.3.1 Merging rules

In order to capture multi-token edits and improve performance, we hence implemented a recursive rule-based merging function. Specifically, we analysed the relationship between human edits and automatic edits and found that the most common multi-token errors involve, for example, phrasal verbs [*look at* → *watch*], possessive nouns [*friends* → *friend 's*], and orthographic changes [*wide spread* → *widespread*]. We thus wrote rules to merge or split alignments based on these observed patterns.

The complete list of rules and their priority is as follows:[3]

1. Split a sequence into subsequences of matches (M) and non-matches and only process the non-matches; e.g. MDDSMMTMSI becomes DDS, T and SI.
2. Merge any adjacent operations that consist of punctuation followed by a case change; e.g. [, → .] + [*we* → *We*] = [, *we* → . *We*].
3. Split transpositions (T) from all other sequences of non-matches; e.g. IITSS becomes II, T and SS.
4. Merge any adjacent operations that consist of a possessive suffix preceded by anything else; e.g. [*friends* → *friend*] + [ε → *'s*] = [*friends* → *friend 's*].
5. Merge any adjacent operations that differ only in terms of whitespace; e.g. [*sub* → *subway*] + [*way* → ε] = [*sub way* → *subway*].
6. Split substitutions (S) that share $> 70\%$ of the same characters, e.g. [*writting* → *writing*], unless they have the same POS as the previous alignment. This exception prevents edits such as [*eated* → *have eaten*] being split into [ε → *have*] + [*eated* → *eaten*].
7. Split substitutions that are preceded by other substitutions; e.g. DDSSII becomes DDS, S and II.
8. Merge any consecutive operations that involve at least one content word; e.g. [*On* → *In*] + [*the* → ε] + [*other* → ε] + [*hand* → *addition*] = [*On the other hand* → *In addition*].
9. Merge any consecutive operations that involve tokens with the same POS; e.g. [*(look) at* → *(look) up*] + [ε → *to*] = [*(look) at* → *(look) up to*].
10. Split any determiner edits at the end of a sequence; e.g. [*saw* → *seen the*] becomes [*saw* → *seen*] + [ε → *the*].

Each sequence of alignment operations between two sentences (e.g. MSDSTDSSM) is processed recursively using the above rules in a top-down fashion. Rules are applied in order, with priority relative to their position in the list. Every time an edit is returned by one of the rules, we process the remaining sub-sequences individually until they are exhausted or no more rules can be applied (see Figure 4.3). It should be noted that rules that iteratively grow the merge range

[3]Note that the latest version of the rule logic is slightly different from that which is reported here and in Felice et al. (2016). See `https://github.com/chrisjbryant/errant/blob/master/changelog.md`.

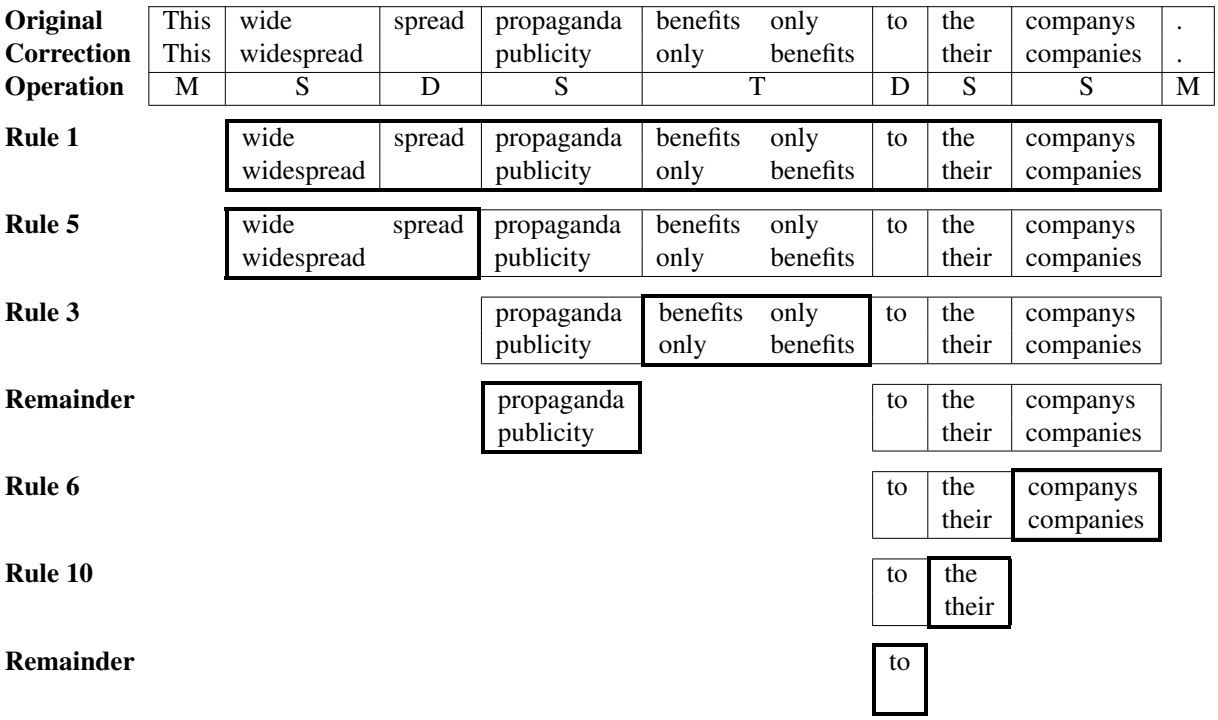

Figure 4.3: A step-by-step edit extraction example.

of the alignment (e.g. #8) can be overridden by others with higher priority (e.g. #4), causing the remaining operations in the truncated subsequence to be reprocessed from scratch.

4.3.2 Merging experiments

We evaluated our rule-based merging method on the same datasets as before, and contrasted it against the following alternative merging strategies:

all-split: All consecutive non-matches are split: DDSI $\rightarrow$ D, D, S, I.

all-merge: All consecutive non-matches are merged: DDSI $\rightarrow$ DDSI.

all-equal: All consecutive same operation non-matches are merged: DDSI $\rightarrow$ DD, S, I.

The results (Table 4.8) showed that while the *all-split* strategy tends to have a high number of true positives (TPs) and low number of false negatives (FNs), it also has the highest number of false positives (FPs). In contrast, *all-merge* tends to have a low number of both TPs and FPs, but the highest number of FNs. *All-equal*, meanwhile, falls somewhere between the two extremes. These results hence show that different merging strategies have different strengths and weaknesses.

Since we designed our rule-based merging function to be more sophisticated than the others however, we were pleased to see improvements of between 4 and 12 F_1 over the second best approach, *all-merge*, on all but one dataset. Specifically, *all-merge* on NUCLE outperformed our rules by 3 F_1. The most likely reason for this however, is again because NUCLE annotators

		Edit Extraction						Extraction + Classification					
Dataset	**Method**	**TP**	**FP**	**FN**	**P**	**R**	**F_1**	**TP**	**FP**	**FN**	**P**	**R**	**F_1**
FCE train	All-split	34,448	18,291	8,209	65.32	80.76	72.22	-	-	-	-	-	-
	All-merge	30,494	7,454	12,163	80.36	71.49	75.66	-	-	-	-	-	-
	All-equal	32,153	13,609	10,504	70.26	75.38	72.73	-	-	-	-	-	-
	Rules	36,308	7,598	6,349	82.69	85.12	**83.89**	-	-	-	-	-	-
FCE dev	All-split	2,766	1,392	617	66.52	81.76	73.36	2396	1762	987	57.62	70.82	63.55
	All-merge	2,345	652	1,038	78.24	69.32	73.51	1997	1000	1386	66.63	59.03	62.60
	All-equal	2,544	1,080	839	70.20	75.20	72.61	2225	1399	1158	61.40	65.77	63.51
	Rules	2,883	600	500	82.77	85.22	**83.98**	2469	1014	914	70.89	72.98	**71.92**
FCE test	All-split	3,663	1,874	822	66.15	81.67	73.10	3095	2442	1390	55.90	69.01	61.76
	All-merge	3,147	764	1,338	80.47	70.17	74.96	2597	1314	1888	66.40	57.90	61.86
	All-equal	3,378	1,411	1,107	70.54	75.32	72.85	2867	1922	1618	59.87	63.92	61.83
	Rules	3,850	740	635	83.88	85.84	**84.85**	3207	1383	1278	69.87	71.51	**70.68**
NUCLE	All-split	30,155	37,282	12,516	44.72	70.67	54.77	-	-	-	-	-	-
	All-merge	33,007	8,006	9,664	80.48	77.35	**78.88**	-	-	-	-	-	-
	All-equal	31,115	20,414	11,556	60.38	72.92	66.06	-	-	-	-	-	-
	Rules	34,407	13,872	8,264	71.27	80.63	75.66	-	-	-	-	-	-
CoNLL-2013	All-split	2,709	1,570	662	63.31	80.36	70.82	2072	2207	1299	48.42	61.47	54.17
	All-merge	2,199	642	1,172	77.40	65.23	70.80	1629	1212	1742	57.34	48.32	52.45
	All-equal	2,414	1,145	957	67.83	71.61	69.67	1836	1723	1535	51.59	54.46	52.99
	Rules	2,789	571	582	83.01	82.74	**82.87**	2077	1283	1294	61.82	61.61	**61.71**
CoNLL-2014.0	All-split	1,863	1,322	519	58.49	78.21	66.93	1274	1911	1108	40.00	53.48	45.77
	All-merge	1,690	401	692	80.82	70.95	75.56	1095	996	1287	52.37	45.97	48.96
	All-equal	1,726	874	656	66.38	72.46	69.29	1150	1450	1232	44.23	48.28	46.17
	Rules	1,922	534	460	78.26	80.69	**79.45**	1274	1182	1108	51.87	53.48	**52.67**
CoNLL-2014.1	All-split	2,636	1,678	637	61.10	80.54	69.49	2046	2268	1227	47.43	62.51	53.93
	All-merge	2,429	551	844	81.51	74.21	77.69	1779	1201	1494	59.70	54.35	56.90
	All-equal	2,447	1,170	826	67.65	74.76	71.03	1890	1727	1383	52.25	57.75	54.86
	Rules	2,846	599	427	82.61	86.95	**84.73**	2122	1323	1151	61.60	64.83	**63.17**

Table 4.8: Performance of different merging methods on the edit extraction and full error classification task. TP: true positives, FP: false positives, FN: false negatives, P: precision, R: recall.

seemed to have a higher preference for longer edits than the other datasets; this also explains why *all-split* performed the worst on NUCLE.

In addition to evaluating edit extraction alone, we also retrained Xue and Hwa's (2014) publicly available implementation of their maximum entropy error type classifier[4] to replicate results for an end-to-end classification system. Specifically, we trained one classifier to predict FCE error types based on FCE train, and another to predict NUCLE error types based on NUCLE. These classifiers were then evaluated on FCE dev and FCE test, and CoNLL-2013 and CoNLL-2014 respectively.

Table 4.8 hence also reports results for error type classification given different edit extraction and merging strategies. The scores are consistent with edit extraction alone, but are otherwise expectedly lower by an average of 19 F_1 because the full classification task is more complex than the edit extraction task. That said, classification using our rule-based merging method was still best overall, with scores improving by an average of 7 F_1 compared to the next best merging method.

One interesting result is that although the FCE and CoNLL datasets score similarly in terms of edit extraction alone, there is a large difference between their classification scores. In particular,

[4] `https://github.com/xuehuichao/correction_detector`

		Edit Extraction			Extraction + Classification		
Dataset	**Method**	**P**	**R**	**F_1**	**P**	**R**	**F_1**
FCE dev	S&Y	77.85	67.93	72.55	66.06	57.64	61.56
	X&H	78.65	79.93	79.28	67.22	68.31	67.76
	Our work	82.77	85.22	**83.98**	70.89	72.98	**71.92**
FCE test	S&Y	80.02	68.76	73.97	65.70	56.45	60.73
	X&H	78.21	79.87	79.03	65.17	66.56	65.86
	Our work	83.88	85.84	**84.85**	69.87	71.51	**70.68**
CoNLL-2013	S&Y	78.24	64.52	70.72	58.42	48.18	52.80
	X&H	78.25	70.22	74.02	58.74	52.71	55.57
	Our work	83.01	82.74	**82.87**	61.82	61.61	**61.71**
CoNLL-2014.0	S&Y	80.56	68.89	74.27	52.53	44.92	48.43
	X&H	77.82	72.04	74.82	51.88	48.03	49.88
	Our work	78.26	80.69	**79.45**	51.87	53.48	**52.67**
CoNLL-2014.1	S&Y	80.76	72.32	76.31	59.09	52.92	55.83
	X&H	81.16	76.84	78.94	60.21	57.01	58.57
	Our work	82.61	86.95	**84.73**	61.60	64.83	**63.17**

Table 4.9: Performance of our proposal vs. previous methods in an end-to-end edit extraction and classification task. S&Y used Levenshtein with an *all-merge* strategy (Swanson and Yamangil, 2012) while X&H used Levenshtein with a maximum entropy merging classifier (Xue and Hwa, 2014).

the FCE scores an average of 10 F_1 better than CoNLL. This is especially surprising given we would expect the FCE error type classifier to perform worse than the NUCLE classifier because it has a much larger confusion set (77 vs. 28 categories). Since both classifiers were trained on an almost identical number of edits however, other possible explanations for this effect include either that the classifier feature set is more highly optimised for the FCE rather than NUCLE, or else NUCLE annotations are more inconsistent than FCE annotations.

4.4 System comparison

To further validate the efficacy of our approach, we also compared it against previous work. Specifically, we reimplemented Swanson and Yamangil's Levenshtein + *all-merge* strategy, and Xue and Hwa's Levenshtein + maximum entropy merging classifier strategy; in the latter case, the merging classifier was trained on NUCLE and FCE train respectively.

Table 4.9 hence shows that our approach achieves state-of-the-art performance on all tasks and datasets, improving upon X&H by an average of 6 F_1 for edit extraction alone and an average of 4.5 F_1 in the full extraction and classification task. In most cases, this improvement is also higher in terms of both precision and recall.

		Single-Token Edits			Multi-Token Edits		
Dataset	**Method**	**P**	**R**	**F_1**	**P**	**R**	**F_1**
FCE dev	S&Y	92.84	70.34	80.04	39.45	56.28	46.38
	X&H	83.13	84.62	83.87	56.83	57.31	57.07
	Our work	87.16	90.86	**88.97**	59.96	58.00	**58.97**
FCE test	S&Y	95.29	70.18	80.83	43.44	62.17	51.14
	X&H	82.01	84.48	83.23	59.59	58.39	58.98
	Our work	88.09	91.12	**89.58**	63.04	61.29	**62.15**
CoNLL-2013	S&Y	95.83	65.89	78.09	39.17	57.98	46.75
	X&H	90.59	73.21	80.98	42.23	55.92	48.12
	Our work	89.22	87.84	**88.52**	55.28	58.32	**56.76**
CoNLL-2014.0	S&Y	96.01	70.10	81.04	46.46	63.85	**53.78**
	X&H	89.39	75.94	82.12	44.95	55.84	49.81
	Our work	83.57	87.66	**85.56**	54.07	51.73	52.88
CoNLL-2014.1	S&Y	94.02	74.14	82.91	47.68	64.52	54.83
	X&H	90.59	80.21	85.09	51.60	62.42	56.50
	Our work	86.54	93.55	**89.91**	63.08	58.71	**60.82**

Table 4.10: Performance of our proposal vs. previous methods in terms of single and multi-token edits. S&Y used Levenshtein with an *all-merge* strategy (Swanson and Yamangil, 2012) while X&H used Levenshtein with a maximum entropy merging classifier (Xue and Hwa, 2014).

4.4.1 Single vs. multi-token evaluation

We additionally carried out a more detailed evaluation of all approaches in terms of single and multi-token edits (Table 4.10). Here, we define a multi-token edit as any edit that has more than two tokens on either the original or corrected side of the edit.

The results show that while our method tends to have a lower precision in the single-token setting, it makes up for this by always having a much higher recall. In contrast, our method always has the highest precision in the multi-token setting, but recall is variable. In particular, although our method has the highest recall in FCE dev and CoNLL-2013, it has the worst recall in both CoNLL-2014 datasets.

Ultimately, our method still performs the best in almost all settings. The only exception is CoNLL-2014.0 where Swanson and Yamangil's Levenshtein + *all-merge* strategy performed better than our own method by less than 1 F_1. Since they achieved this result by means of a much higher recall however, we can conclude that this annotator simply had a much higher preference for multi-token edits than the other annotators.

4.5 Discussion

It is worth stating that many of our reported results are actually an underestimate of true performance. This is because despite gold reference minimisation, there is a high degree of variability in the way humans annotate GEC data. For instance, as shown by Bryant and

Ng (2015), human annotators often have very different perceptions of grammaticality, and it is linguistically plausible that, for example, [*has eaten* → *was eating*] is annotated either as one edit (as above) or two edits ([*has* → *was*] + [*eaten* → *eating*]) by different, or even the same, annotators. This means the *all-split* merging strategy will never match the former while the *all-merge* merging strategy will never match the latter, even though the annotations fundamentally equate to the same thing. Due to this inconsistency, system performance will thus be underestimated regardless of which merge strategy you choose.

In contrast, we consider merge consistency a strength of our rule-based approach. Even if our alignment does not agree with the gold standard, at least the decision to merge or split is consistent across all similar cases. In this way, a desirable property of our approach is that it can be used to standardise ambiguous annotations where splits or merges are equally plausible.

In addition to a quantitative analysis, we also carried out an informal qualitative analysis of the errors made by our system. In particular, we found one source of errors concerns tokens that are affected by more than one mistake; e.g. [*wide spraed* → *widespread*]. While our system includes a rule to merge adjacent alignments where the only difference is white space, this rule does not activate in the above case because one of the tokens also contains a misspelling. This consequently means the alignments are not merged and do not match the gold standard; such cases are difficult to handle.

A related issue is that reference minimisation is unable to handle cases where unchanged tokens occur in the middle of a human edit; e.g. [*can easily been* → *could easily be*]. As an automatic alignment will always consider *easily* a matched token, the remaining non-matches [*can* → *could*] and [*been* → *be*] become isolated and are never merged. Although this might be construed as a limitation of the alignment algorithm, I would instead argue that the algorithm is actually more informative than the human reference in this case because the gold references should generally not contain unchanged words.

Finally, another strength of a rule-based approach is that it is easier to diagnose which rules are responsible for producing a given output sequence. This is in contrast with machine learning techniques where it is often much more difficult to determine why certain edits were merged in a certain way. Considering only about 30% of all edits (at most) in any dataset require merging anyway, a rule-based approach seemed a more cost-effective solution.

4.6 Summary

In this chapter, I presented a method to automatically align parallel original and corrected sentence pairs and extract the edits in a linguistically informed manner. Since up to 30% of all edits contain more than one token on either side of the edit, I also implemented a rule-based merging strategy to capture multi-token edits. This approach outperformed all other alignment and merging strategies and consistently achieved the highest F_1 scores in almost all experiments.

It also outperformed the previous best approach by 6 F_1 in terms of edit extraction and 4.5 F_1 in terms of edit extraction and classification (Xue and Hwa, 2014), and can hence be considered the state of the art.

CHAPTER 5

AUTOMATIC ERROR TYPING

Now that we can automatically align parallel sentences and extract the edits, the next step is to classify them according to an error type framework. Specifically, while CLC edits are classified according to the CLC error type framework and NUCLE edits are classified according to the NUCLE error type framework, our automatically aligned Lang-8 and JFLEG edits are currently unclassified.

Although one solution to this problem might be to train an error type classifier to predict types, as we did in the previous section, we must first decide which framework to train it on. For example, should we train on CLC error types or NUCLE error types? Or should we consider other error type frameworks; e.g. Izumi et al. (2005); Darus and Subramaniam (2009); Chan (2010); Maharjan (2010); Giri (2011); Nagata et al. (2011)? Unfortunately, there is no definitive answer to this question, given that most frameworks are largely arbitrary in design, and so it is very difficult to quantify their effectiveness.

One reason against using an existing framework however, is that it introduces a bias towards the source dataset. For example, a CLC classifier is likely to perform better on CLC data than NUCLE data, while a NUCLE classifier is likely to perform better on NUCLE data than CLC data. Xue and Hwa (2014) actually tested this hypothesis and reported drops of up to 10% F_1 when training and testing on different corpora. This is undesirable if we want to standardise datasets, and so we instead prefer a more dataset-agnostic approach.

This chapter hence introduces our new rule-based classifier which was designed to be a dataset-agnostic compromise between both NUCLE and the CLC. Combined with the automatic alignment method outlined in the previous chapter, the whole program was called the ERRor ANnotation Toolkit (ERRANT)[1] and was publicly released as part of this thesis (Bryant et al., 2017).

[1] `https://github.com/chrisjbryant/errant`

5.1 A rule-based error type framework

The intuition behind our rule-based classifier comes from the observation that a) most error types are based on POS tags, and b) all errors involve either a missing, unnecessary or replacement word; a minimal framework could hence simply combine these two properties. Such a framework would also be very easy to implement because POS tags can be obtained from a POS tagger, and the edit operation can be inferred from whether one side of the edit is empty or not.

While this approach goes a long way towards reliably classifying most edits, not all edits are well-characterised by POS tags alone, e.g. spelling or word order errors. We thus extended this framework with additional rules and error types using other automatically obtained properties of the data.

Ultimately, the final framework contains 25 main error types and relies on roughly 50 rules. All these error types are described in Table 5.1. Note that many of them can be prefixed by 'M:', 'R:' or 'U:', depending on whether they describe a Missing, Replacement, or Unnecessary edit, and that the full list of all 55 valid combinations is shown in Table 5.2. Like the CLC, the typology was designed to be modular to facilitate extracting specific error types at different levels of granularity.

Finally, one caveat of error scheme design is that it is always possible to add new categories; for example, we currently label [*could* → *should*] a tense error, when it might otherwise be considered a modal error. The reason we do not call it a modal error, however, is because it would then become less clear how to handle other cases such as [*can* → *should*] and [*has eaten* → *should eat*], which might otherwise be considered more complex combinations of modal and tense errors. As it is impractical to create new categories and rules to differentiate between increasingly narrow distinctions however, our final framework aims to be a compromise between informativeness and practicality.

This concept was also noted by Nagata et al. (2011):

> "No matter how well an annotation scheme is designed, there will always be exceptions. Every time an exception appears, it becomes necessary to revise the annotation scheme. Another issue we have to remember is that there is a trade-off between the granularity of an annotation scheme and the level of the difficulty in error annotation. The more detailed an annotation scheme is, the more information it can contain and the more difficult identifying errors is, and vice versa."

5.2 Framework prerequisites

Since we deliberately designed our framework to be dataset-agnostic, the only prerequisites for our classifier are that each token in both the original and corrected sentence is POS tagged, lemmatised, stemmed and dependency parsed. We again use spaCy v1.9.0 for all but the

Code	Meaning	Description / Example
ADJ	Adjective	*big → wide*
ADJ:FORM	Adjective Form	Comparative or superlative adjective errors. *biggerest → biggest, bigger → biggest*
ADV	Adverb	*speedily → quickly*
CONJ	Conjunction	*and → but*
CONTR	Contraction	*n't → not*
DET	Determiner	*the → a*
MORPH	Morphology	Same lemma but nothing else in common. *quick (adj) → quickly (adv)*
NOUN	Noun	*house → building*
NOUN:INFL	Noun Inflection	Count-mass noun errors. *informations → information*
NOUN:NUM	Noun Number	*cat → cats*
NOUN:POSS	Noun Possessive	*friends → friend's*
ORTH	Orthography	Case and/or whitespace errors. *Bestfriend → best friend*
OTHER	Other	Unclassified errors; e.g. paraphrases *at his best → well, job → professional*
PART	Particle	*(look) in → (look) at*
PREP	Preposition	*of → at*
PRON	Pronoun	*ours → ourselves*
PUNCT	Punctuation	*! → .*
SPELL	Spelling	*recieve → receive, color → colour*
UNK	Unknown	Detected but not corrected errors
VERB	Verb	*ambulate → walk*
VERB:FORM	Verb Form	Infinitives, gerunds and participles. *to eat → eating, dancing → danced*
VERB:INFL	Verb Inflection	Misapplication of tense morphology. *getted → got, fliped → flipped*
VERB:SVA	Subject-Verb Agreement	*(He) have → (He) has*
VERB:TENSE	Verb Tense	Inflectional, periphrastic, modals and passives. *eats → ate, eats → has eaten, eats → can eat*
WO	Word Order	*only can → can only*

Table 5.1: The list of 25 main error categories in our new framework with examples and explanations.

stemming, which is performed by the Lancaster Stemmer in NLTK (spaCy does not contain a stemmer). Since fine-grained POS tags are often too detailed for the purposes of error type classification, we also map spaCy's Penn Treebank style tags to the coarser set of Universal Dependency tags (Nivre et al., 2016).[2] We use the latest Hunspell GB-large word list[3] to help classify non-word errors. The marked-up tokens in an edit span are then input to the classifier.

[2] `http://universaldependencies.org/tagset-conversion/en-penn-uposf.html`
[3] `https://sourceforge.net/projects/wordlist/files/speller/2018.04.16/`

		Operation Tier		
	Type	**Missing**	**Unnecessary**	**Replacement**
Part Of Speech Tier	Adjective	M:ADJ	U:ADJ	R:ADJ
	Adverb	M:ADV	U:ADV	R:ADV
	Conjunction	M:CONJ	U:CONJ	R:CONJ
	Determiner	M:DET	U:DET	R:DET
	Noun	M:NOUN	U:NOUN	R:NOUN
	Particle	M:PART	U:PART	R:PART
	Preposition	M:PREP	U:PREP	R:PREP
	Pronoun	M:PRON	U:PRON	R:PRON
	Punctuation	M:PUNCT	U:PUNCT	R:PUNCT
	Verb	M:VERB	U:VERB	R:VERB
Token Tier	Contraction	M:CONTR	U:CONTR	R:CONTR
	Morphology	-	-	R:MORPH
	Orthography	-	-	R:ORTH
	Other	M:OTHER	U:OTHER	R:OTHER
	Spelling	-	-	R:SPELL
	Word Order	-	-	R:WO
Morphology Tier	Adjective Form	-	-	R:ADJ:FORM
	Noun Inflection	-	-	R:NOUN:INFL
	Noun Number	-	-	R:NOUN:NUM
	Noun Possessive	M:NOUN:POSS	U:NOUN:POSS	R:NOUN:POSS
	Verb Form	M:VERB:FORM	U:VERB:FORM	R:VERB:FORM
	Verb Inflection	-	-	R:VERB:INFL
	Verb Agreement	-	-	R:VERB:SVA
	Verb Tense	M:VERB:TENSE	U:VERB:TENSE	R:VERB:TENSE

Table 5.2: There are 55 total possible error types. This table shows all of them except UNK, which indicates an uncorrected error. A dash indicates an impossible combination.

5.3 Error type definitions

This section provides a detailed description of all the error types in our new framework and lists the rules that capture them. Note that the error types are introduced alphabetically in terms of granularity (coarse to fine) and that the rules are ordered differently in the actual implementation. See Section 5.3.5 for more details on rule order.

5.3.1 Operation tier

All edits are minimally classified in terms of edit operation; i.e. whether tokens are missing (M), replaced (R) or unnecessary (U) (Table 5.3).

A special case concerns edits such as [*Man* → *The man*] or [*The man* → *Man*], which ostensibly look like replacement edits, but actually denote missing or unnecessary words. We hence treat them as such and ignore the orthographic case change. They are detected by the following rule:

Type	Form
Missing	$[\varepsilon \rightarrow B]$
Replacement	$[A \rightarrow B]$
Unnecessary	$[A \rightarrow \varepsilon]$

Table 5.3: The forms of edits in terms of operation.

1. The number of tokens on each side of the edit is not equal, *and*
2. The lower cased form of the last token is the same on both sides, *and*
3. Removing the last token on both sides results in an empty string on one side.

Finally, any gold edit of the form $[A \rightarrow A]$ or $[\varepsilon \rightarrow \varepsilon]$ is labelled Unknown (UNK), since it ultimately has no effect on the text. These are normally gold edits that humans detected, but were unable or unsure how to correct. UNK edits are analogous to Unclear Meaning (Um) edits in NUCLE.

5.3.2 Part-of-speech tier

POS-based error types are generally assigned based on the Universal Dependency POS tags of the tokens in the edit. Since some Universal tags are not as informative as others however, we ignore the following tags: interjections (INTJ), numerals (NUM), symbols (SYM) and other (X). We also renamed adpositions (ADP) to prepositions (PREP) and treat proper nouns (PROPN) as regular nouns (NOUN). By basing the error types on language-agnostic Universal POS tags, it is hoped that at least some of the framework might extend to other languages in future (cf. Boyd, 2018; Kempfert and Köhn, 2018).

In the majority of cases, an edit may hence be assigned a POS-based error type if it meets the following conditions:

1. All tokens on both sides of the edit have the same POS tag, *and*
2. The edit does not meet any criteria for a more specific type.

Since POS and parse errors are generally unavoidable in learner text however (cf. Foster et al., 2008; Wagner and Foster, 2009; Sakaguchi et al., 2012b; Napoles et al., 2016a; Sakaguchi et al., 2017a), we also make use of other information to capture additional POS-based errors. For example, the following special rule uses dependency parse labels to recover errors when a purely POS-based comparison is uninformative; e.g. [*two (NUM, advmod)* → *too (ADV, advmod)*] in the context "I like cake two/too".

Dep. Label	POS
acomp	ADJ
amod	ADJ
advmod	ADV
det	DET
prep	PREP
prt	PART
punct	PUNCT

Table 5.4: A mapping between some parse and POS tags.

1. The POS tags of all tokens on both sides of the edit are not the same, *and*

2. The dependency labels of all tokens on both sides map to the same POS in Table 5.4.

Note that this rule is generally activated only as a last resort, as if the POS tags were incorrect, it is also likely that the dependency parse labels were incorrect.

There are also special rules for specific POS:

VERB

The following special VERB rule captures edits involving infinitival *to* and/or phrasal verbs; e.g. [*to eat* $\rightarrow \varepsilon$], [*consuming* $\rightarrow$ *to eat*] and [*look at* $\rightarrow$ *see*].

1. All tokens on both sides of the edit are either PART or VERB, *and*

2. The last token on each side has a different lemma.

PART

The following special PART rule captures edits where the tagger or parser confuses a preposition for a particle or vice versa; e.g. [*(look) at* $\rightarrow$ *(look) for*].

1. There is exactly one token on both sides of the edit, *and*

2. (a) The set of POS tags for these tokens is PREP and PART, *or*

   (b) The set of dependency labels for these tokens is prep and part.

DET and PRON

The following special rule differentiates between determiners and pronouns that have the same surface form; e.g. '*His* book' (DET) vs. 'This book is *his*' (PRON).

1. There is exactly one token on both sides of the edit, *and*

2. The set of POS tags for these tokens is DET and PRON, *and*

3. (a) The corrected token dependency label is *poss* (possessive determiner); i.e. DET, *or*

   (b) The corrected token dependency label is *nsjub* or *nsubjpass* (nominal subject), *dobj* (direct object), or *pobj* (prepositional object); i.e. PRON because determiners cannot be subjects or objects.

PUNCT

The following special PUNCT rule captures edits where a change in punctuation also affects the case of the following word; e.g. [. *Because* → *, because*] and [*Because* → *, because*].

1. The lower cased form of the last token is the same on both sides, *and*
2. All remaining tokens are punctuation.

5.3.3 Token tier

Contractions: CONTR

Contraction errors are mainly edits that involve expanding contractions to their full form; e.g. [*n't* → *not*] or [*'ve* → *have*]. The full list of contractions includes: {*'d*, *'ll*, *'m*, *n't*, *'re*, *'s*, and *'ve*}. They are captured by the following rule:

1. There is no more than one token on both sides of the edit, *and*
2. All tokens have the same POS, *and*
3. At least one token on either side is a member of the above set of 7 contractions.

An additional rule captures special case auxiliaries in contractions. Specifically, *can*, *shall* and *will* are respectively shortened to *ca*, *sha* and *wo* in *ca n't*, *sha n't* and *wo n't*. To prevent them being flagged as spelling errors, the following rule captures cases such as [*can* → *ca*] or [*wo* → *will*].

1. There is exactly one token on both sides of the edit, *and*
2. The set of strings for these tokens is *ca* and *can*, *sha* and *shall*, or *wo* and *will*.

Morphology: MORPH

The morphology error type mainly captures derivational morphology edits, e.g. [*quick (ADJ)* → *quickly (ADV)*], and cases where the POS tagger made a mistake, e.g. [*catch (NOUN)* → *catching (VERB)*]. They are captured by the following rule:

1. There is exactly one token on both sides of the edit, *and*
2. Both tokens have the same lemma or stem, *and*
3. Both tokens have nothing else in common.

Other: OTHER

Edits that are not captured by any rules are classified as OTHER. They typically include edits such as [*at (PREP)* → *the (DET)*], which have perhaps been improperly aligned, or else multi-token edits such as [*at his best* → *well*] which are much more semantic in nature.

Orthography: ORTH

Although the definition of orthography can be quite broad, we use it here to only refer to edits that involve case and/or whitespace changes; e.g. [*first* → *First*] or [*Bestfriend* → *best friend*].

1. The lower cased form of both sides of the edit with all whitespace removed results in the same string.

Spelling: SPELL

We use the latest British English Hunspell word list to identify spelling errors (see Section 5.2). It is straightforward to replace this word list with one for other varieties of English if needed. We assume the corrected side of an edit is always a valid word. Spelling errors must meet the following conditions:

1. There is exactly one token on both sides of the edit, *and*
2. The original token is entirely alphabetical (i.e. no numbers or punctuation), *and*
3. The original token is not in the Hunspell word list, *and*
4. The lower cased form of the original token is also not in the Hunspell word list, *and*
5. The original and corrected tokens do not have the same lemma, *and*
6. The original and corrected tokens share at least 50% of the same characters in the same relative order.

We check the dictionary twice with the original and lower cased form of a token because casing can produce false positives. For example *Cat* is not in the word list, but *cat* is, while *France* is in the word list, but *france* is not. We do not want edits such as [*Cat* → *Cats*] to be called spelling errors however, so must make sure both forms of the original word are invalid before considering the spelling error category.

Similarly, the character comparison condition is an approximation designed to filter out more complex edits such as [*greatful* → *pleased*]. While *greatful* is indeed a misspelling of *grateful*, it is ultimately replaced with a synonym *pleased*, and so is more a replacement adjective error rather than a spelling error. When spelling errors hence fail to meet the character comparison condition, they are classified according to their POS if they both have the same POS tag, or are else labelled OTHER.

Word Order: WO

We restrict our definition of word order errors to only include edits whose tokens exactly match on both sides of the edit; e.g. [*house white* → *white house*]. We also investigated allowing majority matches, e.g. [*I saw the man* → *the man saw me*], but found exact matches were qualitatively more reliable in practice.

1. The alphabetically sorted lists of lower cased tokens on both sides of the edit are identical.

5.3.4 Morphology tier

Adjective Form: ADJ:FORM

Adjective form edits involve changes between bare, comparative and superlative adjective forms; e.g. [*big* → *biggest*] or [*smaller* → *small*]. They are captured as followed:

1. There is exactly one token on both sides of the edit, *and*
2. Both tokens have the same lemma, *and*
3. (a) Both tokens are POS tagged as ADJ, *or*
   (b) Both tokens are parsed as *acomp* or *amod*.

A second rule captures multi-token adjective form errors; e.g. [*more big* → *bigger*]:

1. There are no more than two tokens on both sides of the edit, *and*
2. The first token on either side is *more* or *most*, *and*
3. The last token on both sides has the same lemma.

Noun Inflection: NOUN:INFL

Noun inflection errors are usually count-mass noun errors, e.g. [*advices* → *advice*], but also include cases such as [*countrys* → *countries*] and [*childs* → *children*]. They are a special kind of non-word error that must meet the following criteria:

1. There is exactly one token on both sides of the edit, *and*
2. The original token is entirely alphabetical (i.e. no numbers or punctuation), *and*
3. The original token is not in the Hunspell word list, *and*
4. The lower cased form of the original token is also not in the Hunspell word list, *and*
5. The original and corrected tokens have the same lemma, *and*
6. The original and corrected tokens are both POS tagged as NOUN.

Noun Number: NOUN:NUM

Noun number errors all involve count nouns that have been changed from singular to plural or vice versa; e.g. [*cat* → *cats*] or [*dogs* → *dog*]. They are captured by the following rule:

1. There is exactly one token on both sides of the edit, *and*
2. Both tokens have the same lemma, *and*
3. Both tokens are POS tagged as NOUN.

Since it is fairly common for the POS tagger to confuse nouns that look like adjectives, e.g. *musical*, a separate rule uses fine-grained POS tags to classify mis-tagged edits such as [*musical (ADJ)* → *musicals (NOUN)*]:

1. There is exactly one token on both sides of the edit, *and*
2. Both tokens have the same lemma, *and*
3. The original token is POS tagged as ADJ, *and*
4. The corrected token is POS tagged as a plural noun (NNS).

Note that this second rule was only found to be effective in the singular to plural direction and not the other way around.

Noun Possessive: NOUN:POSS

Noun possessive errors typically involve edits that change a possessive suffix on a noun phrase; e.g. [*(Tom)* ε → *(Tom) 's*] or [*(Chris) 's* → *(Chris) '*]. They are captured by the following rule:

1. There is exactly one token on both sides of the edit, *and*
2. At least one side of the edit is POS tagged as a possessive suffix (POS)

While the above rule handles possessive suffixes that have become separated from their dependent nouns as a result of an automatic alignment, the following rule handles multi-token edits where this is not the case; e.g. [*friends* → *friend 's*]:

1. There are exactly two tokens on at least one side of the edit, *and*
2. (a) The original tokens are POS tagged sequentially as NOUN and PART, *or*
   (b) The corrected tokens are POS tagged sequentially as NOUN and PART, *and*
3. The first token on both sides of the edit has the same lemma.

Verb Form: VERB:FORM

Verb form errors involve corrections between members of the set of bare infinitive, *to*-infinitive, gerund and participle forms; e.g. {*eat*, *to eat*, *eating*, *eaten*}. Since infinitives tend to have exactly the same form as non-3rd-person present tense forms however (cf. 'I want to *eat* cake' versus 'I *eat* cake'), we must use fine-grained POS tags to differentiate between them. The majority of verb form errors are hence captured by the following rule:

1. There is exactly one token on both sides of the edit, *and*
2. Both tokens have the same lemma, *and*
3. (a) i. Both tokens are POS tagged as VERB, *and*
   ii. Both tokens are preceded by a dependent auxiliary verb, *or*
   (b) i. Both tokens are POS tagged as VERB, *and*
   ii. At least one token is POS tagged as a gerund (VBG) or participle (VBN), *or*
   (c) i. Both tokens do not have the same POS tag, *and*
   ii. The corrected token is POS tagged as a gerund (VBG) or participle (VBN).

Since tense and agreement always fall on the first auxiliary within a verb phrase, if any, this means all other 1:1 verb edits with the same lemma can only be form errors; e.g. [*(has) eating* → *(has) eaten*] and [*(has) be (eaten)* → *(has) been (eaten)*]. In most other cases, a verb form error involves a gerund and/or participle. When the original token POS tag is not a verb, we instead defer to the corrected token POS tag to classify the edit; e.g. [*watch (NOUN)* → *watching (VBG)*].

Other types of verb form errors involve infinitival *to*. The next rule hence captures missing or unnecessary *to* particles that are not prepositions:

1. There is only one token on one side of the edit, *and*
2. That token is *to*, *and*
3. That token is POS tagged as PART, *and*
4. That token is not parsed as *prep*.

Finally, infinitival *to* may also be involved in more complex, multi-token edits; e.g. [*to eat* → *eating*]. These are captured by the following rule:

1. All tokens on both sides of the edit are POS tagged as PART or VERB, *and*
2. The last token on both sides has the same lemma.

Verb Inflection: VERB:INFL

Verb inflection errors are classified in a similar manner to noun inflection errors, and are a special kind of non-word error. Examples include: [*getted* → *got*], [*danceing* → *dancing*] and [*fliped* → *flipped*].

1. There is exactly one token on both sides of the edit, *and*
2. The original token is entirely alphabetical (i.e. no numbers or punctuation), *and*
3. The original token is not in the Hunspell word list, *and*
4. The lower cased form of the original token is also not in the Hunspell word list, *and*
5. The original and corrected tokens have the same lemma, *and*
6. The original and corrected tokens are both POS tagged as VERB.

Subject-Verb Agreement: VERB:SVA

Subject-verb agreement errors involve edits where the grammatical number of the subject does not agree with the grammatical number of the verb; e.g. [*(I) has* → *(I) have*]. They are captured as follows:

1. There is exactly one token on both sides of the edit, *and*
2. Both tokens have the same lemma, *and*
3. (a) Both tokens are *was* and *were*, *or*
   (b) i. Both tokens are POS tagged as VERB, *and*
       ii. At least one token is POS tagged as a 3rd-person present tense verb form (VBZ), *or*
   (c) i. Both tokens do not have the same POS tag, *and*
       ii. The corrected token is POS tagged as 3rd-person present tense verb form (VBZ).

While most subject-verb agreement errors are detected based on the VBZ POS tag, *was* and *were* are exceptional in that they are the only past tense verbs that have agreement morphology. As with verb form errors, we again rely on the corrected token POS tag alone when the original token POS tag is not a verb; e.g. [*watch (NOUN)* → *watches (VBZ)*].

Tense: VERB:TENSE

Verb tense errors are the most complicated out of all other error types and thus require the most rules. The main reason for this is because although tense can be inflectional, e.g. [*eat* → *ate*], it can also be expressed periphrastically by means of auxiliary verbs; e.g. [*ate* → *has eaten*]. This does not mean all auxiliary verbs are tense errors however, and auxiliary verbs can also be verb form or agreement errors; e.g. [*(is) be (eaten)* → *(is) being (eaten)*] and [*(it) are (eaten)* → *(it) is (eaten)*]. The majority of tense errors are hence captured by the following rules:

1. There is exactly one token on both sides of the edit, *and*
2. (a) i. Both tokens have the same lemma, *and*
   ii. Both tokens are POS tagged as VERB, *and*
   iii. At least one token is POS tagged as a past tense verb form (VBD), *or*
   (b) i. Both tokens have the same lemma, *and*
   ii. Both tokens are POS tagged as VERB, *and*
   iii. Both tokens are parsed as an auxiliary verb (*aux* or *auxpass*), *or*
   (c) i. Both tokens have the same lemma, *and*
   ii. Both tokens do not have the same POS tag, *and*
   iii. The corrected token is POS tagged as a past tense verb form (VBD), *or*
   (d) i. Both tokens do not have the same lemma, *and*
   ii. Both tokens are parsed as an auxiliary verb (*aux* or *auxpass*).

Similar to verb form and agreement errors, most tense errors are detected based on the VBD POS tag. If an auxiliary verb is not a form or agreement error however, we can use the parse to label it a tense error instead. Auxiliaries with different lemmas are usually edits such as [*has (eaten)* → *was (eaten)*].

We also have a special rule for certain auxiliaries in contractions (cf. Section 5.3.3). Specifically, the following rule captures edits such as [*ca (n't)* → *could (n't)*] and [*sha (n't)* → *wo (n't)*].

1. There is exactly one token on both sides of the edit, *and*
2. (a) If one side is *ca*, the other is not *can*, *or*
   (b) If one side is *sha*, the other is not *shall*, *or*
   (c) If one side it *wo*, the other is not *will*.

A further rule captures missing or unnecessary auxiliaries; e.g. [ε *(eaten)* → *has (eaten)*]. In particular, missing or unnecessary auxiliaries can never be form or agreement errors because those are always replacement errors.

1. There are no tokens on one side of the edit, *and*
2. All tokens on the other side are parsed as auxiliary verbs (*aux* or *auxpass*).

A related rule captures multi-token edits such as [*has been (eaten)* → *is (eaten)*]:

1. There is more than one token on at least one side of the edit, *and*
2. All tokens on both sides are parsed as auxiliary verbs (*aux* or *auxpass*).

Finally, another rule captures multi-token edits such as [*has eaten* → *was eating*], where both the auxiliaries and verb forms are different:

1. There is more than one token on at least one side of the edit, *and*
2. All tokens on both sides of the edit are POS tagged as VERB, *and*
3. The last token on both sides has the same lemma.

5.3.5 Rule order

As mentioned at the start of this section, the above rules have been presented in isolation and are actually carefully ordered in the implementation. This is because several rules interact and certain subsections of each error type are captured at different points in the code. For example, single-token replacement errors are handled before multi-token replacement errors, but instances of some error types, such as adjective form errors, meet both conditions; e.g. [*biggerer* → *biggest*] and [*most big* → *biggest*].

Another complex example concerns verb morphology errors. For example, while errors involving gerunds (VBG) or participles (VBN) are usually considered FORM, and errors involving past tense forms (VBD) are usually considered TENSE, edits such as [*VBG* → *VBD*], or vice versa, might ambiguously be FORM or TENSE. Similarly, errors involving a 3rd-person present tense form (VBZ) are usually considered subject-verb agreement, but edits such as [*VBZ* → *VBN*] might ambiguously be SVA or FORM. Although such cases are often caused by a POS tagging error, we ultimately resolved this issue by manually inspecting the data to determine an order of precedence. Specifically, we decided ambiguous errors are first considered FORM if one side is VBG or VBN, second considered SVA if one side is a present tense form (VBP or VBZ), and third considered TENSE if one side is VBD. In our experiments, this order seemed to produce the most reliable results, but must still be recognised as an approximation.

The final rule ordering was thus determined in a similar way, and so whenever it was ambiguous as to whether one rule should precede another or not, we simply tried both options and manually inspected the output to decide which order was better. Given there are roughly 50 rules in the code, it is hence very difficult to coherently describe the full rule order here, and so we instead refer the reader to the code for more information.[4]

5.4 Classifier evaluation

As this new error scheme is based entirely on automatically obtained properties of the data, there are no gold standard labels against which to evaluate. This makes it very difficult to formally quantify classifier performance.

[4]Specifically, we refer the reader to the functions named `getOneSidedType` and `getTwoSidedType` in `scripts/cat-rules.py` available here: `https://github.com/chrisjbryant/errant`

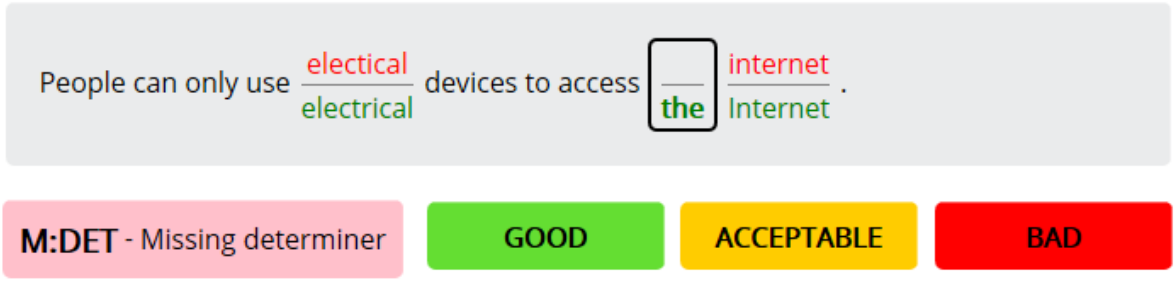

Figure 5.1: The classifier evaluation interface. Researchers were asked to rate the appropriateness of the error type for the circled edit in context.

Rater	Good	Acceptable	Bad
1	92.0%	4.0%	4.0%
2	89.5%	6.5%	4.0%
3	83.0%	13.0%	4.0%
4	84.5%	11.0%	4.5%
5	82.5%	15.5%	2.0%
OVERALL	**86.3%**	**10.0%**	**3.7%**

Table 5.5: The percent distribution for how each researcher rated the appropriateness of the predicted error types. E.g. Rater 3 considered 83% of all predicted types to be 'Good'.

This also means the classifier was only ever evaluated qualitatively during development. In particular, when a new rule or error type was added, I would simply print out and manually inspect the captured edits in order to verify rule reliability. While a more formal evaluation would have been preferable, it was impractical to repeatedly annotate the data with a gold standard while the error scheme was in flux.

Instead, we only carried out a more formal evaluation when the framework was finalised. Specifically, we randomly selected 100 edits from both FCE test and CoNLL-2014 (200 in total) and asked 5 GEC researchers to rate the appropriateness of the predicted error type in context as 'Good', 'Acceptable' or 'Bad' (Figure 5.1). 'Good' meant the chosen type was the most appropriate for the given edit, 'Acceptable' meant the chosen type was appropriate, but probably not optimum, while 'Bad' meant the chosen type was not appropriate for the edit. The researchers were warned that the edit boundaries had been determined automatically and hence might be unusual, but that they should focus on the appropriateness of the error type regardless of whether they agreed with the boundary or not.

The results for this evaluation are shown in Table 5.5. Significantly, all 5 raters considered at least 95% of the predicted error types to be either 'Good' or 'Acceptable', despite the degree of noise introduced by automatic edit extraction. Furthermore, whenever raters judged an edit as 'Bad', this could usually be traced back to a POS or parse error; e.g. [*ring* → *rings*] might ambiguously be a NOUN:NUM or VERB:SVA error. Inter-annotator agreement was also good at 0.724 κ_{free} (Randolph, 2005).

Finally, it is worth stating that the main purpose of this evaluation was not to determine the specific strengths and weaknesses of the classifier, but rather to ascertain how well humans believed the predicted error types characterised each edit. GEC is known to be a highly subjective task (Bryant and Ng, 2015) and so we were more interested in overall judgements than specific disagreements.

5.5 Summary

In this chapter, we introduced a new rule-based classifier designed to automatically classify edits using a new error type framework. One of the key strengths of this classifier is that by being dependent only upon automatically obtained mark-up information, it is entirely dataset independent and does not require labelled training data. This is in contrast with machine learning approaches which not only learn dataset specific biases, but also require large quantities of training data. We evaluated this classifier by asking 5 GEC researchers to rate the appropriateness of the error type for 200 randomly chosen edits in context, and found that 95% of the predicted types were considered either 'Good' (85%) or 'Acceptable' (10%). Combined with the automatic alignment method introduced in the previous chapter, the full system was named the ERRor ANnotation Toolkit (ERRANT) and released as part of this thesis.

CHAPTER 6

ERROR TYPE EVALUATION

Having described the ERRANT annotation system, we can now demonstrate its value by applying it to data. In particular, although one application of ERRANT is to standardise existing datasets, another is to annotate system output for the purposes of evaluation. This is significant because most systems are only ever evaluated in terms of overall performance, yet a system that performs poorly overall may still outperform another at specific error types. In fact a robust specialised system may actually be more desirable than a mediocre general system because precision is typically more important than recall in GEC. Without an error type analysis however, this information is completely unknown.

This chapter hence introduces the ERRANT scorer, a variant of the M2 scorer, and uses it to carry out a detailed evaluation of system error type performance for the first time. Specifically, we applied ERRANT to the system output produced by the 13 teams in the CoNLL-2014 shared task (Section 2.4.2) and compared the error type performance at different levels of granularity (Bryant et al., 2017). This is significant, because no other scorer is currently capable of doing this.

6.1 The ERRANT scorer

Given that existing scorers in GEC were unable to produce error type scores, I had to develop my own. Although it might be possible to adapt the M2 scorer to report error types, error type scoring is likely beyond the capabilities of both the I-measure and GLEU. In particular, multi-token error types such as word order errors do not fit well into the token-based I-measure, while n-gram based GLEU contains no concept of span or error type anyway.

Fortunately however, one benefit of explicitly annotating system hypotheses with ERRANT is that evaluation becomes much more straightforward. For example, while the M2 scorer must dynamically extract the edits that maximally match the reference using a Levenshtein alignment, ERRANT has already extracted and defined these edits for us. This means we only

need to compare the set of hypothesis edits $H = \{h_1, ..., h_n\}$ against the set of reference edits $R = \{r_1, ..., r_n\}$ for each sentence. Specifically, the intersection of hypothesis and reference edits are the true positives $|H \cap R|$, while the remaining hypothesis edits are false positives $|H - R|$ and the remaining reference edits are false negatives $|R - H|$. These counts can then be used to calculate precision, recall and F-score in the standard way as described in Section 2.3.1. The results can then be grouped by error type for the purposes of error type evaluation.

It is worth noting that when there is more than one reference sentence, we compare the hypothesis against each reference individually and choose the one that a) produces the highest F-score, b) has the highest number of TPs, c) has the lowest number of FPs and d) has the lowest number of FNs. This allows us to differentiate between references that have the same F-score, such as when there are no true positives. The M2 scorer takes the same approach, although this is not officially documented.

Finally, it is also worth stating that this scorer is much simpler than other scorers which typically incorporate edit extraction or alignment directly into their algorithms. Our approach, on the other hand, treats edit extraction and evaluation as separate tasks.

6.1.1 Gold references vs. auto references

Before evaluating an automatically annotated hypothesis against a reference, we must also address another mismatch: namely that hypothesis edits are extracted and classified automatically with ERRANT, while reference edits are extracted and classified manually according to different frameworks. Since our scorer reduces evaluation to a straightforward comparison between two sets of edits however, it is especially important that both the hypothesis and reference edits are processed in the same way. For example, if ERRANT extracts the hypothesis edit [*have eating* $\rightarrow$ *has eaten*] but the reference contains [*have* $\rightarrow$ *has*] and [*eating* $\rightarrow$ *eaten*], the scorer will erroneously count this as 1 FP and 2 FNs. This mismatch is the same problem the M2 scorer was designed to fix.

Since we can now automatically annotate the hypothesis edits however, we can solve this problem simply by reprocessing the references in the same way as the hypothesis. Specifically, we can apply ERRANT to the references such that each reference edit is subject to the same automatic extraction and classification criteria as each hypothesis edit. While it may seem unorthodox to discard gold references in favour of automatic references, this is necessary to minimise the difference between hypothesis and reference edits and also standardise the annotations to use the same error type framework.

To show that automatic references are viable alternatives to gold references, we re-evaluated each team in the CoNLL-2014 shared task using gold and auto references with both the M2 scorer and the ERRANT scorer. Table 6.1 hence shows that there is little difference between the overall scores for each team, and we formally validated this hypothesis for precision, recall and $F_{0.5}$ by means of bootstrap significance testing (Efron and Tibshirani, 1993). Ultimately,

	M2 Scorer		ERRANT	
Team	**Gold**	**Auto**	**Gold**	**Auto**
AMU	35.01	35.02	31.95	32.26
CAMB	37.33	37.33	33.34	33.97
CUUI	36.79	37.54	33.32	34.59
IITB	5.90	5.96	5.67	5.74
IPN	7.09	7.80	5.87	6.15
NTHU	29.92	29.69	25.61	25.61
PKU	25.32	25.33	23.40	23.61
POST	30.88	30.99	27.55	28.02
RAC	26.68	26.94	22.85	23.24
SJTU	15.19	15.37	14.85	15.03
UFC	7.84	7.91	7.84	7.91
UMC	25.37	25.45	23.07	23.45

Table 6.1: Overall scores for each team in CoNLL-2014 using gold and auto references with both the M2 scorer and the ERRANT scorer. All scores are $F_{0.5}$.

we found no statistically significant difference between automatic and gold references (1,000 iterations, $p > .05$) which leads us to conclude that our automatic references are qualitatively as good as human references.

6.1.2 Comparison with the M2 scorer

Despite using the same F-score metric, Table 6.1 also shows that the M2 scorer tends to produce slightly higher $F_{0.5}$ scores than the ERRANT scorer. We initially suspected the ERRANT scorer was underestimating performance somehow, but subsequently found instead that the M2 scorer actually tends to overestimate performance (cf. Felice and Briscoe (2015) and Napoles et al. (2015)).

In particular, given a choice between matching [*have eating* → *has eaten*] from Annotator 1 or [*have* → *has*] and [*eating* → *eaten*] from Annotator 2, the M2 scorer will always choose Annotator 2 because two true positives are worth more than one. Similarly, whenever the scorer encounters two false positives within a certain distance of each other (controlled by the `max_unchanged_words` parameter which is set to 2 by default), it merges them and treats them as one false positive; e.g. [*is a cat* → *are a cats*] is selected over [*is* → *are*] and [*cat* → *cats*] even though these edits are best handled separately. In other words, the M2 scorer exploits its dynamic edit boundary prediction to artificially maximise true positives and minimise false positives and hence produce slightly inflated scores.

	AMU			CAMB			CUUI			IITB		
Type	**P**	**R**	**$F_{0.5}$**	**P**	**R**	**$F_{0.5}$**	**P**	**R**	**$F_{0.5}$**	**P**	**R**	**$F_{0.5}$**
Missing	43.94	14.08	30.85	46.32	29.40	**41.54**	26.37	17.82	24.06	15.38	0.58	2.52
Replacement	37.22	26.99	34.59	37.37	28.24	35.10	45.90	23.06	**38.31**	29.85	1.50	6.23
Unnecessary	-	-	-	25.46	27.53	25.85	33.98	33.48	**33.88**	46.15	1.56	6.86

	IPN			NTHU			PKU			POST		
Type	**P**	**R**	**$F_{0.5}$**	**P**	**R**	**$F_{0.5}$**	**P**	**R**	**$F_{0.5}$**	**P**	**R**	**$F_{0.5}$**
Missing	2.94	0.29	1.04	34.59	11.30	24.49	33.33	4.29	14.16	31.33	12.97	24.41
Replacement	9.87	3.88	7.54	27.61	19.27	25.41	29.62	18.36	26.38	33.16	19.38	29.03
Unnecessary	0.00	0.00	0.00	33.86	15.80	27.56	0.00	0.00	0.00	26.32	33.26	27.46

	RAC			SJTU			UFC			UMC		
Type	**P**	**R**	**$F_{0.5}$**	**P**	**R**	**$F_{0.5}$**	**P**	**R**	**$F_{0.5}$**	**P**	**R**	**$F_{0.5}$**
Missing	1.54	0.27	0.79	62.50	4.40	17.16	-	-	-	40.08	23.35	35.06
Replacement	29.50	20.92	27.27	50.54	3.43	13.47	72.00	2.65	11.55	34.71	9.73	22.94
Unnecessary	0.00	0.00	0.00	18.04	11.73	16.29	-	-	-	16.59	17.14	16.70

Table 6.2: Precision, recall and $F_{0.5}$ for missing, unnecessary, and replacement errors for each team. A dash indicates the team's system did not attempt to correct the given error type (TP+FP = 0).

6.2 CoNLL-2014 shared task analysis

To demonstrate the value of ERRANT, we applied it to the system hypotheses and references of all teams in the CoNLL-2014 shared task (Section 2.4.2). Although ERRANT can be applied to any parallel dataset, we chose to evaluate on CoNLL-2014 because it constitutes the largest collection of publicly available GEC system output to date. The following sections hence analyse the error type performance of each participating system.

6.2.1 Operation error types

In the first category experiment, we simply investigated the performance of each system in terms of missing, replacement and unnecessary edits. The results are shown in Table 6.2 while the raw TP, FP and FN counts are shown in Appendix A, Table A.1.

Overall, the most surprising result was that five teams (AMU, IPN, PKU, RAC, UFC) failed to correct any unnecessary token errors at all. This is significant because unnecessary token errors account for roughly 25% of all errors in CoNLL-2014 and so failing to address them significantly limits a system's maximum performance. While the reason for this is clear in some cases, e.g. UFC's rule-based system was never designed to tackle unnecessary tokens (Gupta, 2014), it is less clear in others, e.g. there is no obvious reason why AMU's SMT system failed to learn when to delete tokens (Junczys-Dowmunt and Grundkiewicz, 2014). AMU's result is especially remarkable given that their system still came 3rd overall despite this limitation.

In contrast, CUUI's classifier approach (Rozovskaya et al., 2014a) was the most successful at correcting not only unnecessary token errors, but also replacement token errors, while CAMB's

hybrid MT approach (Felice et al., 2014) significantly outperformed all others in terms of missing token errors. It would hence make sense to combine these two approaches, and indeed recent research has shown this improves overall performance (Rozovskaya and Roth, 2016).

6.2.2 Main error types

In the second experiment, we computed precision, recall and $F_{0.5}$ for each of the 24 main error types for each team. The results are shown in Table 6.3, on the next page, while the raw TP, FP and FN counts are shown in Appendix A, Table A.2. The raw counts are sometimes more helpful in this case as they let us differentiate between error types that are more common than others.

		AMU	CAMB	CUUI	IITB	IPN	NTHU	PKU	POST	RAC	SJTU	UFC	UMC
ADJ	**P**	7.14	8.77	-	0.00	0.00	0.00	66.67	0.00	12.50	0.00	-	0.00
	R	9.09	12.82	-	0.00	0.00	0.00	6.67	0.00	3.23	0.00	-	0.00
	$F_{0.5}$	7.46	9.36	-	0.00	0.00	0.00	**23.81**	0.00	7.94	0.00	-	0.00
ADJ:FORM	**P**	55.56	75.00	75.00	100.00	0.00	25.00	100.00	50.00	4.17	-	-	75.00
	R	62.50	60.00	33.33	40.00	0.00	25.00	14.29	14.29	20.00	-	-	60.00
	$F_{0.5}$	56.82	71.43	60.00	**76.92**	0.00	25.00	45.45	33.33	4.95	-	-	71.43
ADV	**P**	6.25	11.54	0.00	0.00	0.00	0.00	0.00	0.00	0.00	4.76	-	8.62
	R	3.23	21.95	0.00	0.00	0.00	0.00	0.00	0.00	0.00	3.12	-	12.82
	$F_{0.5}$	5.26	**12.75**	0.00	0.00	0.00	0.00	0.00	0.00	0.00	4.31	-	9.23
CONJ	**P**	6.25	0.00	-	-	0.00	0.00	-	-	-	0.00	-	0.00
	R	7.14	0.00	-	-	0.00	0.00	-	-	-	0.00	-	0.00
	$F_{0.5}$	**6.41**	0.00	-	-	0.00	0.00	-	-	-	0.00	-	0.00
CONTR	**P**	29.17	40.00	46.15	-	0.00	-	-	33.33	0.00	66.67	-	28.57
	R	100.00	33.33	85.71	-	0.00	-	-	57.14	0.00	40.00	-	33.33
	$F_{0.5}$	33.98	38.46	50.85	-	0.00	-	-	36.36	0.00	**58.82**	-	29.41
DET	**P**	33.55	36.16	30.71	21.43	0.00	35.57	28.72	26.05	0.00	43.88	-	36.07
	R	14.17	42.93	51.80	0.92	0.00	28.42	7.87	49.76	0.00	12.65	-	23.66
	$F_{0.5}$	26.34	**37.34**	33.43	3.92	0.00	33.87	18.78	28.80	0.00	29.37	-	32.64
MORPH	**P**	54.88	58.33	50.00	28.57	2.22	26.23	19.53	26.47	31.48	66.67	36.36	41.94
	R	50.00	48.28	21.84	5.48	2.82	20.78	30.12	11.69	21.79	2.74	5.13	15.12
	$F_{0.5}$	53.83	**56.00**	39.75	15.50	2.32	24.92	21.01	21.13	28.91	11.76	16.39	30.95
NOUN	**P**	20.90	25.56	0.00	25.00	4.35	0.00	0.00	11.11	15.79	0.00	-	28.57
	R	12.28	19.01	0.00	2.22	2.22	0.00	0.00	1.96	2.94	0.00	-	9.71
	$F_{0.5}$	18.32	**23.91**	0.00	8.20	3.65	0.00	0.00	5.75	8.43	0.00	-	20.58
NOUN:INFL	**P**	60.00	55.56	55.56	-	25.00	100.00	71.43	66.67	66.67	0.00	-	-
	R	85.71	62.50	71.43	-	16.67	33.33	62.50	57.14	66.67	0.00	-	-
	$F_{0.5}$	63.83	56.82	58.14	-	22.73	**71.43**	69.44	64.52	66.67	0.00	-	-
NOUN:NUM	**P**	49.23	43.68	44.20	41.18	14.47	44.25	29.31	30.84	29.00	54.29	-	44.93
	R	56.64	54.26	59.75	3.91	11.40	48.08	42.92	56.67	36.79	10.38	-	17.13
	$F_{0.5}$	**50.55**	45.45	46.63	14.17	13.73	44.96	31.29	33.93	30.28	29.41	-	33.92
NOUN:POSS	**P**	20.00	66.67	-	-	-	-	14.29	-	0.00	25.00	-	50.00
	R	15.00	11.11	-	-	-	-	5.56	-	0.00	4.76	-	5.26
	$F_{0.5}$	18.75	**33.33**	-	-	-	-	10.87	-	0.00	13.51	-	18.52
ORTH	**P**	60.00	66.67	73.81	-	3.45	0.00	28.57	49.32	16.67	-	-	50.00
	R	11.11	41.18	59.62	-	4.55	0.00	6.90	64.29	49.12	-	-	17.24
	$F_{0.5}$	31.91	59.32	**70.45**	-	3.62	0.00	17.54	51.72	19.20	-	-	36.23
OTHER	**P**	19.30	23.03	10.34	0.00	2.38	1.43	16.67	9.76	0.00	0.00	-	11.83
	R	6.47	9.49	0.85	0.00	0.31	0.58	0.58	1.13	0.00	0.00	-	3.21
	$F_{0.5}$	13.82	**17.91**	3.18	0.00	1.02	1.11	2.56	3.86	0.00	0.00	-	7.69
PART	**P**	71.43	25.00	0.00	-	-	12.50	-	-	-	40.00	-	15.38
	R	18.52	14.81	0.00	-	-	16.67	-	-	-	9.52	-	10.00
	$F_{0.5}$	**45.45**	21.98	0.00	-	-	13.16	-	-	-	24.39	-	13.89
PREP	**P**	47.62	41.89	33.98	75.00	0.00	10.95	-	21.74	0.00	40.00	-	20.27
	R	16.33	35.77	13.67	1.42	0.00	12.70	-	2.16	0.00	7.55	-	12.88
	$F_{0.5}$	34.42	**40.51**	26.20	6.61	0.00	11.26	-	7.74	0.00	21.51	-	18.18
PRON	**P**	43.75	20.37	0.00	0.00	10.00	50.00	100.00	30.00	4.76	0.00	-	22.45
	R	10.00	13.92	0.00	0.00	1.75	2.94	1.59	4.84	1.56	0.00	-	14.86
	$F_{0.5}$	**26.12**	18.64	0.00	0.00	5.15	11.90	7.46	14.71	3.38	0.00	-	20.37
PUNCT	**P**	25.00	57.78	37.21	100.00	0.00	44.83	-	27.27	0.00	5.00	-	43.02
	R	3.57	15.76	10.60	1.87	0.00	9.15	-	6.38	0.00	0.98	-	23.42
	$F_{0.5}$	11.36	**37.68**	24.77	8.70	0.00	25.19	-	16.48	0.00	2.75	-	36.85
SPELL	**P**	76.92	77.55	0.00	0.00	25.00	0.00	44.17	68.63	73.98	-	-	100.00
	R	63.83	41.76	0.00	0.00	4.23	0.00	71.29	71.43	85.85	-	-	1.37
	$F_{0.5}$	73.89	66.20	0.00	0.00	12.61	0.00	47.81	69.17	**76.09**	-	-	6.49
VERB	**P**	18.31	17.58	-	0.00	7.14	0.00	12.50	0.00	0.00	0.00	-	18.52
	R	7.93	9.82	-	0.00	0.70	0.00	0.68	0.00	0.00	0.00	-	6.49
	$F_{0.5}$	14.51	**15.18**	-	0.00	2.51	0.00	2.78	0.00	0.00	0.00	-	13.51
VERB:FORM	**P**	35.48	35.82	66.67	0.00	8.93	36.46	31.43	23.08	35.48	28.57	-	28.89
	R	22.68	24.24	23.66	0.00	5.62	36.08	35.48	3.33	33.00	4.55	-	14.44
	$F_{0.5}$	31.88	32.70	**48.89**	0.00	7.99	36.38	32.16	10.56	34.96	13.89	-	24.07
VERB:INFL	**P**	100.00	100.00	-	-	50.00	100.00	50.00	100.00	100.00	-	0.00	-
	R	100.00	100.00	-	-	50.00	50.00	50.00	50.00	100.00	-	0.00	-
	$F_{0.5}$	**100.00**	**100.00**	-	-	50.00	83.33	50.00	83.33	**100.00**	-	0.00	-
VERB:SVA	**P**	50.00	44.58	56.57	50.00	24.53	51.40	63.33	36.73	35.96	59.09	84.21	61.54
	R	27.00	31.90	71.22	1.14	13.68	66.19	19.59	18.00	30.77	13.83	29.91	15.69
	$F_{0.5}$	42.72	41.29	59.00	5.21	21.17	53.80	43.78	30.41	34.78	35.71	**61.78**	38.83
VERB:TENSE	**P**	20.55	26.96	62.50	66.67	3.70	21.43	9.52	19.05	22.78	15.38	-	32.20
	R	8.82	17.92	2.99	1.26	0.62	1.85	3.68	2.34	20.69	2.50	-	11.66
	$F_{0.5}$	16.23	**24.49**	12.56	5.85	1.86	6.88	7.23	7.84	22.33	7.58	-	23.81
WO	**P**	-	38.89	0.00	66.67	-	-	-	0.00	0.00	-	-	41.18
	R	-	33.33	0.00	14.29	-	-	-	0.00	0.00	-	-	35.00
	$F_{0.5}$	-	37.63	0.00	38.46	-	-	-	0.00	0.00	-	-	**39.77**

Table 6.3: Precision, recall and $F_{0.5}$ for each team and error type. A dash indicates the team's system did not attempt to correct the given error type (TP+FP = 0). The highest F-score for each type is highlighted.

The first conclusion we can draw from this table is that CAMB was the most successful team overall in terms of error types, achieving the highest F-score in 11 categories, followed by AMU who scored highest in 5 categories. Only two teams (IPN and POST) did not achieve the best score in at least 1 category, which suggests different approaches and implementations complement different error types. Only CAMB attempted to correct at least one error from every category, although other teams typically only neglected a handful of categories.

Other notable results include:

- Content word errors (ADJ, ADV, NOUN and VERB) were unsurprisingly amongst the hardest to correct for all teams. This is likely because the confusion set for these errors is typically much larger than for other error types.
- Pronoun errors (PRON) were similarly difficult, probably because no system explicitly modelled coreference.
- Conjunction errors (CONJ) are not very common, but only AMU managed to correct any of them.
- Despite several teams building specialised classifiers for determiner and preposition errors (DET and PREP), CAMB's hybrid SMT approach still outperformed them. This might be because the classifiers were trained on a different error type framework however (i.e. NUCLE).
- CUUI's classifiers did however significantly outperform all other approaches on orthography (ORTH) and verb form errors (VERB:FORM), which perhaps suggests classifiers are particularly well-suited for these error types.
- Although spellcheckers are widespread nowadays, almost half of all teams did not seem to use them. This would have been an easy way to boost overall performance and perhaps even win the shared task for CUUI.
- Although UFC's rule-based approach was the best at verb agreement errors (VERB:SVA), CUUI and NTHU's classifiers were not very far behind.
- Conjunction (CONJ), possessive nouns (NOUN:POSS), particles (PART) and word order (WO) errors were each not handled at all by almost half of all participating teams. This may not be because these error types are difficult to correct however, but rather because they are all fairly rare.

6.2.3 Detailed error types

In addition to analysing operation tier and general error types, the modular design of our framework also allows us to evaluate error type performance at an even greater level of detail.

	AMU			CAMB			CUUI		
Type	**P**	**R**	**$F_{0.5}$**	**P**	**R**	**$F_{0.5}$**	**P**	**R**	**$F_{0.5}$**
M:DET	**54.39**	24.22	43.54	44.05	**51.39**	**45.34**	23.86	45.00	26.34
R:DET	21.43	27.63	22.44	19.21	**34.94**	21.11	**27.03**	24.10	**26.39**
U:DET	-	-	-	**43.18**	40.00	**42.51**	35.77	**66.52**	39.41
DET	33.55	14.17	26.34	**36.16**	42.93	**37.34**	30.71	**51.80**	33.43

Table 6.4: Detailed breakdown of determiner (DET) errors for the top three teams overall. The highest precision, recall and F-score for each error type is shown in bold.

For example, Table 6.4 shows the breakdown of determiner errors (DET) for the top three teams in terms of missing, replacement and unnecessary edits.

Note that this table is a representative example of detailed error type performance as an analysis of all error type combinations for all teams would take up too much space and may not be informative. The main application of a detailed error type analysis is to inform system developers about the specific strengths and weaknesses of their models rather than formally rank different approaches.

While CAMB's hybrid SMT approach achieved a higher score than either AMU's SMT approach or CUUI's classifier approach overall, our more detailed evaluation reveals that each of these approaches actually has different strengths and weaknesses. For example, although AMU and CAMB both achieved a similar F-score for missing determiners (M:DET), the detailed analysis reveals that AMU was generally more precise than CAMB, but CAMB had twice the recall. In contrast, CUUI had a slightly lower recall than CAMB, but also a much lower precision.

Similarly, although CAMB performed the worst at replacement determiners (R:DET) out of these three teams, it nevertheless had the highest recall, with CUUI having the highest precision. Finally, although CAMB slightly outperformed CUUI in terms of unnecessary determiners (U:DET), the detailed analysis shows that CUUI actually had a much higher recall than CAMB.

The conclusion we can draw from this analysis is hence that different approaches tend to struggle with replacement determiners more than missing or unnecessary determiners, but that classifiers and SMT may be able to complement each other in different ways.

6.2.4 Single vs. multi token

Another benefit of explicitly annotating all hypothesis edits is that edit spans become fixed; this means we can evaluate system performance in terms of edit size. Table 6.5 hence shows the overall performance for each team in terms of single and multi-token edits, where a multi-token edit is an edit with more than two tokens on at least one side. Roughly 220 out of the 2,100 edits in the CoNLL-2014 references are multi-token edits.

In general, teams did not do well at multi-token edits. In fact only three teams achieved scores greater than 10% $F_{0.5}$ and all of them used MT (AMU, CAMB, UMC). This is significant because recent work has suggested the main goal of GEC should be to produce fluent-sounding,

	Single Token Edits			Multi Token Edits		
Team	**P**	**R**	**$F_{0.5}$**	**P**	**R**	**$F_{0.5}$**
AMU	39.14	24.02	34.77	16.90	5.31	11.76
CAMB	36.58	32.67	35.73	27.22	17.06	**24.32**
CUUI	39.60	28.69	**36.80**	15.69	3.65	9.46
IITB	30.23	1.58	6.53	28.57	0.94	4.15
IPN	9.78	3.16	6.89	3.23	0.47	1.49
NTHU	30.18	21.06	27.78	0.00	0.00	0.00
PKU	29.83	15.67	25.26	25.00	1.40	5.70
POST	31.13	25.28	29.75	12.50	2.80	7.39
RAC	33.09	16.80	27.72	2.93	2.80	2.90
SJTU	29.56	6.34	17.05	10.00	0.47	1.98
UFC	72.00	2.16	9.66	-	-	-
UMC	29.83	15.71	25.28	19.09	9.38	15.81

Table 6.5: Each team's performance in terms of single and multi token edits. A multi token edit contains two tokens on at least one side of the edit.

rather than just grammatical sentences (Sakaguchi et al., 2016), even though this often requires complex multi-token edits. As no system is particularly adept at correcting multi-token errors however, robust fluency correction will likely require more sophisticated methods than are currently available.

6.2.5 Detection vs. correction

Another important aspect of GEC that is seldom reported in the literature is that of error detection; i.e. the extent to which a system can identify erroneous tokens in text. We calculate this at two different levels: the span level and the token level. To give an example, consider the following reference edits (Table 6.6):

	Start	**End**	**Type**	**Correction**
Edit 1	2	2	M:DET	the
Edit 2	3	5	R:WO	can only

Table 6.6: Two sample edits with start and end token spans, error types and corrections.

In correction, a hypothesis edit must exactly match the start offset, the end offset, and the correction string in order to be rewarded. Span level detection is thus a simplification of correction in that a hypothesis must instead only match the start and end span *regardless of the correction*. This is also how *Recognition* scores were calculated in the HOO shared tasks (Section 2.4.1). Token level detection is a variant of span level detection where a system is instead rewarded simply for identifying an edited token *regardless of the exact span and correction*. Since missing word edits start and end on a token boundary however, a token detection system is instead rewarded for flagging the token on the right as an error; i.e. the system should detect

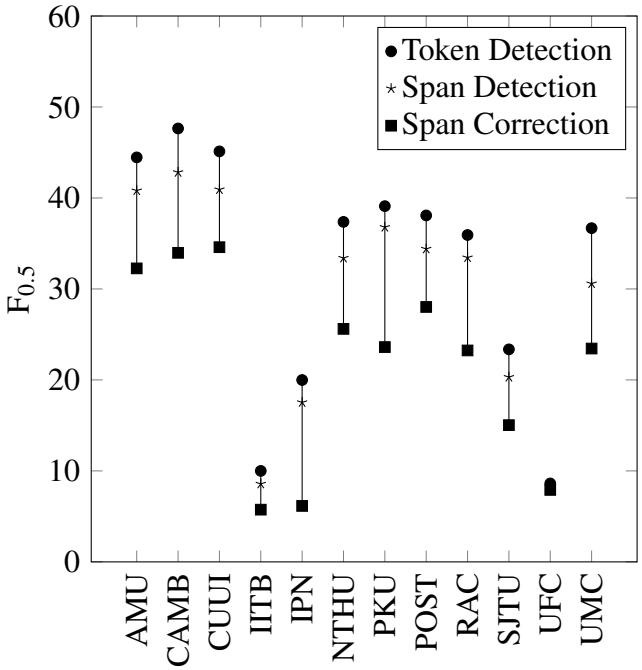

Figure 6.1: The difference between detection and correction scores for each team overall.

token 2-3 as an error in Table 6.6. Similarly, the system would be rewarded for detecting each of tokens 3-4 and 4-5 in Table 6.6 in a multi-token edit.

Figure 6.1 hence shows how each team performed in terms of token-based detection, span-based detection and span-based correction. It is interesting to note that while CAMB scored best overall in terms of both token-based and span-based detection, CUUI actually performed slightly better than CAMB at correction. This suggests CUUI was more successful at correcting errors than CAMB. In contrast, IPN and PKU are notable for detecting many more errors than they were able to correct, which suggests their approaches may be more valuable as detection systems. Ultimately, detection is an important part of GEC, and it is still valuable to know the location of an error in text, even if a system is unable to correct it.

Although we do not do so here, the ERRANT scorer can also evaluate error type detection scores for system output provided the text has also been corrected.

6.3 Summary

In this chapter, we demonstrated how ERRANT can be used to carry out a detailed error-type evaluation of system output. This enabled us to determine, for example, that almost half of all teams in the CoNLL-2014 shared task failed to correct any unnecessary word errors or use spellcheckers in their systems. We also found that different approaches have different strengths and weaknesses which we hope researchers will exploit to further improve performance in the future.

CHAPTER 7

LANGUAGE MODEL BASED CORRECTION

Although the main aim of this thesis was to develop a system capable of automatically annotating parallel data with error types as described in the previous chapter, a secondary aim was to also investigate how these error types could be used in a GEC system. For example, if we predict a verb form error at a given point in a text, we can restrict the set of correction candidates to only consider verb forms and not other types of correction such as unnecessary words or synonyms.

Furthermore, given that the majority of recent work has focused on developing **supervised** correction systems, I also chose to explore error types in the context of **unsupervised** correction systems. In particular, large amounts of training data are not always available for all languages, but it is still important to develop unsupervised correction systems that do not rely on manually annotated data. I hence developed my own unsupervised language model (LM) based system, motivated by the fact that LM based systems achieved competitive second and fourth places in the CoNLL-2013 and CoNLL-2014 shared tasks respectively (Kao et al., 2013; Lee et al., 2014).

Section 7.1 thus introduces the baseline system, which was previously reported in Bryant and Briscoe (2018),[1] while Section 7.2 describes how this system was tuned with error type thresholds at different levels of granularity. Final results are then reported in Section 7.3.

7.1 Baseline system

As described in Section 2.2.2, the core idea behind language modelling in GEC is that low probability sequences are more likely to contain grammatical errors than high probability sequences. For example, '*discuss about the problem' is expected to be a low probability sequence because it contains an error, while 'discuss the problem' or 'talk about the problem' are expected to be higher probability sequences because they do not contain errors. The goal of LM based GEC is thus to determine how to transform the former into the latter based on LM sequence probabilities.

[1] `https://github.com/chrisjbryant/lmgec-lite`

Step	Sentence									Probability
1	I	am	looking	forway	to	see	you	soon	.	-2.71
2 and 3	I	was -2.67 be -3.09 are -3.10 …	look -2.91 looks -2.93 looked -2.95 …	**forward -1.80** Norway -2.36 foray -2.70 …	of -2.98 in -2.99 ε -3.00 …	seeing -3.09 saw -3.25 sees -3.39 …	you	sooner -3.05 soonest -3.20	.	-
4	I	am	looking	**forward**	to	see	you	soon	.	-1.80
5	I	am	looking	forward	to	**seeing**	you	soon	.	-1.65

Table 7.1: A step-by-step example of the baseline LM methodology. All scores are log probabilities.

With this in mind, our approach is fundamentally a simplification of the algorithm proposed by Dahlmeier and Ng (2012a). It consists of 5 steps and is illustrated in Table 7.1:

1. Calculate the normalised log probability of an input sentence.
2. Build a confusion set, if any, for each token in that sentence.
3. Re-score the sentence substituting each candidate in each confusion set.
4. Apply the single best correction that increases the probability above a threshold.
5. Iterate steps 1-4.

7.1.1 Sequence probabilities

Hypothesis corrections are calculated in terms of normalised log probabilities at the sentence level. Normalisation by sentence length is necessary to help overcome the tendency for shorter sequences to have higher probabilities than longer sequences. Dahlmeier and Ng (2012a) similarly used normalised log probabilities to evaluate hypotheses, but did so as part of a more complex combination of other features. In contrast, Kao et al. (2013) and Lee et al. (2014) both evaluated hypotheses in terms of sliding five word windows (5-grams).

7.1.2 Confusion sets

One of the defining characteristics of LM-based GEC is that the approach does not necessarily require annotated training data. For example, spellcheckers and rules both formed key parts of Dahlmeier and Ng's and Lee et al.'s systems. While Lee et al. did additionally make use of annotated training data however, Dahlmeier and Ng instead employed separate classifiers for articles, prepositions and noun number errors trained only on native text.

In our baseline system, we focus on correcting the following error types in English: non-words, morphology, and articles and prepositions. These error types were chosen because they not only constitute a large group of errors in learner text, but were also relatively straightforward to build confusion sets for.[2] We additionally force the first character of every input sentence to be upper case.

[2]Note that targeting other error types may be more appropriate in other languages; e.g. Mandarin Chinese contains very little morphology.

- **Non-words:** We use CyHunspell[3] v1.3.0 with the latest British English Hunspell dictionaries[4] to generate correction candidates for non-word errors. Non-words include genuine misspellings, such as [*freind* → *friend*], and inflectional errors, such as [*advices* → *advice*]. Although CyHunspell is not a context sensitive spell checker, the proposed corrections are evaluated in a context sensitive manner by the language model.
- **Morphology:** Examples of morphological errors include noun number [*cat* → *cats*], verb tense [*eat* → *ate*] and adjective form [*big* → *bigger*]. To generate correction candidates for morphological errors, we use an Automatically Generated Inflection Database (AGID),[5] which contains all the morphological forms of many English words. The confusion set for a word is hence derived from this database.
- **Articles and Prepositions:** Since articles and prepositions are closed class words, we defined confusion sets for these error types manually. Specifically, the article confusion set consists of $\{\varepsilon$, a, an, the$\}$, while the preposition confusion set consists of the top ten most frequent prepositions: $\{\varepsilon$, about, at, by, for, from, in, of, on, to, with$\}$. Both sets also contain a null character which represents a deletion.

Unlike Dahlmeier and Ng and Lee et al., we do not handle missing word errors ($\sim$20% of all errors) because it is often difficult to know where to insert them. For example, it is impractical to try inserting every kind of determiner or preposition in front of every single word in a sentence.

7.1.3 Iteration

The main reason to iteratively correct only one word at a time is because errors sometimes interact. For example, correcting [*see* → *seeing*] in Table 7.1 initially reduces the log probability of the input sentence from -2.71 to -3.09. After correcting [*foray* → *forward*] however, [*see* → *seeing*] subsequently increases the probability of the sentence from -1.80 to -1.65 in the second iteration. Consequently, correcting the most serious errors first, in terms of language model probability increase, often helps facilitate the correction of less serious errors later. Dahlmeier and Ng and Lee et al. also both used iterative correction strategies in their systems, but did so as part of a beam search and pipeline approach respectively.

7.2 Tuning

Although our LM based approach does not require annotated training data, it does require a small amount of annotated development data for tuning purposes. For example, although the edit [*am* → *was*] in Table 7.1 increases the normalised sentence log probability from -2.71 to -2.67,

[3] `https://pypi.python.org/pypi/CyHunspell`
[4] `https://sourceforge.net/projects/wordlist/files/speller/2018.04.16/`
[5] `http://wordlist.aspell.net/other/`

this is such a small improvement that it is likely to be a false positive. In order to minimise false positives, we hence set thresholds such that a candidate correction must improve the average token probability of the sentence by at least X% before it is applied. While it may be unusual to use percentages in log space, this is just one way to compare the difference between two sentences which we found worked well in practice.

7.2.1 Error type thresholds

Since one of the main goals of this chapter is to show how error types can be used in a GEC system, we experimented with four different types of threshold at different levels of granularity:

1. **Global**: A single threshold for all errors.
2. **Operation**: A threshold for replacement (R) and unnecessary (U) errors.
3. **Main**: A threshold for each of the 25 main error types in ERRANT; e.g. determiners (DET).
4. **Detailed**: A threshold for each of the 54 detailed error types in ERRANT; e.g. replacement determiners (R:DET).

Specifically, the global threshold is the baseline threshold that treats all error types equally, while the operation thresholds split the global threshold into two thresholds for replacement and unnecessary errors. We did not include a threshold for missing word errors at this stage because the baseline system is unable to correct them. In contrast, the main and detailed thresholds are the more sophisticated thresholds that make finer distinctions between error types.

While it is possible to exhaustively test all the combinations of global and operation thresholds where the total number of thresholds is relatively small (i.e. ≤ 2), this quickly becomes impractical for larger numbers of thresholds. I thus developed the following method to automatically optimise the main and detailed error type thresholds:

1. Initialise all error type thresholds as the global optimum and evaluate using ERRANT.
2. Sort the error types by frequency, largest to smallest, in terms of the sum of true positives (TP), false positives (FP) and false negatives (FN).
3. Iteratively increase or decrease the threshold of the most frequent error type until the overall $F_{0.5}$ performance decreases or remains the same.
4. Repeat step 3 for the next most common error type until all thresholds are optimised.

Dataset	Tokenizer	Sentences	References	Edits
CoNLL-2013	NLTK	1381	1	3404
CoNLL-2014	NLTK	1312	2	6104
FCE-dev	spaCy	2371	1	4419
FCE-test	spaCy	2805	1	5556
JFLEG-dev	NLTK	754	4	10576
JFLEG-test	NLTK	747	4	10082

Table 7.2: Various statistics about the development and test datasets we use.

Weights are initialised according to the best global threshold in order to approximate a good starting point in the search space. This hopefully increases the chance of the algorithm converging on an optimum threshold combination. Thresholds were then optimised by frequency because the most common error types typically have a greater impact on overall performance than the rarer error types. Note however that error types where $TP + FP = 0$ were not optimised because this implies they were not handled by the baseline system; there is no point tuning a parameter that has no effect on the final score.

7.2.2 Data and resources

In all our experiments, we used a 5-gram language model trained on the One Billion Word Benchmark dataset[6] (Chelba et al., 2014) with the KenLM[7] language modelling toolkit (Heafield, 2011). While a neural language model would likely perform better than an n-gram model, efficient training on such a large amount of data is still an active area of research (Grave et al., 2017).

We used the following small datasets to tune and test our system: CoNLL-2013 and CoNLL-2014 (Ng et al., 2013, 2014), the public FCE (Yannakoudakis et al., 2011), and JFLEG (Napoles et al., 2017) (see Section 2.1). All the datasets were also preprocessed and standardised with ERRANT. This preprocessing is especially important for JFLEG, which does not contain explicit error annotations and so otherwise cannot be evaluated in terms of F-score. Finally, note that results on CoNLL-2014 and JFLEG are typically higher than on other datasets because they contain more than one set of reference annotations. See Table 7.2 for more information about each of the development and test sets.

7.2.3 Tuning experiments

In our first experiment, we simply investigated global thresholds in the range of 0-10% on the development sets. The results were calculated using the ERRANT scorer and are presented in Figure 7.1.

[6] `http://www.statmt.org/lm-benchmark/`
[7] `https://github.com/kpu/kenlm`

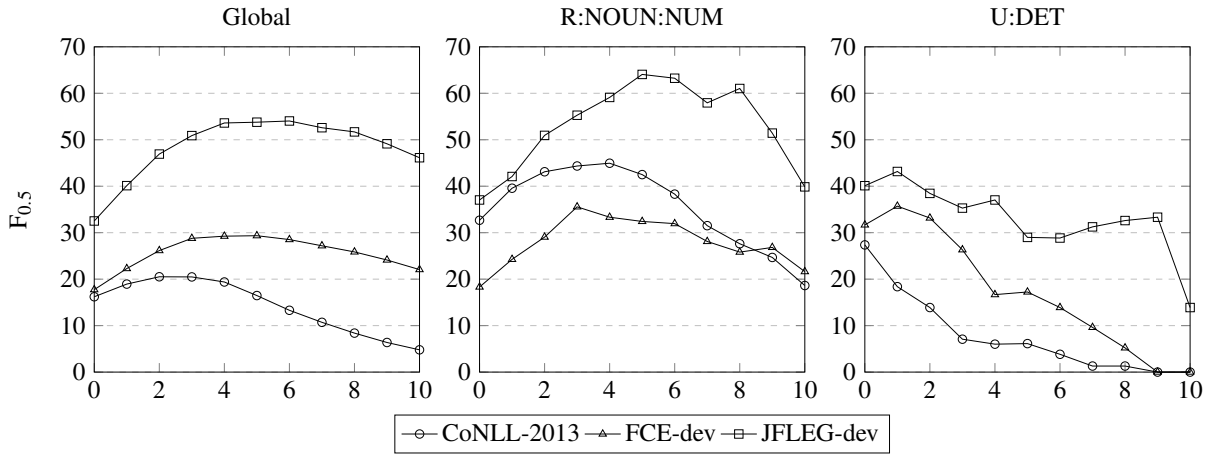

Figure 7.1: The effect of changing the global threshold on all errors, noun number errors and unnecessary determiner errors. The detailed error types are representative of all error types.

The first observation we can make from this figure is that the globally optimum threshold for CoNLL-2013 (2%) is very different from that of FCE-dev (5%) and JFLEG-dev (6%). This is most likely because each dataset has a different error type distribution; for example, spelling errors make up just 0.3% of all errors in CoNLL-2013, but closer to 10% in FCE-dev and JFLEG-dev. This suggests that different error types contribute to the overall score in different ways.

To test this hypothesis, Figure 7.1 also shows the effect of changing the global threshold on noun number (R:NOUN:NUM) and unnecessary determiner (U:DET) errors. In particular, the graphs show that R:NOUN:NUM errors tend to prefer higher valued thresholds, while U:DET errors tend to prefer lower valued thresholds; this supports the idea that error type thresholds can improve system performance. It is also worth noting that these error type graphs are also a lot less smooth than the global graph. This is because the sample size of the error type results is significantly smaller than the global result, and so $F_{0.5}$ scores are much more sensitive to threshold variation. This highlights the importance of having enough samples to calculate reliable thresholds.

Having motivated the need for error type thresholds, Table 7.3 shows the tuned results for all development sets using the different tuning strategies. The threshold values used to make this table are shown in Appendix B, Table B.1 and B.2.

These results first show that increasing the number of error type thresholds at finer levels of granularity almost always improves the overall performance of the system. Although the difference between the scores for the global and operation thresholds is typically small, this can be explained by the fact that the discriminative power between 1 and 2 thresholds is also small. Additionally, the baseline system only handles very specific unnecessary word errors (namely U:DET and U:PREP), so the U threshold is further limited by the fact that it regulates significantly fewer errors than the R threshold.

Dataset	Setting	TP	FP	FN	P	R	$F_{0.5}$
CoNLL-2013	Global (1)	372	1053	2998	26.11	**11.04**	20.51
	Operation (2)	302	634	3068	32.26	8.96	21.23
	Main (24)	359	729	3011	33.00	10.65	23.25
	Detailed (54)	367	716	3003	**33.89**	10.89	**23.82**
FCE-dev	Global (1)	419	495	3064	45.84	12.03	29.35
	Operation (2)	426	510	3057	45.51	12.23	29.47
	Main (24)	442	431	3041	**50.63**	12.69	31.68
	Detailed (54)	446	438	3037	50.45	**12.81**	**31.77**
JFLEG-dev	Global (1)	407	131	1207	75.65	25.22	54.04
	Operation (2)	407	131	1207	75.65	25.22	54.04
	Main (24)	440	121	1202	**78.43**	26.80	**56.61**
	Detailed (54)	445	128	1200	77.66	**27.05**	56.52

Table 7.3: Table showing how different error type thresholds affect system performance in three different development sets.

In contrast, there is a large improvement between 2 and 24 thresholds. Although there is again only a small performance difference between 24 and 54 thresholds, this can again be explained by the fact that, for example, 15 of the detailed ERRANT error type thresholds regulate missing word errors which are not corrected by the system. Consequently, the most detailed error type thresholds may only be useful for discriminating between a small number of categories, such as R:DET and U:DET. It may nevertheless be helpful to preserve all category thresholds in the system in case they are handled by future upgrades.

Ultimately, these results show that error type thresholds can be successfully integrated into a GEC system, and so I only report scores using the the global and detailed thresholds on the main test sets.

7.3 Results and discussion

The main results applying tuned development thresholds to their respective test sets are shown in Table 7.4 and are compared against several state of the art systems. Unfortunately, we cannot compare results with Dahlmeier and Ng (2012a) because this system is neither publicly available nor has it previously been evaluated on these test sets. Results are reported in terms of ERRANT $F_{0.5}$, M2 $F_{0.5}$, and GLEU. Results are not available in all cases because system output was never made available.

It is worth reiterating that our main intention was not to necessarily improve upon the state of the art, but rather i) quantify the extent to which a minimally supervised LM based approach could compete against more sophisticated models trained on millions of words of annotated text, and ii) investigate whether error type thresholds could increase baseline performance. This is significant because minimally supervised systems may be particularly attractive to GEC

		ERRANT			M2 Scorer			
Test Set	**System**	**P**	**R**	**$F_{0.5}$**	**P**	**R**	**$F_{0.5}$**	**GLEU**
CoNLL-2014	*Global*	36.01	19.91	31.00	40.00	20.89	33.81	59.30
	Detailed	41.58	18.71	33.41	44.46	19.03	35.08	59.63
	POST14	31.01	20.79	28.23	34.51	21.73	30.88	57.28
	AMU14	37.97	20.21	32.29	41.62	21.40	35.01	58.96
	$CU17_{CU16}$	-	-	-	51.09	25.30	42.44	66.42
	$CU17_{AMU16}$	-	-	-	59.88	32.16	51.08	68.69
	AMU18	-	-	-	66.77	34.49	56.25	-
FCE-test	*Global*	44.85	11.85	28.81	47.63	12.27	30.21	59.80
	Detailed	46.28	12.48	30.03	48.97	12.90	31.40	60.16
	$CU17_{CU16}$	-	-	-	64.25	36.13	55.60	71.76
	$CU17_{AMU16}$	-	-	-	43.34	19.88	35.07	64.78
JFLEG-test	*Global*	77.55	25.61	55.17	79.07	26.39	56.51	48.16
	Detailed	76.94	27.31	56.43	78.92	28.08	57.94	48.81
	JHU17	-	-	-	65.80	40.96	58.68	53.98
	NUS18	-	-	-	-	-	66.80	57.47
	AMU18	-	-	-	-	-	-	61.50

Table 7.4: Table showing ERRANT, M2 and GLEU scores for our LM approach (using a global or detailed error type threshold) compared against state of the art results on several test sets. POST14 is the LM system that came 4th on the CoNLL-2014 shared task (Lee and Lee, 2014), while AMU14 is the SMT system that came 3rd (Junczys-Dowmunt and Grundkiewicz, 2014). The CU17 systems are different versions of reranked SMT systems originally reported in Yannakoudakis et al. (2017), while JHU17, AMU18 and NUS18 are all state of the art NMT systems (Sakaguchi et al., 2017b; Grundkiewicz and Junczys-Dowmunt, 2018; Chollampatt and Ng, 2018a). Out of all these systems, only ours does **not** use annotated training data.

researchers who work with low-resource languages.

With this in mind, we were first pleased to improve upon the previous best LM approach in the CoNLL-2014 shared task by POST14 (Lee and Lee, 2014); this is especially significant since we also did so without any annotated training data. While this would have still placed our global threshold system third overall however, the detailed error type threshold system further improved performance just enough to beat AMU14 (Junczys-Dowmunt and Grundkiewicz, 2014) to third place (35.08 $F_{0.5}$ vs. 35.01 $F_{0.5}$). This not only shows how error types can improve a GEC system, but also how a minimally supervised approach could even beat an early SMT system that relied on large amounts of annotated training data.

We were also surprised by the high performance on JFLEG-test, where the detailed error type threshold system came to within 1 $F_{0.5}$ of one of the best supervised systems JHU17 (Sakaguchi et al., 2017b). This is especially surprising given our system only corrects a limited number of error types (roughly 14 out of the 55 in ERRANT[8]), and so can maximally correct only 40-60% of all errors in each test set.

[8]R:ADJ:FORM, R:DET, R:MORPH, R:NOUN:INFL, R:NOUN:NUM, R:ORTH, R:PREP, R:SPELL, R:VERB:FORM, R:VERB:INFL, R:VERB:SVA, R:VERB:TENSE, U:DET, U:PREP

One possible explanation for this, however, is that unlike CoNLL-2014 and FCE-test which were corrected with minimal edits, JFLEG was corrected for fluency (Sakaguchi et al., 2016), and so it intuitively makes sense that LM based approaches perform better with fluent references. An alternative and more likely explanation however, is that JFLEG's error type distribution simply happens to correlate with our LM system's capabilities more than the other test sets. For example, forcing sentences to start with an upper case character increases performance by only 0.3 $F_{0.5}$ in CoNLL-2014 and FCE-test, but by over 5 $F_{0.5}$ in JFLEG-test. This shows that even the simplest of correction strategies can have a significant effect on results depending on the error type distribution of the test set. This motivates the need for larger, more balanced test sets in the future.

Finally, although we did not perform as well against the latest state-of-the-art results on CoNLL-2014 and FCE-test, we also note a large discrepancy between results tuned on different datasets. For example, while $CU17_{AMU16}$ tuned on CoNLL-2013 achieves significantly better scores than $CU17_{CU16}$ tuned on FCE-dev on the CoNLL-2014 test set (51.04 vs. 42.44 $F_{0.5}$), it achieves significantly worse scores than $CU17_{CU16}$ on FCE-test (35.07 vs. 55.60 $F_{0.5}$). This highlights the possibility that systems may be overfitting to their training corpora and that systems may not be very generalisable. It is hence unfortunate that the current best system AMU18 (Grundkiewicz and Junczys-Dowmunt, 2018) only reports specific metrics on specific test sets and generalisability cannot be determined.

7.4 Summary

In this chapter, I motivated the need to re-examine unsupervised approaches in GEC and consequently developed a new prototype language model based system. Although this system was only able to correct non-words, morphology and certain determiner and preposition errors, it was still somewhat competitive with more sophisticated approaches that relied on large quantities of training data. This suggests that the distance between supervised and unsupervised systems is not as large as was previously thought and that it should still be possible to build correction systems for low-resource languages.

In addition to developing a baseline unsupervised system, I also investigated how to incorporate different error type thresholds into the system to improve performance. The results showed that the biggest gains were made by using the most detailed error types in ERRANT, typically increasing scores by roughly 2 $F_{0.5}$ on several datasets. This hence demonstrates that ERRANT can also be used to improve a system, as well as in preprocessing and evaluation.

CHAPTER 8

CONCLUSION

This thesis has explored automatic error type annotation in the context of grammatical error correction (GEC). Its main contribution is the ERRor ANnotation Toolkit (ERRANT), which automatically aligns parallel original and corrected sentence pairs in as linguistically intuitive a way as possible and then classifies the edits according to a new rule-based error type framework. Possible applications of this toolkit include:

- standardising existing corpora,
- reducing annotator workload,
- quickly annotating system output,
- facilitating detailed error type evaluation,
- improving existing GEC systems.

In order to build this toolkit, Chapter 3 first discussed various preprocessing issues in several of the largest and most popular GEC corpora and described how to overcome them. In particular, human annotators typically annotate *essays* at the *character* level, but GEC systems typically process *sentences* at the *token* level, so the first challenge was to transform character edits in essays into token edits in sentences. This was achieved by expanding character spans that did not precisely map to tokens and sentence tokenising essay paragraphs. This process was not flawless however, and a small number of edits, such as those that crossed sentence boundaries, were lost during the transformation. Future work might explore how to recover some of these lost edits, but as they usually account for less than 1% of all edits, preprocessing is largely a solved problem.

Having preprocessed the input corpora, Chapter 4 described a method to automatically align the parallel sentences according to human linguistic intuition. In particular, conventional alignment algorithms such as Levenshtein do not take linguistic information into account, and so we incorporated features such as part of speech, lemma and character difference into a custom

substitution cost function. While this somewhat improved performance in terms of comparison with human edits, up to 30% of all human edits also involve more than one token on either side, and so we additionally implemented a rule-based merging strategy to combine certain edits. This new alignment and merging strategy outperformed all previous approaches by approximately 6 F_1 on several datasets and is the current state of the art. Although machine learning approaches might produce better results in future, statistical classifiers would also learn corpus specific biases and human inconsistencies, and so we consider our rule-based approach to be conceptually more robust.

After aligning the sentence pairs, the edits were next extracted and classified according to a new dataset-agnostic rule-based error type framework (Chapter 5). This framework was designed to be a compromise between informativeness and practicality and was inspired by the best characteristics of the NUCLE and CLC error type frameworks. One of the benefits of a rule-based approach in this context is that the classifier does not require annotated training data and so does not suffer from any corpus bias. To validate the efficacy of this classifier, we asked 5 researchers to rate the appropriateness of the automatic error types in 200 randomly chosen edits, and found 95% of all types were considered either 'Good' (85%) or 'Acceptable' (10%).

With respect to the aims of this thesis, this means I was successful in building the first fully automatic error type annotation system. Future work might investigate the contribution of new rules or error types to the framework, although it is worth stating that the number of rules and rule interactions is already fairly complex. Since the vast majority of rules depend only on universal dependency POS tags however, it should nevertheless be fairly straightforward to extend the framework to other languages, as has recently been done for German (Boyd, 2018; Kempfert and Köhn, 2018).

In terms of the other aims of this thesis, I also applied ERRANT to the output produced by each team in the CoNLL-2014 shared task to carry out a detailed error type analysis of system performance for the first time (Chapter 6). Results showed that almost half of all participants failed to correct spelling errors, despite the prevalence of online spellcheckers, and only one team attempted to correct at least one error of every type. This suggests researchers may be overlooking some of the most obvious ways to improve their systems, which is something significant that would otherwise be unknown without our error type analysis. Most teams nevertheless still achieved the best score in at least one category however, which suggests better results might be obtained by combining systems with complementary strengths. It is hoped that researchers will make similar use of ERRANT to diagnose weaknesses and identify avenues for improvement when developing their own systems in the future.

In addition to automatic annotation, Chapter 7 also introduced a baseline language model based approach to GEC to highlight the fact that unsupervised methods have largely been neglected in GEC. Although supervised approaches have become increasingly successful in terms of what they can correct, they also suffer from the fact that no training corpus will

ever completely represent the space of all possible corrections. Consequently, unsupervised approaches may be vital for helping systems adapt to unseen errors and can additionally be used to develop correction systems for low-resource languages. With this in mind, I not only succeeded in my goal to build an unsupervised correction system, but also created a system that was surprisingly competitive with state-of-the-art systems on several datasets, despite significant limitations in terms of the types of errors it could correct. One of the most obvious ways to improve this system would hence be to add new resources that enable the system to correct additional error types such as missing and unnecessary word errors and synonyms (e.g. via WordNet[1]).

On a similar topic, I also fulfilled another aim of this thesis by showing how my baseline language model system could be further improved by means of error type information. In particular, I showed that different error type corrections improve sentence probabilities by different amounts and that we can use error type thresholds to control whether a particular candidate correction is applied or not. More concretely, the system that used the most detailed error type thresholds increased performance over the baseline system by roughly 2 $F_{0.5}$ on several datasets. This ultimately validated the hypothesis that automatic annotation is not only useful in terms of system evaluation, but can also be used in system development.

Finally, it is worth stating that my experiments with language model based correction are fairly preliminary and that there is certainly room for improvement. For example, neural language models have been shown to be more reliable than n-gram models (Bengio et al., 2003), so there is definitely scope to increase the fidelity of the sentence probability statistics. In fact it may also be desirable to abstract away from language model probabilities and instead estimate edit quality by means of a more complex function that takes, for example, part-of-speech tags or parse probabilities into account. Additionally, another possible extension could be to redefine the problem in terms of a finite state transducer (FST) to allow for a more efficient representation and exploration of the search space. Ultimately, these experiments showed that unsupervised approaches to GEC are still a promising area of research that deserves to be explored further.

To summarise, the main contribution of this thesis, ERRANT, is a system capable of automatically annotating error types in parallel text. ERRANT can not only be used to carry out detailed error type analyses of system performance and hence inform system development, but can also be incorporated into a correction system itself to improve performance. It is hoped that ERRANT will be a valuable tool in helping to drive the field forward in future.

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

BIBLIOGRAPHY

Antti Arppe. 2000. Developing a grammar checker for Swedish. In *Proceedings of the 12th Nordic Conference of Computational Linguistics (NODALIDA 1999)*. Department of Linguistics, Norwegian University of Science and Technology, Norway, pages 13–27. http://www.aclweb.org/anthology/W99-1002. Cited on page 27.

Hiroki Asano, Tomoya Mizumoto, and Kentaro Inui. 2017. Reference-based metrics can be replaced with reference-less metrics in evaluating grammatical error correction systems. In *Proceedings of the Eighth International Joint Conference on Natural Language Processing (Volume 2: Short Papers)*. Asian Federation of Natural Language Processing, pages 343–348. http://aclweb.org/anthology/I17-2058. Cited on pages 33 and 37.

Yoshua Bengio, Réjean Ducharme, Pascal Vincent, and Christian Janvin. 2003. A neural probabilistic language model. *J. Mach. Learn. Res.* 3:1137–1155. http://dl.acm.org/citation.cfm?id=944919.944966. Cited on page 109.

Gabor Berend, Veronika Vincze, Sina Zarrieß, and Richárd Farkas. 2013. LFG-based features for noun number and article grammatical errors. In *Proceedings of the Seventeenth Conference on Computational Natural Language Learning: Shared Task*. Association for Computational Linguistics, pages 62–67. http://www.aclweb.org/anthology/W13-3608. Cited on page 40.

Adam L. Berger, Vincent J. Della Pietra, and Stephen A. Della Pietra. 1996. A maximum entropy approach to natural language processing. *Computational Linguistics* 22(1). http://www.aclweb.org/anthology/J96-1002. Cited on page 29.

Shane Bergsma, Dekang Lin, and Randy Goebel. 2009. Web-scale n-gram models for lexical disambiguation. In *Proceedings of the 21st International Jont Conference on Artifical Intelligence*. Morgan Kaufmann Publishers Inc., San Francisco, CA, USA, pages 1507–1512. http://dl.acm.org/citation.cfm?id=1661445.1661687. Cited on page 28.

Pinaki Bhaskar, Aniruddha Ghosh, Santanu Pal, and Sivaji Bandyopadhyay. 2011. May I check the English of your paper!!! In *Proceedings of the 13th European Workshop on Natural Language Generation*. Association for Computational Linguistics, pages 250–253. http://www.aclweb.org/anthology/W11-2839. Cited on page 38.

Pinaki Bhaskar, Aniruddha Ghosh, Santanu Pal, and Sivaji Bandyopadhyay. 2012. Detection and correction of preposition and determiner errors in English: HOO 2012. In *Proceedings of the Seventh Workshop on Building Educational Applications Using NLP*. Association for Computational Linguistics, Montréal, Canada, pages 201–207. http://www.aclweb.org/anthology/W12-2023. Cited on pages 38 and 39.

Steven Bird, Ewan Klein, and Edward Loper. 2009. *Natural Language Processing with Python*. O'Reilly Media, Inc., 1st edition. Cited on page 26.

Juhani Birn. 2000. Detecting grammar errors with Lingsoft's Swedish grammar checker. In *Proceedings of the 12th Nordic Conference of Computational Linguistics (NODALIDA 1999)*. Department of Linguistics, Norwegian University of Science and Technology, Norway, pages 28–40. http://www.aclweb.org/anthology/W99-1003. Cited on page 27.

Tiberiu Boroş, Stefan Daniel Dumitrescu, Adrian Zafiu, Verginica Barbu Mititelu, and Ionut Paul Vaduva. 2014. RACAI GEC – a hybrid approach to grammatical error correction. In *Proceedings of the Eighteenth Conference on Computational Natural Language Learning: Shared Task*. Association for Computational Linguistics, Baltimore, Maryland, pages 43–48. http://www.aclweb.org/anthology/W14-1705. Cited on page 40.

Adriane Boyd. 2018. Using wikipedia edits in low resource grammatical error correction. In *Proceedings of the 2018 EMNLP Workshop W-NUT: The 4th Workshop on Noisy User-generated Text*. Association for Computational Linguistics, pages 79–84. http://aclweb.org/anthology/W18-6111. Cited on pages 75 and 108.

Adriane Boyd and Detmar Meurers. 2011. Data-driven correction of function words in non-native English. In *Proceedings of the 13th European Workshop on Natural Language Generation*. Association for Computational Linguistics, pages 267–269. http://www.aclweb.org/anthology/W11-2844. Cited on page 38.

Adriane Boyd, Marion Zepf, and Detmar Meurers. 2012. Informing determiner and preposition error correction with hierarchical word clustering. In *Proceedings of the Seventh Workshop on Building Educational Applications Using NLP*. Association for Computational Linguistics, Montréal, Canada, pages 208–215. http://www.aclweb.org/anthology/W12-2024. Cited on page 38.

Ted Briscoe, John Carroll, and Rebecca Watson. 2006. The second release of the RASP system. In *Proceedings of the COLING/ACL on Interactive Presentation Sessions*. Association for Computational Linguistics, Stroudsburg, PA, USA, COLING-ACL '06, pages 77–80. https://doi.org/10.3115/1225403.1225423. Cited on page 26.

Chris Brockett, William B. Dolan, and Michael Gamon. 2006. Correcting ESL errors using phrasal SMT techniques. In *Proceedings of the 21st International Conference on Computational Linguistics and 44th Annual Meeting of the Association for Computational Linguistics*. Association for Computational Linguistics, pages 249–256. http://www.aclweb.org/anthology/P06-1032. Cited on page 31.

Peter E. Brown, Stephen A. Della Pietra, Vincent J. Della Pietra, and Robert L. Mercer. 1993. The mathematics of statistical machine translation: Parameter estimation. *Computational Linguistics* 19(2). http://www.aclweb.org/anthology/J93-2003. Cited on page 31.

Christopher Bryant and Ted Briscoe. 2018. Language model based grammatical error correction without annotated training data. In *Proceedings of the Thirteenth Workshop on Innovative Use of NLP for Building Educational Applications*. Association for Computational Linguistics, pages 247–253. http://aclweb.org/anthology/W18-0529. Cited on pages 28 and 97.

Christopher Bryant and Mariano Felice. 2016. Issues in preprocessing current datasets for grammatical error correction. Technical Report UCAM-CL-TR-894, University of Cambridge, Computer Laboratory. http://www.cl.cam.ac.uk/techreports/UCAM-CL-TR-894.pdf. Cited on page 45.

Christopher Bryant, Mariano Felice, and Ted Briscoe. 2017. Automatic annotation and evaluation of error types for grammatical error correction. In *Proceedings of the 55th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*. Association for Computational Linguistics, Vancouver, Canada, pages 793–805. http://aclweb.org/anthology/P17-1074. Cited on pages 71 and 87.

Christopher Bryant and Hwee Tou Ng. 2015. How far are we from fully automatic high quality grammatical error correction? In *Proceedings of the 53rd Annual Meeting of the Association for Computational Linguistics and the 7th International Joint Conference on Natural Language Processing (Volume 1: Long Papers)*. Association for Computational Linguistics, Beijing, China, pages 697–707. http://www.aclweb.org/anthology/P15-1068. Cited on pages 24, 33, 68, and 86.

Jack Bryant. 1962. Anticoincidence counting method for standardizing radioactive materials. *The International Journal of Applied Radiation and Isotopes* 13(6):273–276. https://doi.org/10.1016/0020-708X(62)90104-7. Cited on page 7.

Jack Bryant. 1963. Coincidence counting corrections for dead-time loss and accidental coincidences. *The International Journal of Applied Radiation and Isotopes* 14(3):143–151. https://doi.org/10.1016/0020-708X(63)90109-1. Cited on page 7.

Jack Bryant, David G. Jones, and Archie McNair. 1967. Development of a 4π liquid scintillation counting method for the absolute standardization of radioactive solutions with particular reference to efficiency tracing measurements. In *Proceedings of a Symposium on Standardization of Radionuclides*. International Atomic Energy Agency (IAEA), pages 47–56. http://www.iaea.org/inis/collection/NCLCollectionStore/_Public/44/072/44072915.pdf?r=1. Cited on page 7.

Christian Buck, Kenneth Heafield, and Bas van Ooyen. 2014. N-gram counts and language models from the Common Crawl. In *Proceedings of the Ninth International Conference on Language Resources and Evaluation (LREC-2014)*. European Language Resources Association (ELRA). http://www.aclweb.org/anthology/L14-1074. Cited on page 27.

Flora Ramfrez Bustamante and Fernando Sanchez Leon. 1996. Gramcheck: A grammar and style checker. In *COLING 1996 Volume 1: The 16th International Conference on Computational Linguistics*. http://www.aclweb.org/anthology/C96-1031. Cited on page 27.

Jan Buys and Brink van der Merwe. 2013. A tree transducer model for grammatical error correction. In *Proceedings of the Seventeenth Conference on Computational Natural Language Learning: Shared Task*. Association for Computational Linguistics, pages 43–51. http://www.aclweb.org/anthology/W13-3606. Cited on pages 39 and 40.

Aoife Cahill, Nitin Madnani, Joel Tetreault, and Diane Napolitano. 2013. Robust systems for preposition error correction using wikipedia revisions. In *Proceedings of the 2013 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies*. Association for Computational Linguistics, pages 507–517. http://www.aclweb.org/anthology/N13-1055. Cited on page 29.

Alice Y. W. Chan. 2010. Toward a taxonomy of written errors: Investigation into the written errors of Hong Kong Cantonese ESL learners. *TESOL Quarterly* 44(2):295–319. https://doi.org/10.5054/tq.2010.219941. Cited on page 71.

Jim Chang, Jiancheng Wu, and Jason S. Chang. 2013. Detecting english grammatical errors based on machine translation. In *Proceedings of the 25th Conference on Computational Linguistics and Speech Processing (ROCLING 2013)*. The Association for Computational Linguistics and Chinese Language Processing (ACLCLP), pages 56–58. http://www.aclweb.org/anthology/O13-1006. Cited on page 31.

Ciprian Chelba, Tomas Mikolov, Mike Schuster, Qi Ge, Thorsten Brants, Phillipp Koehn, and Tony Robinson. 2014. One billion word benchmark for measuring progress in statistical language modeling. In *INTERSPEECH 2014, 15th Annual Conference of the International Speech Communication Association, Singapore, September 14-18, 2014*. pages 2635–2639.

http://www.isca-speech.org/archive/interspeech_2014/i14_2635.html. Cited on pages 28 and 101.

Kyunghyun Cho, Bart van Merrienboer, Caglar Gulcehre, Dzmitry Bahdanau, Fethi Bougares, Holger Schwenk, and Yoshua Bengio. 2014. Learning phrase representations using RNN encoder–decoder for statistical machine translation. In *Proceedings of the 2014 Conference on Empirical Methods in Natural Language Processing (EMNLP)*. Association for Computational Linguistics, pages 1724–1734. https://doi.org/10.3115/v1/D14-1179. Cited on pages 31 and 32.

Martin Chodorow, Markus Dickinson, Ross Israel, and Joel Tetreault. 2012. Problems in evaluating grammatical error detection systems. In *Proceedings of COLING 2012*. The COLING 2012 Organizing Committee, pages 611–628. http://www.aclweb.org/anthology/C12-1038. Cited on page 35.

Martin Chodorow, Joel Tetreault, and Na-Rae Han. 2007. Detection of grammatical errors involving prepositions. In *Proceedings of the Fourth ACL-SIGSEM Workshop on Prepositions*. Association for Computational Linguistics, pages 25–30. http://www.aclweb.org/anthology/W07-1604. Cited on page 29.

Shamil Chollampatt, Duc Tam Hoang, and Hwee Tou Ng. 2016a. Adapting grammatical error correction based on the native language of writers with neural network joint models. In *Proceedings of the 2016 Conference on Empirical Methods in Natural Language Processing*. Association for Computational Linguistics, pages 1901–1911. https://doi.org/10.18653/v1/D16-1195. Cited on pages 31, 41, 42, and 43.

Shamil Chollampatt and Hwee Tou Ng. 2017. Connecting the dots: Towards human-level grammatical error correction. In *Proceedings of the 12th Workshop on Innovative Use of NLP for Building Educational Applications*. Association for Computational Linguistics, pages 327–333. http://aclweb.org/anthology/W17-5037. Cited on pages 31, 41, 42, and 43.

Shamil Chollampatt and Hwee Tou Ng. 2018a. A multilayer convolutional encoder-decoder neural network for grammatical error correction. In *Proceedings of the Thirty-Second AAAI Conference on Artificial Intelligence*. Cited on pages 31, 42, 43, and 104.

Shamil Chollampatt and Hwee Tou Ng. 2018b. A reassessment of reference-based grammatical error correction metrics. In *Proceedings of the 27th International Conference on Computational Linguistics*. Association for Computational Linguistics, pages 2730–2741. http://aclweb.org/anthology/C18-1231. Cited on pages 36 and 37.

Shamil Chollampatt, Kaveh Taghipour, and Hwee Tou Ng. 2016b. Neural network translation models for grammatical error correction. In *Proceedings of the Twenty-Fifth International*

Joint Conference on Artificial Intelligence, IJCAI 2016, New York, NY, USA, 9-15 July 2016. pages 2768–2774. http://www.ijcai.org/Abstract/16/393. Cited on pages 31, 41, and 42.

Leshem Choshen and Omri Abend. 2018a. Automatic metric validation for grammatical error correction. In *Proceedings of the 56th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*. Association for Computational Linguistics, pages 1372–1382. http://aclweb.org/anthology/P18-1127. Cited on page 37.

Leshem Choshen and Omri Abend. 2018b. Reference-less measure of faithfulness for grammatical error correction. In *Proceedings of the 2018 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, Volume 2 (Short Papers)*. Association for Computational Linguistics, pages 124–129. http://aclweb.org/anthology/N18-2020. Cited on page 33.

Corinna Cortes and Vladimir Vapnik. 1995. Support-vector networks. *Machine Learning* 20(3):273–297. https://doi.org/10.1007/BF00994018. Cited on page 29.

Daniel Dahlmeier and Hwee Tou Ng. 2011. Grammatical error correction with alternating structure optimization. In *Proceedings of the 49th Annual Meeting of the Association for Computational Linguistics: Human Language Technologies*. Association for Computational Linguistics, pages 915–923. http://www.aclweb.org/anthology/P11-1092. Cited on page 29.

Daniel Dahlmeier and Hwee Tou Ng. 2012a. A beam-search decoder for grammatical error correction. In *Proceedings of the 2012 Joint Conference on Empirical Methods in Natural Language Processing and Computational Natural Language Learning*. Association for Computational Linguistics, pages 568–578. http://www.aclweb.org/anthology/D12-1052. Cited on pages 31, 98, 99, and 103.

Daniel Dahlmeier and Hwee Tou Ng. 2012b. Better evaluation for grammatical error correction. In *Proceedings of the 2012 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies*. Association for Computational Linguistics, pages 568–572. http://www.aclweb.org/anthology/N12-1067. Cited on pages 33 and 34.

Daniel Dahlmeier, Hwee Tou Ng, and Eric Jun Feng Ng. 2012. NUS at the HOO 2012 shared task. In *Proceedings of the Seventh Workshop on Building Educational Applications Using NLP*. Association for Computational Linguistics, Montréal, Canada, pages 216–224. http://www.aclweb.org/anthology/W12-2025. Cited on pages 38 and 39.

Daniel Dahlmeier, Hwee Tou Ng, and Thanh Phu Tran. 2011. NUS at the HOO 2011 pilot shared task. In *Proceedings of the 13th European Workshop on Natural Language Generation*.

Association for Computational Linguistics, pages 257–259. http://www.aclweb.org/anthology/W11-2841. Cited on page 38.

Daniel Dahlmeier, Hwee Tou Ng, and Siew Mei Wu. 2013. Building a large annotated corpus of learner English: The NUS corpus of learner english. In *Proceedings of the Eighth Workshop on Innovative Use of NLP for Building Educational Applications*. Association for Computational Linguistics, Atlanta, Georgia, pages 22–31. http://www.aclweb.org/anthology/W13-1703. Cited on page 23.

Robert Dale, Ilya Anisimoff, and George Narroway. 2012. HOO 2012: A report on the preposition and determiner error correction shared task. In *Proceedings of the Seventh Workshop on Building Educational Applications Using NLP*. Association for Computational Linguistics, pages 54–62. http://www.aclweb.org/anthology/W12-2006. Cited on pages 15 and 38.

Robert Dale and Adam Kilgarriff. 2011. Helping Our Own: The HOO 2011 pilot shared task. In *Proceedings of the 13th European Workshop on Natural Language Generation*. Association for Computational Linguistics, pages 242–249. http://www.aclweb.org/anthology/W11-2838. Cited on pages 15 and 37.

Fred J. Damerau. 1964. A technique for computer detection and correction of spelling errors. *Communications of the ACM* 7(3):171–176. https://doi.org/10.1145/363958.363994. Cited on page 59.

Saadiyah Darus and Kaladevi Subramaniam. 2009. Error analysis of the written english essays of secondary school students in Malaysia: A case study. *European Journal of Social Sciences* 8(3):483–495. Cited on page 71.

Vidas Daudaravicius. 2012. VTEX determiner and preposition correction system for the HOO 2012 shared task. In *Proceedings of the Seventh Workshop on Building Educational Applications Using NLP*. Association for Computational Linguistics, Montréal, Canada, pages 225–232. http://www.aclweb.org/anthology/W12-2026. Cited on pages 38 and 39.

Rachele De Felice. 2008. *Automatic error detection in non-native English*. Ph.D. thesis, Oxford University, UK. Cited on page 29.

Rachele De Felice and Stephen Pulman. 2007. Automatically acquiring models of preposition use. In *Proceedings of the Fourth ACL-SIGSEM Workshop on Prepositions*. Association for Computational Linguistics, pages 45–50. http://www.aclweb.org/anthology/W07-1607. Cited on page 29.

Rachele De Felice and Stephen G. Pulman. 2008. A classifier-based approach to preposition and determiner error correction in L2 English. In *Proceedings of the 22nd International Conference*

on Computational Linguistics (COLING 2008). COLING 2008 Organizing Committee, pages 169–176. http://www.aclweb.org/anthology/C08-1022. Cited on page 29.

A. P. Dempster, N. M. Laird, and D. B. Rubin. 1977. Maximum likelihood from incomplete data via the EM algorithm. *Journal of the Royal Statistical Society. Series B (Methodological)* 39(1):1–38. http://www.jstor.org/stable/2984875. Cited on page 30.

Marina Dodigovic. 2007. Artificial intelligence and second language learning: An efficient approach to error remediation. *Language Awareness* 16(2):99–113. https://doi.org/10.2167/la416.0. Cited on page 26.

Rickard Domeij, Ola Knutsson, Johan Carlberger, and Viggo Kann. 2000. Granska–an efficient hybrid system for swedish grammar checking. In *Proceedings of the 12th Nordic Conference of Computational Linguistics (NODALIDA 1999)*. Department of Linguistics, Norwegian University of Science and Technology, Norway, pages 49–56. http://www.aclweb.org/anthology/W99-1005. Cited on page 27.

Bradley Efron and Robert J. Tibshirani. 1993. *An Introduction to the Bootstrap*. Number 57 in Monographs on Statistics and Applied Probability. Chapman & Hall/CRC, Boca Raton, Florida, USA. Cited on page 88.

Nava Ehsan and Heshaam Faili. 2010. Towards grammar checker development for Persian language. In *Proceedings of the 6th International Conference on Natural Language Processing and Knowledge Engineering (NLPKE-2010)*. pages 1–8. https://doi.org/10.1109/NLPKE.2010.5587839. Cited on page 27.

Mariano Felice. 2016. *Artificial error generation for translation-based grammatical error correction*. Ph.D. thesis, University of Cambridge, Computer Laboratory. http://www.cl.cam.ac.uk/techreports/UCAM-CL-TR-895.pdf. Cited on page 19.

Mariano Felice and Ted Briscoe. 2015. Towards a standard evaluation method for grammatical error detection and correction. In *Proceedings of the 2015 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies*. Association for Computational Linguistics, pages 578–587. https://doi.org/10.3115/v1/N15-1060. Cited on pages 15, 35, and 89.

Mariano Felice, Christopher Bryant, and Ted Briscoe. 2016. Automatic extraction of learner errors in ESL sentences using linguistically enhanced alignments. In *Proceedings of COLING 2016, the 26th International Conference on Computational Linguistics: Technical Papers*. The COLING 2016 Organizing Committee, Osaka, Japan, pages 825–835. http://aclweb.org/anthology/C16-1079. Cited on pages 57 and 64.

Mariano Felice and Zheng Yuan. 2014. Generating artificial errors for grammatical error correction. In *Proceedings of the Student Research Workshop at the 14th Conference of the European Chapter of the Association for Computational Linguistics*. Association for Computational Linguistics, pages 116–126. https://doi.org/10.3115/v1/E14-3013. Cited on page 19.

Mariano Felice, Zheng Yuan, Øistein E. Andersen, Helen Yannakoudakis, and Ekaterina Kochmar. 2014. Grammatical error correction using hybrid systems and type filtering. In *Proceedings of the Eighteenth Conference on Computational Natural Language Learning: Shared Task*. Association for Computational Linguistics, Baltimore, Maryland, pages 15–24. http://www.aclweb.org/anthology/W14-1702. Cited on pages 40, 41, and 91.

Dan Flickinger and Jiye Yu. 2013. Toward more precision in correction of grammatical errors. In *Proceedings of the Seventeenth Conference on Computational Natural Language Learning: Shared Task*. Association for Computational Linguistics, pages 68–73. http://www.aclweb.org/anthology/W13-3609. Cited on page 40.

Jennifer Foster and Oistein Andersen. 2009. GenERRate: Generating errors for use in grammatical error detection. In *Proceedings of the Fourth Workshop on Innovative Use of NLP for Building Educational Applications*. Association for Computational Linguistics, pages 82–90. http://www.aclweb.org/anthology/W09-2112. Cited on page 19.

Jennifer Foster, Joachim Wagner, and Josef van Genabith. 2008. Adapting a wsj-trained parser to grammatically noisy text. In *Proceedings of ACL-08: HLT, Short Papers*. Association for Computational Linguistics, pages 221–224. http://aclweb.org/anthology/P08-2056. Cited on page 75.

Yoav Freund and Robert E. Schapire. 1999. Large margin classification using the perceptron algorithm. *Machine Learning* 37(3):277–296. https://doi.org/10.1023/A:1007662407062. Cited on page 29.

Michael Gamon, Jianfeng Gao, Chris Brockett, Alexandre Klementiev, William B. Dolan, Dmitriy Belenko, and Lucy Vanderwende. 2008. Using contextual speller techniques and language modeling for ESL error correction. In *Proceedings of the Third International Joint Conference on Natural Language Processing: Volume-I*. http://www.aclweb.org/anthology/I08-1059. Cited on pages 28 and 29.

Tao Ge, Furu Wei, and Ming Zhou. 2018a. Fluency boost learning and inference for neural grammatical error correction. In *Proceedings of the 56th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*. Association for Computational Linguistics, pages 1055–1065. http://aclweb.org/anthology/P18-1097. Cited on pages 25, 31, 42, and 43.

Tao Ge, Furu Wei, and Ming Zhou. 2018b. Reaching human-level performance in automatic grammatical error correction: An empirical study. *CoRR* abs/1807.01270. http://arxiv.org/abs/1807.01270. Cited on page 19.

Mandeep Singh Gill and Gurpreet Singh Lehal. 2008. A grammar checking system for Punjabi. In *COLING 2008: Companion volume: Demonstrations*. COLING 2008 Organizing Committee, pages 149–152. http://www.aclweb.org/anthology/C08-3002. Cited on page 27.

Anju Giri. 2011. Errors in the use of english grammar. *Journal of NELTA* 15(1-2):54–63. https://www.nepjol.info/index.php/NELTA/article/view/4610. Cited on page 71.

Édouard Grave, Armand Joulin, Moustapha Cissé, David Grangier, and Hervé Jégou. 2017. Efficient softmax approximation for GPUs. In Doina Precup and Yee Whye Teh, editors, *Proceedings of the 34th International Conference on Machine Learning*. PMLR, International Convention Centre, Sydney, Australia, volume 70 of *Proceedings of Machine Learning Research*, pages 1302–1310. http://proceedings.mlr.press/v70/grave17a.html. Cited on page 101.

Roman Grundkiewicz and Marcin Junczys-Dowmunt. 2018. Near human-level performance in grammatical error correction with hybrid machine translation. In *Proceedings of the 2018 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, Volume 2 (Short Papers)*. Association for Computational Linguistics, pages 284–290. http://aclweb.org/anthology/N18-2046. Cited on pages 31, 32, 42, 43, 104, and 105.

Roman Grundkiewicz, Marcin Junczys-Dowmunt, and Edward Gillian. 2015. Human evaluation of grammatical error correction systems. In *Proceedings of the 2015 Conference on Empirical Methods in Natural Language Processing*. Association for Computational Linguistics, pages 461–470. https://doi.org/10.18653/v1/D15-1052. Cited on pages 33, 36, and 37.

Anubhav Gupta. 2014. Grammatical error detection using tagger disagreement. In *Proceedings of the Eighteenth Conference on Computational Natural Language Learning: Shared Task*. Association for Computational Linguistics, Baltimore, Maryland, pages 49–52. http://www.aclweb.org/anthology/W14-1706. Cited on pages 40 and 90.

Na-Rae Han, Martin Chodorow, and Claudia Leacock. 2006. Detecting errors in English article usage by non-native speakers. *Nat. Lang. Eng.* 12(2):115–129. https://doi.org/10.1017/S1351324906004190. Cited on page 29.

Kenneth Heafield. 2011. KenLM: faster and smaller language model queries. In *Proceedings of the EMNLP 2011 Sixth Workshop on Statistical Machine Translation*. Edinburgh, Scotland, United Kingdom, pages 187–197. https://kheafield.com/papers/avenue/kenlm.pdf. Cited on page 101.

Michael Heilman, Aoife Cahill, Nitin Madnani, Melissa Lopez, Matthew Mulholland, and Joel Tetreault. 2014. Predicting grammaticality on an ordinal scale. In *Proceedings of the 52nd Annual Meeting of the Association for Computational Linguistics (Volume 2: Short Papers)*. Association for Computational Linguistics, pages 174–180. https://doi.org/10.3115/v1/P14-2029. Cited on page 25.

Michael Heilman, Aoife Cahill, and Joel Tetreault. 2012. Precision isn't everything: A hybrid approach to grammatical error detection. In *Proceedings of the Seventh Workshop on Building Educational Applications Using NLP*. Association for Computational Linguistics, pages 233–241. http://www.aclweb.org/anthology/W12-2027. Cited on pages 28 and 38.

S. David Hernandez and Hiram Calvo. 2014. CoNLL 2014 shared task: Grammatical error correction with a syntactic n-gram language model from a big corpora. In *Proceedings of the Eighteenth Conference on Computational Natural Language Learning: Shared Task*. Association for Computational Linguistics, Baltimore, Maryland, pages 53–59. http://www.aclweb.org/anthology/W14-1707. Cited on page 40.

Duc Tam Hoang, Shamil Chollampatt, and Hwee Tou Ng. 2016. Exploiting n-best hypotheses to improve an SMT approach to grammatical error correction. In *Proceedings of the Twenty-Fifth International Joint Conference on Artificial Intelligence*. AAAI Press, IJCAI'16, pages 2803–2809. http://dl.acm.org/citation.cfm?id=3060832.3061013. Cited on pages 31, 41, and 42.

Sepp Hochreiter and Jürgen Schmidhuber. 1997. Long short-term memory. *Neural Computation* 9(8):1735–1780. https://doi.org/10.1162/neco.1997.9.8.1735. Cited on page 32.

Matthew Honnibal and Mark Johnson. 2015. An improved non-monotonic transition system for dependency parsing. In *Proceedings of the 2015 Conference on Empirical Methods in Natural Language Processing*. Association for Computational Linguistics, Lisbon, Portugal, pages 1373–1378. https://aclweb.org/anthology/D/D15/D15-1162. Cited on pages 26 and 50.

Paul Howson. 2013. The English effect. Technical report, British Council. https://www.britishcouncil.org/sites/default/files/english-effect-report-v2.pdf. Cited on page 15.

Elitza Ivanova, Delphine Bernhard, and Cyril Grouin. 2011. Handling outlandish occurrences: Using rules and lexicons for correcting NLP articles. In *Proceedings of the 13th European Workshop on Natural Language Generation*. Association for Computational Linguistics, pages 254–256. http://www.aclweb.org/anthology/W11-2840. Cited on page 38.

Emi Izumi, Kiyotaka Uchimoto, and Hitoshi Isahara. 2005. Error annotation for corpus of Japanese learner English. In *Proceedings of the Sixth International Workshop on Linguistically*

Interpreted Corpora (LINC-2005). http://www.aclweb.org/anthology/I05-6009. Cited on page 71.

Jianshu Ji, Qinlong Wang, Kristina Toutanova, Yongen Gong, Steven Truong, and Jianfeng Gao. 2017. A nested attention neural hybrid model for grammatical error correction. In *Proceedings of the 55th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*. Association for Computational Linguistics, pages 753–762. https://doi.org/10.18653/v1/P17-1070. Cited on pages 31 and 42.

Zhongye Jia, Peilu Wang, and Hai Zhao. 2013. Grammatical error correction as multiclass classification with single model. In *Proceedings of the Seventeenth Conference on Computational Natural Language Learning: Shared Task*. Association for Computational Linguistics, pages 74–81. http://www.aclweb.org/anthology/W13-3610. Cited on page 40.

Marcin Junczys-Dowmunt and Roman Grundkiewicz. 2014. The AMU system in the CoNLL-2014 shared task: Grammatical error correction by data-intensive and feature-rich statistical machine translation. In *Proceedings of the Eighteenth Conference on Computational Natural Language Learning: Shared Task*. Association for Computational Linguistics, Baltimore, Maryland, pages 25–33. http://www.aclweb.org/anthology/W14-1703. Cited on pages 40, 41, 90, and 104.

Marcin Junczys-Dowmunt and Roman Grundkiewicz. 2016. Phrase-based machine translation is state-of-the-art for automatic grammatical error correction. In *Proceedings of the 2016 Conference on Empirical Methods in Natural Language Processing*. Association for Computational Linguistics, pages 1546–1556. https://doi.org/10.18653/v1/D16-1161. Cited on pages 25, 31, 41, and 42.

Marcin Junczys-Dowmunt, Roman Grundkiewicz, Shubha Guha, and Kenneth Heafield. 2018. Approaching neural grammatical error correction as a low-resource machine translation task. In *Proceedings of the 2018 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, Volume 1 (Long Papers)*. Association for Computational Linguistics, pages 595–606. http://aclweb.org/anthology/N18-1055. Cited on pages 31, 32, 42, and 43.

Daniel Jurafsky and James H. Martin. 2009. *Speech and Language Processing (2Nd Edition)*. Prentice-Hall, Inc., Upper Saddle River, NJ, USA. Cited on page 28.

Ting-hui Kao, Yu-wei Chang, Hsun-wen Chiu, Tzu-Hsi Yen, Joanne Boisson, Jian-cheng Wu, and Jason S. Chang. 2013. CoNLL-2013 shared task: Grammatical error correction NTHU system description. In *Proceedings of the Seventeenth Conference on Computational Natural Language Learning: Shared Task*. Association for Computational Linguistics, pages 20–25. http://www.aclweb.org/anthology/W13-3603. Cited on pages 39, 40, 97, and 98.

Inga Kempfert and Christine Köhn. 2018. An automatic error tagger for german. In *Proceedings of the 7th workshop on NLP for Computer Assisted Language Learning*. LiU Electronic Press, Stockholm, Sweden, pages 32–40. https://www.aclweb.org/anthology/W18-7104. Cited on pages 75 and 108.

Ekaterina Kochmar, Øistein Andersen, and Ted Briscoe. 2012. HOO 2012 error recognition and correction shared task: Cambridge university submission report. In *Proceedings of the Seventh Workshop on Building Educational Applications Using NLP*. Association for Computational Linguistics, Montréal, Canada, pages 242–250. http://www.aclweb.org/anthology/W12-2028. Cited on pages 38 and 39.

Philipp Koehn. 2010. *Statistical Machine Translation*. Cambridge University Press, New York, NY, USA, 1st edition. Cited on page 31.

Philipp Koehn, Franz J. Och, and Daniel Marcu. 2003. Statistical phrase-based translation. In *Proceedings of the 2003 Human Language Technology Conference of the North American Chapter of the Association for Computational Linguistics*. http://www.aclweb.org/anthology/N03-1017. Cited on page 31.

Anoop Kunchukuttan, Sriram Chaudhury, and Pushpak Bhattacharyya. 2014. Tuning a grammar correction system for increased precision. In *Proceedings of the Eighteenth Conference on Computational Natural Language Learning: Shared Task*. Association for Computational Linguistics, Baltimore, Maryland, pages 60–64. http://www.aclweb.org/anthology/W14-1708. Cited on page 40.

Anoop Kunchukuttan, Ritesh Shah, and Pushpak Bhattacharyya. 2013. IITB system for CoNLL 2013 shared task: A hybrid approach to grammatical error correction. In *Proceedings of the Seventeenth Conference on Computational Natural Language Learning: Shared Task*. Association for Computational Linguistics, pages 82–87. http://www.aclweb.org/anthology/W13-3611. Cited on page 40.

John D. Lafferty, Andrew McCallum, and Fernando C. N. Pereira. 2001. Conditional random fields: Probabilistic models for segmenting and labeling sequence data. In *Proceedings of the Eighteenth International Conference on Machine Learning*. Morgan Kaufmann Publishers Inc., San Francisco, CA, USA, ICML '01, pages 282–289. http://dl.acm.org/citation.cfm?id=645530.655813. Cited on page 29.

Jieun Lee, Jung-Tae Lee, and Hae-Chang Rim. 2012. Korea university system in the HOO 2012 shared task. In *Proceedings of the Seventh Workshop on Building Educational Applications Using NLP*. Association for Computational Linguistics, Montréal, Canada, pages 251–256. http://www.aclweb.org/anthology/W12-2029. Cited on pages 38 and 39.

John Lee and Stephanie Seneff. 2008. Correcting misuse of verb forms. In *Proceedings of ACL-08: HLT*. Association for Computational Linguistics, pages 174–182. http://www.aclweb.org/anthology/P08-1021. Cited on page 29.

Kyusong Lee and Gary Geunbae Lee. 2014. POSTECH grammatical error correction system in the CoNLL-2014 shared task. In *Proceedings of the Eighteenth Conference on Computational Natural Language Learning: Shared Task*. Association for Computational Linguistics, Baltimore, Maryland, pages 65–73. http://www.aclweb.org/anthology/W14-1709. Cited on pages 40, 41, and 104.

Lung-Hao Lee, Gaoqi Rao, Liang-Chih Yu, Endong Xun, Baolin Zhang, and Li-Ping Chang. 2016. Overview of NLP-TEA 2016 shared task for Chinese grammatical error diagnosis. In *Proceedings of the 3rd Workshop on Natural Language Processing Techniques for Educational Applications (NLPTEA2016)*. The COLING 2016 Organizing Committee, pages 40–48. http://aclweb.org/anthology/W16-4906. Cited on page 37.

Lung-Hao Lee, Liang-Chih Yu, and Li-Ping Chang. 2015. Overview of the NLP-TEA 2015 shared task for Chinese grammatical error diagnosis. In *Proceedings of the 2nd Workshop on Natural Language Processing Techniques for Educational Applications*. Association for Computational Linguistics, pages 1–6. https://doi.org/10.18653/v1/W15-4401. Cited on page 37.

Lung-Hao Lee, Liang-Chih Yu, Kuei-Ching Lee, Yuen-Hsien Tseng, Li-Ping Chang, and Hsin-Hsi Chen. 2014. A sentence judgment system for grammatical error detection. In *Proceedings of COLING 2014, the 25th International Conference on Computational Linguistics: System Demonstrations*. Dublin City University and Association for Computational Linguistics, pages 67–70. http://www.aclweb.org/anthology/C14-2015. Cited on pages 28, 97, 98, and 99.

Vladimir I. Levenshtein. 1966. Binary Codes Capable of Correcting Deletions, Insertions and Reversals. *Soviet Physics Doklady* 10:707–710. Cited on pages 34 and 57.

Chuan-Jie Lin and Shao-Heng Chen. 2018. Detecting grammatical errors in the NTOU CGED system by identifying frequent subsentences. In *Proceedings of the 5th Workshop on Natural Language Processing Techniques for Educational Applications*. Association for Computational Linguistics, pages 203–206. http://aclweb.org/anthology/W18-3730. Cited on page 28.

Gerard Lynch, Erwan Moreau, and Carl Vogel. 2012. A Naive Bayes classifier for automatic correction of preposition and determiner errors in ESL text. In *Proceedings of the Seventh Workshop on Building Educational Applications Using NLP*. Association for Computational Linguistics, Montréal, Canada, pages 257–262. http://www.aclweb.org/anthology/W12-2030. Cited on pages 38 and 39.

Nitin Madnani, Joel Tetreault, and Martin Chodorow. 2012. Exploring grammatical error correction with not-so-crummy machine translation. In *Proceedings of the Seventh Workshop on Building Educational Applications Using NLP*. Association for Computational Linguistics, pages 44–53. http://www.aclweb.org/anthology/W12-2005. Cited on page 31.

Laxmi Maharjan. 2010. Learners' errors and their evaluation. *Journal of NELTA* 14(1):71–81. https://www.nepjol.info/index.php/NELTA/article/view/3093. Cited on page 71.

Christopher D. Manning, Mihai Surdeanu, John Bauer, Jenny Finkel, Steven J. Bethard, and David McClosky. 2014. The Stanford CoreNLP natural language processing toolkit. In *Association for Computational Linguistics (ACL) System Demonstrations*. pages 55–60. http://www.aclweb.org/anthology/P/P14/P14-5010. Cited on page 26.

Mitchell P. Marcus, Mary Ann Marcinkiewicz, and Beatrice Santorini. 1993. Building a large annotated corpus of english: The penn treebank. *Comput. Linguist.* 19(2):313–330. http://dl.acm.org/citation.cfm?id=972470.972475. Cited on page 26.

Kathleen F. McCoy, Christopher A. Pennington, and Linda Z. Suri. 1996. English error correction: A syntactic user model based on principled Mal-Rule scoring. In *Proceedings of the Fifth International Conference on User Modeling*. Inc, pages 59–66. Cited on page 26.

Lisa N. Michaud, Kathleen F. McCoy, and Christopher A. Pennington. 2000. An intelligent tutoring system for deaf learners of written English. In *Proceedings of the Fourth International ACM Conference on Assistive Technologies*. ACM, New York, NY, USA, Assets '00, pages 92–100. https://doi.org/10.1145/354324.354348. Cited on page 26.

Tomoya Mizumoto, Yuta Hayashibe, Mamoru Komachi, Masaaki Nagata, and Yuji Matsumoto. 2012. The effect of learner corpus size in grammatical error correction of ESL writings. In *Proceedings of COLING 2012: Posters*. The COLING 2012 Organizing Committee, pages 863–872. http://www.aclweb.org/anthology/C12-2084. Cited on pages 24 and 31.

Tomoya Mizumoto, Mamoru Komachi, Masaaki Nagata, and Yuji Matsumoto. 2011. Mining revision log of language learning SNS for automated Japanese error correction of second language learners. In *Proceedings of 5th International Joint Conference on Natural Language Processing*. Asian Federation of Natural Language Processing, pages 147–155. http://www.aclweb.org/anthology/I11-1017. Cited on page 24.

Tomoya Mizumoto and Yuji Matsumoto. 2016. Discriminative reranking for grammatical error correction with statistical machine translation. In *Proceedings of the 2016 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies*. Association for Computational Linguistics, pages 1133–1138. https://doi.org/10.18653/v1/N16-1133. Cited on pages 31, 41, and 42.

Behrang Mohit, Alla Rozovskaya, Nizar Habash, Wajdi Zaghouani, and Ossama Obeid. 2014. The first QALB shared task on automatic text correction for Arabic. In *Proceedings of the EMNLP 2014 Workshop on Arabic Natural Language Processing (ANLP)*. Association for Computational Linguistics, pages 39–47. https://doi.org/10.3115/v1/W14-3605. Cited on page 37.

Ryo Nagata, Edward Whittaker, and Vera Sheinman. 2011. Creating a manually error-tagged and shallow-parsed learner corpus. In *Proceedings of the 49th Annual Meeting of the Association for Computational Linguistics: Human Language Technologies*. Association for Computational Linguistics, Portland, Oregon, USA, pages 1210–1219. http://www.aclweb.org/anthology/P11-1121. Cited on pages 71 and 72.

Courtney Napoles, Aoife Cahill, and Nitin Madnani. 2016a. The effect of multiple grammatical errors on processing non-native writing. In *Proceedings of the 11th Workshop on Innovative Use of NLP for Building Educational Applications*. Association for Computational Linguistics, San Diego, CA, pages 1–11. http://www.aclweb.org/anthology/W16-0501. Cited on page 75.

Courtney Napoles and Chris Callison-Burch. 2017. Systematically adapting machine translation for grammatical error correction. In *Proceedings of the 12th Workshop on Innovative Use of NLP for Building Educational Applications*. Association for Computational Linguistics, pages 345–356. http://aclweb.org/anthology/W17-5039. Cited on page 31.

Courtney Napoles, Keisuke Sakaguchi, Matt Post, and Joel Tetreault. 2015. Ground truth for grammatical error correction metrics. In *Proceedings of the 53rd Annual Meeting of the Association for Computational Linguistics and the 7th International Joint Conference on Natural Language Processing (Volume 2: Short Papers)*. Association for Computational Linguistics, pages 588–593. https://doi.org/10.3115/v1/P15-2097. Cited on pages 15, 36, 37, and 89.

Courtney Napoles, Keisuke Sakaguchi, Matt Post, and Joel Tetreault. 2016b. GLEU without tuning. *eprint arXiv:1605.02592 [cs.CL]* http://arxiv.org/abs/1605.02592. Cited on pages 36 and 37.

Courtney Napoles, Keisuke Sakaguchi, and Joel Tetreault. 2016c. There's no comparison: Reference-less evaluation metrics in grammatical error correction. In *Proceedings of the 2016 Conference on Empirical Methods in Natural Language Processing*. Association for Computational Linguistics, pages 2109–2115. https://doi.org/10.18653/v1/D16-1228. Cited on pages 33 and 36.

Courtney Napoles, Keisuke Sakaguchi, and Joel Tetreault. 2017. JFLEG: a fluency corpus and benchmark for grammatical error correction. In *Proceedings of the 15th Conference of*

the European Chapter of the Association for Computational Linguistics: Volume 2, Short Papers. Association for Computational Linguistics, Valencia, Spain, pages 229–234. http://www.aclweb.org/anthology/E17-2037. Cited on pages 25 and 101.

Hwee Tou Ng, Siew Mei Wu, Ted Briscoe, Christian Hadiwinoto, Raymond Hendy Susanto, and Christopher Bryant. 2014. The CoNLL-2014 shared task on grammatical error correction. In *Proceedings of the Eighteenth Conference on Computational Natural Language Learning: Shared Task*. Association for Computational Linguistics, pages 1–14. https://doi.org/10.3115/v1/W14-1701. Cited on pages 15, 24, 40, and 101.

Hwee Tou Ng, Siew Mei Wu, Yuanbin Wu, Christian Hadiwinoto, and Joel Tetreault. 2013. The CoNLL-2013 shared task on grammatical error correction. In *Proceedings of the Seventeenth Conference on Computational Natural Language Learning: Shared Task*. Association for Computational Linguistics, pages 1–12. http://www.aclweb.org/anthology/W13-3601. Cited on pages 15, 24, 39, and 101.

Diane Nicholls. 2003. The cambridge learner corpus: Error coding and analysis for lexicography and ELT. In *Proceedings of the Corpus Linguistics 2003 conference*. pages 572–581. Cited on page 20.

Joakim Nivre, Marie-Catherine de Marneffe, Filip Ginter, Yoav Goldberg, Jan Hajic, Christopher D. Manning, Ryan McDonald, Slav Petrov, Sampo Pyysalo, Natalia Silveira, Reut Tsarfaty, and Daniel Zeman. 2016. Universal dependencies v1: A multilingual treebank collection. In Nicoletta Calzolari (Conference Chair), Khalid Choukri, Thierry Declerck, Sara Goggi, Marko Grobelnik, Bente Maegaard, Joseph Mariani, Helene Mazo, Asuncion Moreno, Jan Odijk, and Stelios Piperidis, editors, *Proceedings of the Tenth International Conference on Language Resources and Evaluation (LREC 2016)*. European Language Resources Association (ELRA), Paris, France. Cited on page 73.

Franz Josef Och and Hermann Ney. 2003. A systematic comparison of various statistical alignment models. *Computational Linguistics* 29(1). http://www.aclweb.org/anthology/J03-1002. Cited on page 57.

Kishore Papineni, Salim Roukos, Todd Ward, and Wei-Jing Zhu. 2002. BLEU: a method for automatic evaluation of machine translation. In *Proceedings of the 40th Annual Meeting of the Association for Computational Linguistics*. http://www.aclweb.org/anthology/P02-1040. Cited on page 36.

Jong C. Park, Martha Palmer, and Gay Washburn. 1997. An English grammar checker as a writing aid for students of English as a second language. In *Proceedings of the Fifth Conference on Applied Natural Language Processing: Descriptions of System Demonstrations and Videos*.

Association for Computational Linguistics, Stroudsburg, PA, USA, ANLC '97, pages 24–24. https://doi.org/10.3115/974281.974296. Cited on page 26.

Jeff Pasternack and Dan Roth. 2008. The wikipedia corpus. Technical report, University of Illinois. http://cogcomp.org/papers/PasternackRo08.pdf. Cited on page 27.

Slav Petrov, Dipanjan Das, and Ryan McDonald. 2012. A universal part-of-speech tagset. In *Proceedings of the Eighth International Conference on Language Resources and Evaluation (LREC-2012)*. European Language Resources Association (ELRA), Istanbul, Turkey, pages 2089–2096. http://www.lrec-conf.org/proceedings/lrec2012/pdf/274_Paper.pdf. Cited on page 26.

Desmond Darma Putra and Lili Szabo. 2013. UdS at CoNLL 2013 shared task. In *Proceedings of the Seventeenth Conference on Computational Natural Language Learning: Shared Task*. Association for Computational Linguistics, pages 88–95. http://www.aclweb.org/anthology/W13-3612. Cited on page 40.

Li Quan, Oleksandr Kolomiyets, and Marie-Francine Moens. 2012. KU Leuven at HOO-2012: A hybrid approach to detection and correction of determiner and preposition errors in non-native English text. In *Proceedings of the Seventh Workshop on Building Educational Applications Using NLP*. Association for Computational Linguistics, Montréal, Canada, pages 263–271. http://www.aclweb.org/anthology/W12-2031. Cited on pages 38 and 39.

John Ross Quinlan. 1986. Induction of decision trees. *Machine Learning* 1(1):81–106. https://doi.org/10.1007/BF00116251. Cited on page 29.

Justus J. Randolph. 2005. Free-marginal multirater kappa: An alternative to Fleiss' fixed-marginal multirater kappa. *Joensuu University Learning and Instruction Symposium* http://files.eric.ed.gov/fulltext/ED490661.pdf. Cited on page 85.

Gaoqi Rao, Qi Gong, Baolin Zhang, and Endong Xun. 2018. Overview of NLPTEA-2018 share task Chinese grammatical error diagnosis. In *Proceedings of the 5th Workshop on Natural Language Processing Techniques for Educational Applications*. Association for Computational Linguistics, pages 42–51. http://aclweb.org/anthology/W18-3706. Cited on page 37.

Gaoqi Rao, Baolin Zhang, Endong XUN, and Lung-Hao Lee. 2017. IJCNLP-2017 task 1: Chinese grammatical error diagnosis. In *Proceedings of the IJCNLP 2017, Shared Tasks*. Asian Federation of Natural Language Processing, pages 1–8. http://aclweb.org/anthology/I17-4001. Cited on page 37.

Adwait Ratnaparkhi. 1996. A maximum entropy model for part-of-speech tagging. In *Conference on Empirical Methods in Natural Language Processing*. http://www.aclweb.org/anthology/W96-0213. Cited on page 29.

Marek Rei, Mariano Felice, Zheng Yuan, and Ted Briscoe. 2017. Artificial error generation with machine translation and syntactic patterns. In *Proceedings of the 12th Workshop on Innovative Use of NLP for Building Educational Applications*. Association for Computational Linguistics, pages 287–292. http://aclweb.org/anthology/W17-5032. Cited on page 19.

Marek Rei and Helen Yannakoudakis. 2016. Compositional sequence labeling models for error detection in learner writing. In *Proceedings of the 54th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*. Association for Computational Linguistics, pages 1181–1191. https://doi.org/10.18653/v1/P16-1112. Cited on pages 41 and 46.

Alla Rozovskaya, Houda Bouamor, Nizar Habash, Wajdi Zaghouani, Ossama Obeid, and Behrang Mohit. 2015. The second QALB shared task on automatic text correction for Arabic. In *Proceedings of the Second Workshop on Arabic Natural Language Processing*. Association for Computational Linguistics, pages 26–35. https://doi.org/10.18653/v1/W15-3204. Cited on page 37.

Alla Rozovskaya, Kai-Wei Chang, Mark Sammons, and Dan Roth. 2013. The university of Illinois system in the CoNLL-2013 shared task. In *Proceedings of the Seventeenth Conference on Computational Natural Language Learning: Shared Task*. Association for Computational Linguistics, pages 13–19. http://www.aclweb.org/anthology/W13-3602. Cited on pages 39 and 40.

Alla Rozovskaya, Kai-Wei Chang, Mark Sammons, Dan Roth, and Nizar Habash. 2014a. The Illinois-Columbia system in the CoNLL-2014 shared task. In *Proceedings of the Eighteenth Conference on Computational Natural Language Learning: Shared Task*. Association for Computational Linguistics, Baltimore, Maryland, pages 34–42. http://www.aclweb.org/anthology/W14-1704. Cited on pages 40, 41, and 90.

Alla Rozovskaya and Dan Roth. 2010a. Generating confusion sets for context-sensitive error correction. In *Proceedings of the 2010 Conference on Empirical Methods in Natural Language Processing*. Association for Computational Linguistics, pages 961–970. http://www.aclweb.org/anthology/D10-1094. Cited on page 29.

Alla Rozovskaya and Dan Roth. 2010b. Training paradigms for correcting errors in grammar and usage. In *Human Language Technologies: The 2010 Annual Conference of the North American Chapter of the Association for Computational Linguistics*. Association for Computational Linguistics, pages 154–162. http://www.aclweb.org/anthology/N10-1018. Cited on page 29.

Alla Rozovskaya and Dan Roth. 2013. Joint learning and inference for grammatical error correction. In *Proceedings of the 2013 Conference on Empirical Methods in Natural Language*

Processing. Association for Computational Linguistics, pages 791–802. http://www.aclweb.org/anthology/D13-1074. Cited on page 29.

Alla Rozovskaya and Dan Roth. 2014. Building a state-of-the-art grammatical error correction system. *Transactions of the Association for Computational Linguistics* 2:419–434. http://aclweb.org/anthology/Q14-1033. Cited on page 29.

Alla Rozovskaya and Dan Roth. 2016. Grammatical error correction: Machine translation and classifiers. In *Proceedings of the 54th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*. Association for Computational Linguistics, pages 2205–2215. https://doi.org/10.18653/v1/P16-1208. Cited on pages 31, 41, 42, and 91.

Alla Rozovskaya, Dan Roth, and Vivek Srikumar. 2014b. Correcting grammatical verb errors. In *Proceedings of the 14th Conference of the European Chapter of the Association for Computational Linguistics*. Association for Computational Linguistics, pages 358–367. https://doi.org/10.3115/v1/E14-1038. Cited on page 29.

Alla Rozovskaya, Mark Sammons, Joshua Gioja, and Dan Roth. 2011. University of Illinois system in HOO text correction shared task. In *Proceedings of the 13th European Workshop on Natural Language Generation*. Association for Computational Linguistics, pages 263–266. http://www.aclweb.org/anthology/W11-2843. Cited on page 38.

Alla Rozovskaya, Mark Sammons, and Dan Roth. 2012. The UI system in the HOO 2012 shared task on error correction. In *Proceedings of the Seventh Workshop on Building Educational Applications Using NLP*. Association for Computational Linguistics, Montréal, Canada, pages 272–280. http://www.aclweb.org/anthology/W12-2032. Cited on pages 38 and 39.

Keisuke Sakaguchi, Yuta Hayashibe, Shuhei Kondo, Lis Kanashiro, Tomoya Mizumoto, Mamoru Komachi, and Yuji Matsumoto. 2012a. NAIST at the HOO 2012 shared task. In *Proceedings of the Seventh Workshop on Building Educational Applications Using NLP*. Association for Computational Linguistics, Montréal, Canada, pages 281–288. http://www.aclweb.org/anthology/W12-2033. Cited on pages 38 and 39.

Keisuke Sakaguchi, Tomoya Mizumoto, Mamoru Komachi, and Yuji Matsumoto. 2012b. Joint English spelling error correction and POS tagging for language learners writing. In *Proceedings of COLING 2012*. The COLING 2012 Organizing Committee, pages 2357–2374. http://www.aclweb.org/anthology/C12-1144. Cited on page 75.

Keisuke Sakaguchi, Courtney Napoles, Matt Post, and Joel Tetreault. 2016. Reassessing the goals of grammatical error correction: Fluency instead of grammaticality. *Transactions of the Association for Computational Linguistics* 4:169–182. http://aclweb.org/anthology/Q16-1013. Cited on pages 24, 25, 36, 58, 95, and 105.

Keisuke Sakaguchi, Matt Post, and Benjamin Van Durme. 2017a. Error-repair dependency parsing for ungrammatical texts. In *Proceedings of the 55th Annual Meeting of the Association for Computational Linguistics (Volume 2: Short Papers)*. Association for Computational Linguistics, pages 189–195. https://doi.org/10.18653/v1/P17-2030. Cited on pages 37 and 75.

Keisuke Sakaguchi, Matt Post, and Benjamin Van Durme. 2017b. Grammatical error correction with neural reinforcement learning. In *Proceedings of the Eighth International Joint Conference on Natural Language Processing (Volume 2: Short Papers)*. Asian Federation of Natural Language Processing, pages 366–372. http://aclweb.org/anthology/I17-2062. Cited on pages 31, 43, and 104.

Allen Schmaltz, Yoon Kim, Alexander Rush, and Stuart Shieber. 2017. Adapting sequence models for sentence correction. In *Proceedings of the 2017 Conference on Empirical Methods in Natural Language Processing*. Association for Computational Linguistics, pages 2807–2813. http://aclweb.org/anthology/D17-1298. Cited on page 42.

David Schneider and Kathleen F. McCoy. 1998. Recognizing syntactic errors in the writing of second language learners. In *COLING 1998 Volume 2: The 17th International Conference on Computational Linguistics*. http://www.aclweb.org/anthology/C98-2191. Cited on page 26.

Hongsuck Seo, Jonghoon Lee, Seokhwan Kim, Kyusong Lee, Sechun Kang, and Gary Geunbae Lee. 2012. A meta learning approach to grammatical error correction. In *Proceedings of the 50th Annual Meeting of the Association for Computational Linguistics (Volume 2: Short Papers)*. Association for Computational Linguistics, pages 328–332. http://www.aclweb.org/anthology/P12-2064. Cited on page 29.

Claude E. Shannon. 1948. A mathematical theory of communication. *Bell System Technical Journal* 27(3):379–423. https://doi.org/10.1002/j.1538-7305.1948.tb01338.x. Cited on pages 30 and 31.

Itamar Shatz. 2017. Native language influence during second language acquisition: A large-scale learner corpus analysis. In *Proceedings of the Pacific Second Language Research Forum (PacSLRF 2016)*. Japan Second Language Association, pages 175–180. Cited on page 16.

Grigori Sidorov, Anubhav Gupta, Martin Tozer, Dolors Catala, Angels Catena, and Sandrine Fuentes. 2013. Rule-based system for automatic grammar correction using syntactic n-grams for English language learning (L2). In *Proceedings of the Seventeenth Conference on Computational Natural Language Learning: Shared Task*. Association for Computational Linguistics, pages 96–101. http://www.aclweb.org/anthology/W13-3613. Cited on page 40.

Gary F. Simons and Charles D. Fennig. 2018. *Ethnologue: Languages of the World*. SIL International, Dallas, Texas, 21 edition. https://www.ethnologue.com/language/eng. Cited on page 15.

Raymond Hendy Susanto, Peter Phandi, and Hwee Tou Ng. 2014. System combination for grammatical error correction. In *Proceedings of the 2014 Conference on Empirical Methods in Natural Language Processing (EMNLP)*. Association for Computational Linguistics, pages 951–962. https://doi.org/10.3115/v1/D14-1102. Cited on pages 31, 41, and 42.

Ilya Sutskever, Oriol Vinyals, and Quoc V Le. 2014. Sequence to sequence learning with neural networks. In Z. Ghahramani, M. Welling, C. Cortes, N. D. Lawrence, and K. Q. Weinberger, editors, *Advances in Neural Information Processing Systems 27*, Curran Associates, Inc., pages 3104–3112. http://papers.nips.cc/paper/5346-sequence-to-sequence-learning-with-neural-networks.pdf. Cited on page 32.

Ben Swanson and Elif Yamangil. 2012. Correction detection and error type selection as an ESL educational aid. In *Proceedings of the 2012 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies*. Association for Computational Linguistics, pages 357–361. http://www.aclweb.org/anthology/N12-1037. Cited on pages 58, 67, and 68.

Toshikazu Tajiri, Mamoru Komachi, and Yuji Matsumoto. 2012. Tense and aspect error correction for ESL learners using global context. In *Proceedings of the 50th Annual Meeting of the Association for Computational Linguistics (Volume 2: Short Papers)*. Association for Computational Linguistics, pages 198–202. http://www.aclweb.org/anthology/P12-2039. Cited on pages 24, 29, and 54.

Joel Tetreault, Jennifer Foster, and Martin Chodorow. 2010. Using parse features for preposition selection and error detection. In *Proceedings of the ACL 2010 Conference Short Papers*. Association for Computational Linguistics, pages 353–358. http://www.aclweb.org/anthology/P10-2065. Cited on page 29.

Joel R. Tetreault and Martin Chodorow. 2008. The ups and downs of preposition error detection in ESL writing. In *Proceedings of the 22nd International Conference on Computational Linguistics (Coling 2008)*. Coling 2008 Organizing Committee, pages 865–872. http://www.aclweb.org/anthology/C08-1109. Cited on page 29.

Jenine Turner and Eugene Charniak. 2007. Language modeling for determiner selection. In *Human Language Technologies 2007: The Conference of the North American Chapter of the Association for Computational Linguistics; Companion Volume, Short Papers*. Association for Computational Linguistics, pages 177–180. http://www.aclweb.org/anthology/N07-2045. Cited on page 28.

Antal van den Bosch and Peter Berck. 2012. Memory-based text correction for preposition and determiner errors. In *Proceedings of the Seventh Workshop on Building Educational Applications Using NLP*. Association for Computational Linguistics, Montréal, Canada, pages 289–294. http://www.aclweb.org/anthology/W12-2034. Cited on pages 38 and 39.

Antal van den Bosch and Peter Berck. 2013. Memory-based grammatical error correction. In *Proceedings of the Seventeenth Conference on Computational Natural Language Learning: Shared Task*. Association for Computational Linguistics, pages 102–108. http://www.aclweb.org/anthology/W13-3614. Cited on page 40.

Cornelis J. van Rijsbergen. 1979. *Information Retrieval*. Butterworth-Heinemann, Newton, MA, USA, 2nd edition. Cited on page 33.

Joachim Wagner and Jennifer Foster. 2009. The effect of correcting grammatical errors on parse probabilities. In *Proceedings of the 11th International Conference on Parsing Technologies (IWPT'09)*. Association for Computational Linguistics, pages 176–179. http://aclweb.org/anthology/W09-3827. Cited on page 75.

Peilu Wang, Zhongye Jia, and Hai Zhao. 2014a. Grammatical error detection and correction using a single maximum entropy model. In *Proceedings of the Eighteenth Conference on Computational Natural Language Learning: Shared Task*. Association for Computational Linguistics, Baltimore, Maryland, pages 74–82. http://www.aclweb.org/anthology/W14-1710. Cited on page 40.

Yiming Wang, Longyue Wang, Xiaodong Zeng, Derek F. Wong, Lidia S. Chao, and Yi Lu. 2014b. Factored statistical machine translation for grammatical error correction. In *Proceedings of the Eighteenth Conference on Computational Natural Language Learning: Shared Task*. Association for Computational Linguistics, Baltimore, Maryland, pages 83–90. http://www.aclweb.org/anthology/W14-1711. Cited on page 40.

Ralph M. Weischedel, Wilfried M. Voge, and Mark James. 1978. An artificial intelligence approach to language instruction. *Artificial Intelligence* 10(3):225 – 240. https://doi.org/10.1016/S0004-3702(78)80015-0. Cited on page 27.

L. Amber Wilcox-O'Hearn. 2013. A noisy channel model framework for grammatical correction. In *Proceedings of the Seventeenth Conference on Computational Natural Language Learning: Shared Task*. Association for Computational Linguistics, pages 109–114. http://www.aclweb.org/anthology/W13-3615. Cited on pages 39 and 40.

Jian-cheng Wu, Jim Chang, and Jason S. Chang. 2013. Correcting serial grammatical errors based on n-grams and syntax. In *International Journal of Computational Linguistics & Chinese Language Processing, Volume 18, Number 4, December 2013-Special Issue on Selected Papers from ROCLING XXV*. http://www.aclweb.org/anthology/O13-5003. Cited on page 31.

Jian-Cheng Wu, Joseph Chang, Yi-Chun Chen, Shih-Ting Huang, Mei-Hua Chen, and Jason S. Chang. 2012. Helping Our Own: NTHU NLPLAB system description. In *Proceedings of the Seventh Workshop on Building Educational Applications Using NLP*. Association for Computational Linguistics, Montréal, Canada, pages 295–301. http://www.aclweb.org/anthology/W12-2035. Cited on page 38.

Jian-Cheng Wu, Tzu-Hsi Yen, Jim Chang, Guan-Cheng Huang, Jimmy Chang, Hsiang-Ling Hsu, Yu-Wei Chang, and Jason S. Chang. 2014. NTHU at the CoNLL-2014 shared task. In *Proceedings of the Eighteenth Conference on Computational Natural Language Learning: Shared Task*. Association for Computational Linguistics, Baltimore, Maryland, pages 91–95. http://www.aclweb.org/anthology/W14-1712. Cited on page 40.

Xiupeng Wu, Peijie Huang, Jundong Wang, Qingwen Guo, Yuhong Xu, and Chuping Chen. 2015. Chinese grammatical error diagnosis system based on hybrid model. In *Proceedings of the 2nd Workshop on Natural Language Processing Techniques for Educational Applications*. Association for Computational Linguistics, pages 117–125. https://doi.org/10.18653/v1/W15-4418. Cited on page 27.

Yang Xiang, Bo Yuan, Yaoyun Zhang, Xiaolong Wang, Wen Zheng, and Chongqiang Wei. 2013. A hybrid model for grammatical error correction. In *Proceedings of the Seventeenth Conference on Computational Natural Language Learning: Shared Task*. Association for Computational Linguistics, pages 115–122. http://www.aclweb.org/anthology/W13-3616. Cited on pages 29 and 40.

Ziang Xie, Anand Avati, Naveen Arivazhagan, Dan Jurafsky, and Andrew Y. Ng. 2016. Neural language correction with character-based attention. *CoRR* abs/1603.09727. http://arxiv.org/abs/1603.09727. Cited on pages 31 and 42.

Junwen Xing, Longyue Wang, Derek F. Wong, Lidia S. Chao, and Xiaodong Zeng. 2013. UM-Checker: a hybrid system for English grammatical error correction. In *Proceedings of the Seventeenth Conference on Computational Natural Language Learning: Shared Task*. Association for Computational Linguistics, pages 34–42. http://www.aclweb.org/anthology/W13-3605. Cited on page 40.

Huichao Xue and Rebecca Hwa. 2014. Improved correction detection in revised ESL sentences. In *Proceedings of the 52nd Annual Meeting of the Association for Computational Linguistics (Volume 2: Short Papers)*. Association for Computational Linguistics, pages 599–604. https://doi.org/10.3115/v1/P14-2098. Cited on pages 58, 59, 66, 67, 68, 70, and 71.

Helen Yannakoudakis, Ted Briscoe, and Ben Medlock. 2011. A new dataset and method for automatically grading ESOL texts. In *Proceedings of the 49th Annual Meeting of the Association for Computational Linguistics: Human Language Technologies*. Association for

Computational Linguistics, Portland, Oregon, USA, pages 180–189. http://www.aclweb.org/anthology/P11-1019. Cited on pages 22, 46, and 101.

Helen Yannakoudakis, Marek Rei, Øistein E. Andersen, and Zheng Yuan. 2017. Neural sequence-labelling models for grammatical error correction. In *Proceedings of the 2017 Conference on Empirical Methods in Natural Language Processing*. Association for Computational Linguistics, pages 2795–2806. http://aclweb.org/anthology/D17-1297. Cited on pages 31, 41, 42, 43, and 104.

Jui-Feng Yeh, Li-Ting Chang, Chan-Yi Liu, and Tsung-Wei Hsu. 2017. Chinese spelling check based on n-gram and string matching algorithm. In *Proceedings of the 4th Workshop on Natural Language Processing Techniques for Educational Applications (NLPTEA 2017)*. Asian Federation of Natural Language Processing, pages 35–38. http://aclweb.org/anthology/W17-5906. Cited on page 28.

Bong-Jun Yi, Ho-Chang Lee, and Hae-Chang Rim. 2013. KUNLP grammatical error correction system for CoNLL-2013 shared task. In *Proceedings of the Seventeenth Conference on Computational Natural Language Learning: Shared Task*. Association for Computational Linguistics, pages 123–127. http://www.aclweb.org/anthology/W13-3617. Cited on page 40.

Ippei Yoshimoto, Tomoya Kose, Kensuke Mitsuzawa, Keisuke Sakaguchi, Tomoya Mizumoto, Yuta Hayashibe, Mamoru Komachi, and Yuji Matsumoto. 2013. NAIST at 2013 CoNLL grammatical error correction shared task. In *Proceedings of the Seventeenth Conference on Computational Natural Language Learning: Shared Task*. Association for Computational Linguistics, pages 26–33. http://www.aclweb.org/anthology/W13-3604. Cited on pages 39 and 40.

Liang-Chih Yu, Lung-Hao Lee, and Liping Chang. 2014. Overview of grammatical error diagnosis for learning Chinese as a foreign language. In *Workshop Proceedings of the 22nd International Conference on Computers in Education, ICCE 2014*. pages 42–47. Cited on page 37.

Zheng Yuan and Ted Briscoe. 2016. Grammatical error correction using neural machine translation. In *Proceedings of the 2016 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies*. Association for Computational Linguistics, pages 380–386. https://doi.org/10.18653/v1/N16-1042. Cited on pages 31, 41, 42, and 43.

Zheng Yuan, Ted Briscoe, and Mariano Felice. 2016. Candidate re-ranking for SMT-based grammatical error correction. In *Proceedings of the 11th Workshop on Innovative Use of*

NLP for Building Educational Applications. Association for Computational Linguistics, pages 256–266. https://doi.org/10.18653/v1/W16-0530. Cited on pages 31, 41, 42, and 43.

Zheng Yuan and Mariano Felice. 2013. Constrained grammatical error correction using statistical machine translation. In *Proceedings of the Seventeenth Conference on Computational Natural Language Learning: Shared Task*. Association for Computational Linguistics, pages 52–61. http://www.aclweb.org/anthology/W13-3607. Cited on pages 39 and 40.

Torsten Zesch. 2011. Helping Our Own 2011: UKP Lab system description. In *Proceedings of the 13th European Workshop on Natural Language Generation*. Association for Computational Linguistics, pages 260–262. http://www.aclweb.org/anthology/W11-2842. Cited on page 38.

Torsten Zesch and Jens Haase. 2012. HOO 2012 shared task: UKP Lab system description. In *Proceedings of the Seventh Workshop on Building Educational Applications Using NLP*. Association for Computational Linguistics, Montréal, Canada, pages 302–306. http://www.aclweb.org/anthology/W12-2036. Cited on pages 38 and 39.

Longkai Zhang and Houfeng Wang. 2014. A unified framework for grammar error correction. In *Proceedings of the Eighteenth Conference on Computational Natural Language Learning: Shared Task*. Association for Computational Linguistics, Baltimore, Maryland, pages 96–102. http://www.aclweb.org/anthology/W14-1713. Cited on page 40.

Jianbo Zhao, Hao Liu, Zuyi Bao, Xiaopeng Bai, Si Li, and Zhiqing Lin. 2017. N-gram model for Chinese grammatical error diagnosis. In *Proceedings of the 4th Workshop on Natural Language Processing Techniques for Educational Applications (NLPTEA 2017)*. Asian Federation of Natural Language Processing, pages 39–44. http://aclweb.org/anthology/W17-5907. Cited on page 28.

APPENDIX A

ERROR TYPE COUNTS FOR CONLL-2014 ANALYSIS

	AMU			CAMB			CUUI			IITB		
Type	**TP**	**FP**	**FN**	**TP**	**FP**	**FN**	**TP**	**FP**	**FN**	**TP**	**FP**	**FN**
Missing	58	74	354	132	153	317	77	215	355	2	11	343
Replacement	428	722	1,158	475	796	1,207	381	449	1,271	20	47	1,317
Unnecessary	0	0	407	125	366	329	157	305	312	6	7	379

	IPN			NTHU			PKU			POST		
Type	**TP**	**FP**	**FN**	**TP**	**FP**	**FN**	**TP**	**FP**	**FN**	**TP**	**FP**	**FN**
Missing	1	33	343	46	87	361	16	32	357	52	114	349
Replacement	53	484	1,314	299	784	1,253	279	663	1,241	312	629	1,298
Unnecessary	0	2	384	64	125	341	0	1	391	155	434	311

	RAC			SJTU			UFC			UMC		
Type	**TP**	**FP**	**FN**	**TP**	**FP**	**FN**	**TP**	**FP**	**FN**	**TP**	**FP**	**FN**
Missing	1	64	374	15	9	326	0	0	342	99	148	325
Replacement	326	779	1,232	47	46	1,325	36	14	1,322	143	269	1,326
Unnecessary	0	5	403	46	209	346	0	0	376	73	367	353

Table A.1: True positive, false positive and false negative counts for each team in CoNLL-2014 in terms of missing, replacement and unnecessary edits. The total number of edits may vary for each system as this depends on the individual references that are chosen during evaluation. These results were used to make Table 6.2.

		AMU	CAMB	CUUI	IITB	IPN	NTHU	PKU	POST	RAC	SJTU	UFC	UMC
ADJ	**TP**	3	5	0	0	0	0	2	0	1	0	0	0
	FP	39	52	0	3	1	3	1	4	7	7	0	22
	FN	30	34	32	25	26	28	28	35	30	23	22	28
ADJ:FORM	**TP**	5	6	3	2	0	2	1	1	1	0	0	3
	FP	4	2	1	0	1	6	0	1	23	0	0	1
	FN	3	4	6	3	5	6	6	6	4	5	5	2
ADV	**TP**	1	9	0	0	0	0	0	0	0	1	0	5
	FP	15	69	1	1	2	2	1	1	4	20	0	53
	FN	30	32	32	32	30	32	34	37	36	31	32	34
CONJ	**TP**	1	0	0	0	0	0	0	0	0	0	0	0
	FP	15	18	0	0	1	1	0	0	0	6	0	26
	FN	13	16	13	11	12	13	12	14	12	13	12	13
CONTR	**TP**	7	2	6	0	0	0	0	4	0	2	0	2
	FP	17	3	7	0	1	0	0	8	1	1	0	5
	FN	0	4	1	5	5	5	5	3	5	3	5	4
DET	**TP**	52	179	230	3	0	106	27	210	0	43	0	88
	FP	103	316	519	11	13	192	67	596	6	55	0	156
	FN	315	238	214	324	322	267	316	212	342	297	324	284
MORPH	**TP**	45	42	19	4	2	16	25	9	17	2	4	13
	FP	37	30	19	10	88	45	103	25	37	1	7	18
	FN	45	45	68	69	69	61	58	68	61	71	74	73
NOUN	**TP**	14	23	0	2	2	0	0	2	3	0	0	10
	FP	53	67	3	6	44	12	28	16	16	17	0	25
	FN	100	98	109	88	88	102	104	100	99	93	91	93
NOUN:INFL	**TP**	6	5	5	0	1	2	5	4	4	0	0	0
	FP	4	4	4	0	3	0	2	2	2	1	0	0
	FN	1	3	2	6	5	4	3	3	2	6	6	6
NOUN:NUM	**TP**	128	121	141	7	22	100	97	136	78	19	0	31
	FP	132	156	178	10	130	126	234	305	191	16	0	38
	FN	98	102	95	172	171	108	129	104	134	164	176	150
NOUN:POSS	**TP**	3	2	0	0	0	0	1	0	0	1	0	1
	FP	12	1	0	0	0	0	6	0	38	3	0	1
	FN	17	16	20	18	19	21	17	21	20	20	19	18
ORTH	**TP**	3	14	31	0	1	0	2	36	28	0	0	5
	FP	2	7	11	0	28	1	5	37	140	0	0	5
	FN	24	20	21	21	21	26	27	20	29	24	21	24
OTHER	**TP**	22	35	3	0	1	2	2	4	0	0	0	11
	FP	92	117	26	7	41	138	10	37	46	11	0	82
	FN	318	334	352	323	321	342	340	350	344	322	324	332
PART	**TP**	5	4	0	0	0	4	0	0	0	2	0	2
	FP	2	12	3	0	0	28	0	0	0	3	0	11
	FN	22	23	23	20	21	20	23	21	20	19	17	18
PREP	**TP**	40	93	35	3	0	31	0	5	0	16	0	30
	FP	44	129	68	1	2	252	0	18	3	24	0	118
	FN	205	167	221	208	209	213	220	226	218	196	209	203
PRON	**TP**	7	11	0	0	1	2	1	3	1	0	0	11
	FP	9	43	1	5	9	2	0	7	20	22	0	38
	FN	63	68	60	55	56	66	62	59	63	59	59	63
PUNCT	**TP**	5	26	16	2	0	13	0	9	0	1	0	37
	FP	15	19	27	0	15	16	0	24	36	19	0	49
	FN	135	139	135	105	112	129	121	132	130	101	108	121
SPELL	**TP**	60	38	0	0	3	0	72	70	91	0	0	1
	FP	18	11	1	1	9	2	91	32	32	0	0	0
	FN	34	53	74	68	68	74	29	28	15	70	70	72
VERB	**TP**	13	16	0	0	1	0	1	0	0	0	0	10
	FP	58	75	0	6	13	11	7	4	6	17	0	44
	FN	151	147	171	139	142	158	147	170	156	146	137	144
VERB:FORM	**TP**	22	24	22	0	5	35	33	3	33	4	0	13
	FP	40	43	11	1	51	61	72	10	60	10	0	32
	FN	75	75	71	89	84	62	60	87	67	84	84	77
VERB:INFL	**TP**	2	2	0	0	1	1	1	1	2	0	0	0
	FP	0	0	0	0	1	0	1	0	0	0	1	0
	FN	0	0	2	2	1	1	1	1	0	2	2	2
VERB:SVA	**TP**	27	37	99	1	13	92	19	18	32	13	32	16
	FP	27	46	76	1	40	87	11	31	57	9	6	10
	FN	73	79	40	87	82	47	78	82	72	81	75	86
VERB:TENSE	**TP**	15	31	5	2	1	3	6	4	36	4	0	19
	FP	58	84	3	1	26	11	57	17	122	22	0	40
	FN	155	142	162	157	160	159	157	167	138	156	157	144
WO	**TP**	0	7	0	2	0	0	0	0	0	0	0	7
	FP	0	11	10	1	0	0	0	2	1	0	0	10
	FN	12	14	14	12	12	11	12	12	12	11	11	13

Table A.2: True positive, false positive and false negative counts for each team in CoNLL-2014 for the main error types. The total number of edits may vary for each system as this depends on the individual references that are chosen during evaluation. These results were used to make Table 6.3.

APPENDIX B

TUNED ERROR TYPE THRESHOLDS

Setting	Threshold	CoNLL-2013	FCE-dev	JFLEG-dev
Global	All	0.02	0.05	0.06
Operation	R	0.03	0.05	0.06
	U	0.00	0.02	0.06
Main	ADJ	0.04	0.05	0.06
	ADJ:FORM	0.03	0.07	0.06
	ADV	0.03	0.05	0.06
	CONJ	0.02	0.05	0.06
	CONTR	0.02	0.06	0.06
	DET	0.02	0.06	0.06
	MORPH	0.03	0.04	0.07
	NOUN	0.02	0.06	0.06
	NOUN:INFL	0.04	0.04	0.05
	NOUN:NUM	0.00	0.06	0.05
	NOUN:POSS	0.02	0.05	0.06
	ORTH	0.02	0.07	0.05
	OTHER	0.04	0.06	0.06
	PART	0.04	0.05	0.06
	PREP	0.04	0.05	0.08
	PRON	0.02	0.05	0.06
	PUNCT	0.02	0.05	0.06
	SPELL	0.05	0.02	0.04
	VERB	0.02	0.05	0.06
	VERB:FORM	0.03	0.04	0.08
	VERB:INFL	0.02	0.05	0.06
	VERB:SVA	0.03	0.03	0.07
	VERB:TENSE	0.05	0.10	0.09
	WO	0.02	0.05	0.06

Table B.1: Optimum tuned error type thresholds for three development sets in the global, operation and main error type setting. These values were used to make the results in Table 7.3.

Setting	Threshold	CoNLL-2013	FCE-dev	JFLEG-dev
Detailed	M:ADJ	0.02	0.05	0.06
	M:ADV	0.02	0.05	0.06
	M:CONJ	0.02	0.05	0.06
	M:CONTR	0.02	0.05	0.06
	M:DET	0.02	0.05	0.06
	M:NOUN	0.02	0.05	0.06
	M:NOUN:POSS	0.02	0.05	0.06
	M:OTHER	0.02	0.05	0.06
	M:PART	0.02	0.05	0.06
	M:PREP	0.02	0.05	0.06
	M:PRON	0.02	0.05	0.06
	M:PUNCT	0.02	0.05	0.06
	M:VERB	0.02	0.05	0.06
	M:VERB:FORM	0.02	0.05	0.06
	M:VERB:TENSE	0.02	0.05	0.06
	R:ADJ	0.04	0.05	0.06
	R:ADJ:FORM	0.03	0.07	0.06
	R:ADV	0.03	0.05	0.06
	R:CONJ	0.02	0.05	0.06
	R:CONTR	0.02	0.06	0.06
	R:DET	0.03	0.06	0.06
	R:MORPH	0.03	0.04	0.07
	R:NOUN	0.02	0.06	0.06
	R:NOUN:INFL	0.04	0.04	0.05
	R:NOUN:NUM	0.00	0.06	0.05
	R:NOUN:POSS	0.02	0.05	0.06
	R:ORTH	0.02	0.07	0.05
	R:OTHER	0.04	0.06	0.06
	R:PART	0.04	0.05	0.06
	R:PREP	0.04	0.05	0.08
	R:PRON	0.02	0.05	0.06
	R:PUNCT	0.02	0.05	0.06
	R:SPELL	0.05	0.02	0.04
	R:VERB	0.02	0.05	0.06
	R:VERB:FORM	0.03	0.04	0.08
	R:VERB:INFL	0.02	0.05	0.06
	R:VERB:SVA	0.03	0.03	0.07
	R:VERB:TENSE	0.05	0.10	0.07
	R:WO	0.02	0.05	0.06
	U:ADJ	0.02	0.05	0.06
	U:ADV	0.02	0.05	0.06
	U:CONJ	0.02	0.05	0.06
	U:CONTR	0.02	0.05	0.06
	U:DET	-0.01	0.05	0.06
	U:NOUN	0.02	0.05	0.06
	U:NOUN:POSS	0.02	0.05	0.06
	U:OTHER	0.02	0.05	0.06
	U:PART	0.03	0.05	0.06
	U:PREP	0.01	0.03	0.06
	U:PRON	0.02	0.05	0.06
	U:PUNCT	0.02	0.05	0.06
	U:VERB	0.02	0.05	0.06
	U:VERB:FORM	0.02	0.05	0.06
	U:VERB:TENSE	0.02	0.05	0.06

Table B.2: Optimum tuned error type thresholds for three development sets in the most detailed error type setting. These values were used to make the results in Table 7.3. Thresholds that are equal to the global threshold often indicate error types that are uncorrected by the system.