ROBUST PARSING FOR UNGRAMMATICAL SENTENCES

by

Homa Baradaran Hashemi

B.Sc. in Software Engineering, Iran University of Science and Technology, 2007

M.Sc. in Software Engineering, University of Tehran, 2011

M.Sc. in Intelligent Systems Program, University of Pittsburgh, 2014

Submitted to the Graduate Faculty of

the Kenneth P. Dietrich School of

Arts and Sciences in partial fulfillment

of the requirements for the degree of

Doctor of Philosophy

University of Pittsburgh

2017

UNIVERSITY OF PITTSBURGH

KENNETH P. DIETRICH SCHOOL OF ARTS AND SCIENCES

This dissertation was presented

by

Homa Baradaran Hashemi

It was defended on

October 17th 2017

and approved by

Dr. Rebecca Hwa, Department of Computer Science

Dr. Diane Litman, Department of Computer Science

Dr. Christian Schunn, Department of Psychology

Dr. Na-Rae Han, Department of Linguistics

Dissertation Director: Dr. Rebecca Hwa, Department of Computer Science

ROBUST PARSING FOR UNGRAMMATICAL SENTENCES

Homa Baradaran Hashemi, PhD

University of Pittsburgh, 2017

Natural Language Processing (NLP) is a research area that specializes in studying computational approaches to human language. However, not all of the natural language sentences are grammatically correct. Sentences that are ungrammatical, awkward, or too casual/colloquial tend to appear in a variety of NLP applications, from product reviews and social media analysis to intelligent language tutors or multilingual processing. In this thesis, we focus on syntactic parsing, an essential component of many NLP applications. We investigate the impact of ungrammatical sentences on statistical parsers. We also hypothesize that breaking up parse trees from problematic parts prevents NLP applications from degrading due to incorrect syntactic analysis.

A parser is *robust* if it can overlook problems such as grammar mistakes and produce a parse tree that closely resembles the correct analysis for the intended sentence. We develop a robustness evaluation metric and conduct a series of experiments to compare the performances of state-of-the-art parsers on the ungrammatical sentences. The evaluation results show that ungrammatical sentences present challenges for statistical parsers, because the well-formed syntactic trees they produce may not be appropriate for ungrammatical sentences. We also define a new framework for reviewing the parses of ungrammatical sentences and extracting the coherent parts whose syntactic analyses make sense. We call this task *parse tree fragmentation*. The experimental results suggest that the proposed overall fragmentation framework is a promising way to handle syntactically unusual sentences; they also validate the utility of parse tree fragmentation methods in two external tasks of sentential grammaticality judgment and semantic role labeling.

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

ACKNOWLEDGEMENT

I am immensely grateful to my advisor, Rebecca Hwa, for teaching me the range of things one can possibly learn in graduate school, from doing great research to writing papers. I am thankful for her for occasionally having more faith in me than I had myself (especially we got four rejections in a row). Needless to say, without her guidance and encouragement this work would not be completed.

I would like to express my sincere gratitude to the other members of my dissertation committee, Diane Litman, Na-Rae Han and Christian Schunn for their invaluable advice, especially their thoughtful comment on unifying our ideas on one syntactic representation which vastly improved the quality of this dissertation. I am also grateful to Jan Wiebe for discussions and comments at earlier stages of this research. I thank Azadeh Shakery, my Masters advisor at University of Tehran, under whose guidance I first started doing research in information retrieval and natural language processing. I am also thankful to my mentors during my internships, Silke Witt, Reiner Kraft and Thomas Polzin who helped me to achieve a better view of real world NLP challenges.

I was lucky to be part of ISP, a small and diverse program which presented me with so many perspectives on artificial intelligence. Thanks so much Michele Thomas for being such a wonderful administrator and for making the ISP seminars interesting and fun. I cannot forget Wendy Bergstein, whose warm presence made ISP enjoyable in the first years of our PhD when I was new to the country and culture. My PhD research was supported in part by the National Science Foundation awards IIS-0745914, #1550635 and #1735752, and the University of Pittsburgh Center for Research Computing through the resources provided. I specifically acknowledge the assistance of Ketan Maheshwari.

I am grateful to my friends at ISP, CS and NLP group for making the last six years at Pittsburgh such a memorable experience. Thanks to Huichao Xue, Fan Zhang, Fattaneh Jabbari, ChangSheng Liu, Lingjia Deng, Yoonjung Choi, Huy Nguyen, Wencan Luo, Omid Kashefi, Roya Hosseini, Tazin Afrin and Mahbaneh Eshaghzadeh. Many special thanks to Zahra Rahimi, who has been influential in pretty much every aspect of my graduate life since the first time we spoke at the University of Tehran in 2008. I spent almost the last year of my PhD in Cambridge, MA; a special thanks to Yevgeni Berzak from MIT for introducing the EFCAMDAT dataset and Allen Schmaltz from Harvard for helping in running seq2seq experiments which saved me much time and effort.

My deepest gratitude goes to my parents Narges and Ahmed for their love and unquestioning support throughout. It was through talking to them during my lunch time/their dinner time, that my battery was recharged everyday. I am grateful to my sister Sima and her husband Nima for all the joy they have added to our life by driving to Pittsburgh even for a weekend. I thank my brother Ali for being a role model for me (I followed his path without knowing it myself) and most importantly for introducing Mahdi to our family. I also thank Mahdi's parents Fatemeh and Nemat for their patience and continuous encouragements.

Last, but certainly not least, I would like to thank my dearest Mahdi, my best friend and husband. It was through his selfless friendship and love that shaped my life during this long journey for the last thirteen years.

1.0 INTRODUCTION

1.1 MOTIVATION

Natural Language Processing (NLP) is a research area that focuses on studying computational methods that analyze human languages. The ultimate goal of NLP is to construct systems that understand and produce natural languages as humans do. To advance this goal, various NLP tasks are established, from name entity recognition (e.g. identifying people in a sentence) and part-of-speech tagging (e.g. detecting verbs in a sentence) to more complicated tasks such as question answering and automatic summarization.

The input of NLP applications could come from different domains (note that by domain we mean a collection of texts from certain genres, topics, and styles of expressions); not all of these domains are necessarily grammatically correct. The input text can come from heavily edited domains by humans, such as news, or from unedited domains, such as microblogs, consumer reviews, forums, etc.

Knowing the relationship between individual words in a natural language sentence is essential for an application that attempts to process the input text in some way, such as extracting the information or translating the text to another language. It is the role of the syntactic parsing to produce this kind of analysis. A computational analyzer that assigns syntactic structures to sentences is a parser. Current state-of-the-art parsers employ supervised methods to learn models from annotated training data (treebanks). These treebanks include a collection of sentences with their manually annotated syntactic analysis.

A statistical parser trained on a standard treebank, however, often produces full, syntactically well-formed trees for all the input sentences that might not even be appropriate for the sentences. For example, Figure 1 shows incorrect syntactic analyses for an output of a machine translation

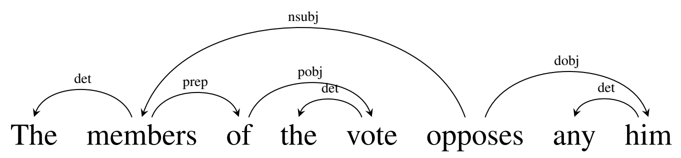


Figure 1: An ungrammatical sentence gets a well-formed but inappropriate parse tree.

system; even though the sentence has several problems such as unusual phrases *members of the vote* and *any him*, the parser still groups them into clauses to serve as the subject and the object of the main verb, respectively. These incorrect analyses may later impact the performances of the downstream NLP applications.

Moreover, as shown in the past, current state-of-the-art statistical parsers perform well on standard (newswire) benchmarks with accuracies above 90%; however, these high accuracies are limited to heavily edited domains, and their analyses for sentences from different domains are less reliable (Gildea, 2001; McClosky et al., 2010; Foster, 2010; Petrov et al., 2010; Foster et al., 2011a). Ungrammatical sentences (or even awkward sentences that are technically grammatical) can be seen as special kinds of out-of-domain sentences; in some cases, it is not even clear whether a complete parse should be given to the sentence.

In this thesis, we study parsing ungrammatical sentences and introduce a framework to generate meaningful syntactic analyses for these sentences based on their grammatical mistakes.

1.2 RESEARCH QUESTIONS

The primary goal of this research is to investigate the impact of ungrammatical sentences on parsers by addressing the following questions:

- In what ways does a parser's performance degrade when dealing with ungrammatical sentences?
- Is it feasible to automatically identify parse tree fragments that are plausible interpretations for

the phrases they cover?

- Do the resulting parse tree fragments provide some useful information for downstream NLP applications?

1.3 THESIS STATEMENT

While parsers have trouble when sentences contain certain types of mistakes (e.g., extra word errors or multiple errors that are close to each other), there are still reliable parts in the parse tree unaffected by the errors. The thesis of this dissertation is that we can identify the unaffected areas of the parse tree and prune the problematic parts, resulting in a set of tree fragments. These tree fragments will contain useful syntactic information that can help downstream applications such as fluency judgment and semantic role labeling.

1.4 THESIS OVERVIEW

In the following, we present a general overview of the key concepts and the steps that we take to address the research questions.

1.4.1 Ungrammatical Sentences

A sentence is considered ungrammatical if all its words are valid in the language, but it still contains grammatical or usage errors (Foster, 2007). As a rule of thumb, ungrammatical sentences require a set of corrections in order to sound "natural" to a native-speaker (Pinker, 2015). In this thesis, we study two written data sources in which the sentences may contain grammatical mistakes: writings of English-as-a-Second language (ESL) students and automatic machine translation (MT) outputs. We primarily focus on written domains that their major goal is to generate fluent and grammatical sentences; in addition, these domains have a wider range of error types, such as missing phrases, insertion of unnecessary phrases, and incorrect phrasal ordering, than spoken domains.

1.4.2 Impact of Ungrammatical Sentences on Parsers

For the purpose of analyzing the impact of ungrammatical sentences on parsers, we need to evaluate their generated parse trees for these problematic sentences. However, previous works on parser evaluation have primarily focused on accuracy and speed of parsers (Choi et al., 2015; Kummerfeld et al., 2012; McDonald and Nivre, 2011; Kong and Smith, 2014), and have not taken ungrammatical sentences into consideration. The main reason is that typically parser evaluation requires manually annotated gold standards (treebanks); while, there does not exist large-scale annotated corpus for ungrammatical sentences. Therefore, to evaluate the parsers for ungrammatical sentences, rather than creating a treebank or adapting the annotation schema for ungrammatical sentences (which may not always be valid (Cahill, 2015; Ragheb and Dickinson, 2012)), we propose an alternative approach to consider the automatically produced parse trees of well-formed sentences as gold standards and compare the parser output for the corresponding problematic sentences against them. We say that a parser is *robust* for ungrammatical sentences, if it can overlook problems such as grammar mistakes and produce a parse tree that closely resembles the correct analysis for the intended sentence.

Evaluating robustness of parsers, however, presents another challenge; since the words of the ungrammatical sentence and its grammatical counterpart do not necessarily match (there might be missing or extra words). Hence, we introduce a modified evaluation metric to compare parse trees of ungrammatical sentences against parse trees for their corresponding grammatical sentences. In the first part of this thesis, we present our proposed *robustness evaluation methodology*, and we compare state-of-the-art dependency parsers to see how much and in what ways they are robust when applied to ungrammatical sentences.

1.4.3 Parse Tree Fragmentation Framework

Our parser robustness analyses show that ungrammatical sentences present challenges for statistical parsers; however, when ignoring the erroneous parts of the sentences, a typical parser does reasonably well on recognizing the syntactic structures of the remaining grammatical parts of the sentences. Therefore, a reasonable approach to parse ungrammatical sentences would be to identify well-formed syntactic structures of those parts of the sentences that do make sense. We establish this idea by proposing a new framework to parse ungrammatical sentences. We call this framework *parse tree fragmentation*, since it re-interprets the parse trees by pruning the implausible syntactic relations. Pruning syntactic relations results a set of tree fragments that are linguistically appropriate for the phrases they cover. Figure 2 shows a set of implausible syntactic relations for the ungrammatical sentence which results in four fragments.

To automatically fragment the parse trees of ungrammatical sentences, we need a sizable tree fragmentation gold standard corpus. Ideally, this corpus would be a collection of trees of ungrammatical sentences and their corresponding sets of tree fragments extracted by knowledgeable annotators who agree with each other. However, such a corpus does not exist. Annotating a corpus of ungrammatical sentences with tree fragments is not also suitable for a large scale human annotation. Because people may not agree on the best set of fragments, and furthermore, the fragmentation decisions may depends on the downstream application that uses the parse trees. Instead, detecting parse tree fragments is significantly easier if a grammatical version of the sentence is also given. Therefore, in this thesis, we propose methods to collect annotations leveraging existing NLP corpora that have ungrammatical sentences and their corrected versions. An example data source of this type is a machine translation evaluation corpus, which consists of machine translated sentences and their corresponding references. Since the intended meaning is known, people can make fragmentation decisions with high inter-annotator agreement. In the most informative case, the fluent paraphrases also come with an explanation for each replacement. An example data source of this type is an English-as-a-Second language learner's corpus, which consists of student sentences and their detailed corrections. Constrained by the location and type of each error, the fragmentation decisions may be made deterministically without a human annotator. Therefore, in this thesis, we create a gold standard corpus by extracting tree fragments from the more informative data sources such as an MT evaluation corpus and an ESL learner's corpus.

By assuming the existence of a gold standard training corpus, we propose three fragmentation strategies to automatically produce parse tree fragments for ungrammatical sentences. In one, we propose a post-hoc process on the outputs of off-the-shelf parsers for the ungrammatical sentences. In the other two, we only make use of the training data to jointly learn to parse and fragment the ungrammatical sentences. The two joint methods are based on a parser retraining method and a sequence-to-sequence labeling method.

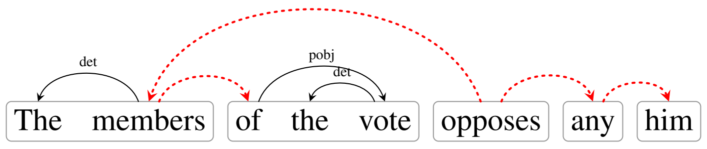


Figure 2: The red dotted dependencies show a set of implausible syntactic relations which results in four fragments.

1.4.4 Applications of Parse Tree Fragmentation

To validate the utility of the parse tree fragmentation, we use it in two downstream NLP tasks which benefit from syntactic parsing:

- *Sentential fluency judgment* task which predicts how "natural" a sentence might sound to a native-speaker human. These predictions can be useful, for instance, to help grading students' writings. An automatic fluency judge uses syntactic analysis to make predictions on the sentence-level.
- *Semantic role labeling (SRL)* task which identifies semantic relations of groups of words with respect to a particular verb in a sentence. The obtained semantic relations can be useful for other NLP tasks such as question answering. The semantic relations are typically extracted on the word-level.

Because the two applications process the sentences at different levels, we would be able to investigate the usefulness of parse tree fragmentation in two distinct applications. We hypothesize the parse tree fragmentation can provide informative signals to help downstream NLP applications. Through a set of empirical studies, we show that our hypothesis holds.

1.5 THESIS CONTRIBUTIONS

This thesis advances the research on parsing ungrammatical sentences in the following ways:

- We have designed a metric and methodology for evaluating the impact of ungrammatical sentences on statistical parsers.
  - We have conducted a quantitative comparison of parser accuracy of leading dependency parsers on ungrammatical sentences; this may help practitioners to select an appropriate parser for their applications.
  - We have conducted a suite of robustness analyses for the parsers on specific kinds of problems in the ungrammatical sentences; this may help developers to improve parser robustness in the future.
- We have proposed parse tree fragmentation framework as a way to address the mismatch between ungrammatical sentences and statistical parsers that are not trained to handle them.
  - We have devised methods for extracting gold standard tree fragments using evaluative corpora available for other NLP applications.
  - We have proposed three practical fragmentation methods based on availability of resources for each ungrammatical domain.
  - We have verified utility of extracted tree fragments for two downstream NLP applications of fluency detection and semantic role labeling.

2.0 PRELIMINARIES AND BACKGROUND

2.1 INTRODUCTION

In this chapter we give an overview of the terminology and the concepts that are discussed throughout this dissertation. In doing so, we start out with reviewing several domains that contain ungrammatical sentences. We then turn to syntactic parsing with the special focus on partial parsing approaches, which have similar concept as our parse tree fragmentation framework. Next, we provide an overview of evaluating parse trees, especially on downstream NLP applications that leverage parsing as a component. We discuss two specific applications that we explore in this thesis, by defining the tasks and giving an overview of the related work.

2.2 UNGRAMMATICAL SENTENCES

Different domains of ungrammatical sentences might have unique properties that introduce various challenges for the NLP applications. In this section, we review several natural data sources in which the sentences may contain grammatical mistakes: writings of ESL students, automatic machine translation outputs, Twitter data, and Automatic Speech Recognition (ASR) transcripts. For each data source, we first present its main characteristics and available corpora; then we introduce some common NLP approaches used to process its sentences. Finally, we compare these ungrammatical domains from the parsing perspective. In this thesis, we focus on written domain of ESL and MT, since their major goal is to generate fluent and grammatical sentences.

2.2.1 English-as-a-Second Language (ESL)

2.2.1.1 ESL Corpora

Because English-as-a-Second Language (ESL) learners tend to make mistakes when learning English, they often create ungrammatical sentences. To further study ESL mistakes, researchers have created learner corpora where English experts mark and correct errors. These learner corpora have different annotation standards and different error categories. Despite their differences, they all include basically the same general types of errors. They all consider missing, unnecessary and replacing word errors based on the part of speech tag of the involved word. By knowing the expert corrections of the sentences that show the location and type of the errors, one can easily reconstruct the corrected version of each ungrammatical ESL sentence. The following is an example of the given information in an ESL corpus:

ESL Sentence: We live in changeable world.

Corrections: (Missing determiner "a" at position 3)

(An adjective needs replacing with "changing" between positions 3 and 4)

Given this information, the corrected version of the ESL sentence can be reconstructed:

Corrected ESL Sentence: We live in a changing world.

In this thesis, we use three available ESL corpora:

- **First Certificate in English (FCE)** (Yannakoudakis et al., 2011). This is a commonly used corpus in the grammar error correction community and has around 31,500 sentences written by students taking Cambridge English exams. 21,000 of the sentences have at least one grammar mistake. These sentences are corrected by English teachers with the detailed list of corrections (containing the type and the position of errors).
- **National University of Singapore Corpus of Learner English (NUCLE)** (Dahlmeier et al., 2013). This corpus is used in the grammar error correction shared tasks of CoNLL-2013 (Ng et al., 2013) and CoNLL-2014 (Ng et al., 2014). It contains 60,800 sentences written by Singaporean college students; among which 21,500 sentences have at least one mistake. The erroneous sentences are corrected by English teachers with the detailed list of corrections (containing the type and the position of errors).

- **EF-Cambridge Open Language Database (EFCAMDAT)** (Geertzen et al., 2013)[1]. This corpus has a considerable size of sentences submitted to Englishtwon, the online school of EF that is accessed by thousands of learners each day. This corpus will continue to grow as new data come it. The version of corpus that we used has more than 1,200,000 sentences with at least one grammar mistakes.[2] These sentences are corrected by teachers or correctors to provide feedback to learners. Even though these errors are annotated with some error codes (e.g. article or verb tense error types), the corrections are not as detailed and accurate as FCE and NUCLE corpora. Since, the corrections are not reliable enough, we only used them to reconstruct the grammatical sentences from the ungrammatical sentences. Thus, we use this huge parallel sentences (ungrammatical/grammatical) as a resource to train our automatic fragmentation methods that require a large amount of data.

2.2.1.2 NLP Research on ESL

NLP techniques are used to automatically assess learners' writings, detect any errors, and suggest possible corrections for these errors. In the following, we focus on the area of grammar error detection and correction, and its connection with parsing ESL writings.

Grammar Error Correction (GEC)

The ultimate goal of GEC is to build a system to automatically provides feedback to writers, whether they are second language learners or native speakers of a language. Spellcheckers and grammar checking tools (e.g. Microsoft word's grammar checker) are the most visible fruits of GEC research. In this thesis, our focus is on processing writings of English learners.

In the past few years, the interest in GEC systems has grown considerably. The recent shared tasks of Helping Our Own (HOO) (Dale and Kilgarriff, 2011; Dale et al., 2012) and Conference on Natural Language Learning (CoNLL) (Ng et al., 2013, 2014) played an important role in progress on GEC research. Three leading state-of-the-art approaches of correcting grammatical errors are: 1) building specific classifiers for different error types (Rozovskaya and Roth, 2014), 2) using statistical machine translation to correct whole sentences (Rozovskaya and Roth, 2016; Yuan and

[1] https://corpus.mml.cam.ac.uk/efcamdat1/EFCamDat_UserManual_v02.pdf

[2] We filter out the annotated sentences that have only capitalization errors, or merging two sentences together. Because these error types does not make any difference for our parsing strategies or are not in the sentence-level.

Briscoe, 2016), and 3) using sequence-to-sequence approaches to generate the correct sentences (Schmaltz et al., 2016, 2017).

Parsing ESL Writings

The typical approach to parse domain specific sentences is to train a parser using manually annotated gold parse trees for the sentences of that domain. However, there are a few small semi-manually constructed treebanks on learner text (Geertzen et al., 2013; Ott and Ziai, 2010; Berzak et al., 2016), their size makes them unsuitable for training a parser. Moreover, some researchers also raise valid questions over the merit of creating a treebank for ESL writings or adapting the annotation schema (Cahill, 2015; Ragheb and Dickinson, 2012).

An alternative approach to parse ESL sentences is to use GEC systems to first correct erroneous sentences, and then parse the corrected versions in a pipeline manner. However, fixing problematic sentences may not always be possible, specially when they are very jumbled. Moreover, GEC systems are not perfect and even have mediocre performance on standard ESL corpora. The precision, recall and F-measure of leading GEC systems on the standard ESL corpora are around 0.60, 0.25, 0.50 respectively (Rozovskaya and Roth, 2014, 2016).

2.2.2 Machine Translation (MT)

2.2.2.1 MT Corpora

Machine translation outputs are another source of problematic data that contain grammatical errors. Unlike the ESL corpora, the MT corpora do not have detailed error corrections. The MT corpora often contain the machine translation outputs and human translations against which MT systems are evaluated. The human translation sentences are called *reference* sentences in the MT community. In some cases, the machine translation outputs are manually revised with the goal of performing minimal necessary operations. This set of refined translations is called *human post-editions*. A good source of MT corpora is the annual conference on statistical machine translation which is built on a series of annual Workshops on Machine Translation (WMT)[3]. The released datasets contain machine translation outputs submitted to various shared tasks along with the reference translations and occasionally human post-editions. The following exemplifies an MT output

[3]http://www.statmt.org

and its accompanied resources:

MT Output: For almost 18 years ago the Sunda space "Ulysses" flies in the area.

Reference Sentence: For almost 18 years, the probe "Ulysses" has been flying through space.

Post-edited Sentence: For almost 18 years the "Ulysses" space probe has been flying in space.

In this thesis, we use the following machine translation corpora that contain machine translations and their human post-editions:

- **LIG corpus** (Potet et al., 2012)[4]. This corpus contains 10,881 French-English machine translation outputs and their human post-editions.
- **LISMI's TRACE corpus** (Wisniewski et al., 2013)[5]. This corpus has 6,693 French-to-English machine translation outputs and their human post-editions.

2.2.2.2 NLP Research on MT Outputs

The NLP research on the machine translation outputs is mainly on evaluating the translations. Existing automatic MT evaluation metrics, such as BLEU (Papineni et al., 2002), METEOR (Denkowski and Lavie, 2011), TER (Snover et al., 2006) and MEANT (Lo and Wu, 2011), primarily provide a single-value evaluation of the quality of the translation. But the task of improving a translation system needs more detailed information about identifying source of errors in a given system. One of the early works on error analysis of MT outputs is done by Vilar et al. (2006). They introduced a general MT error typology that has been widely used in the literature (Fishel et al., 2011; Berka et al., 2012; Popović and Ney, 2011). Although these error types have a significant overlap with ESL error categories, the MT systems do not make spelling errors; instead they are not able to translate some words and keep them untranslated.

In our own research, we have previously analyzed MT errors in terms of ESL mistake categories (Hashemi and Hwa, 2014). We assumed that an automatic translate-to-English system might be seen as an English as a Second Language (ESL) writer whose native language is the source language. The results suggested that MT systems have fairly similar distributions regardless of their source languages, and the high-performing MT systems have error distributions that are more sim-

[4] http://www-clips.imag.fr/geod/User/marion.potet/index.php?page=download
[5] anrtrace.limsi.fr/trace_postedit.tar.bz2

ilar to those of the low-performing MT systems than to those of ESL learners with the same L1. This may be due to the common English language model component that all the MT systems use.

2.2.3 Twitter

2.2.3.1 Twitter Properties

Twitter is a new domain of noisy data for NLP. One property of tweets is that users are free to send any short message up to 140 characters. The messages are often informal written texts that do not follow the standard rules of writing. Tweets contain different forms of non-standard language: using special character (e.g. # hashtags and emoticons), lengthening words (e.g. *coooollll!!!!!*), shortening words (e.g. *u* or *2moro*), and abbreviating phrases (e.g. *lol*) (Eisenstein, 2013). Aside from these specific properties of tweets, tweets might also be ungrammatical and contain unintentional spelling errors.

2.2.3.2 NLP Research on Tweets

Tweets differ from standard language, such as news reports, both in style and vocabulary. The problem is that the traditional NLP tools (trained on edited text) work poorly on them (Kong et al., 2014; Ritter et al., 2011; Foster et al., 2011a). The NLP researches have followed two approaches to deal with the informal language of tweets: normalization and domain adaptation (Eisenstein, 2013). Normalization is the process of replacing non-standard words with "the contextually appropriate word or sequence of words" (Foster et al., 2011a). For example, transforming *coooollll!!!!!* to *cool!*. Although normalization is a challenging task (because it is sometimes impossible to keep the meaning of the sentences), some normalization labeled corpora have been created and accuracy of automatic methods are climbing (Han and Baldwin, 2011; Han et al., 2012).

Another approach is to adapt NLP tools with the properties of tweets. Some NLP tools are specially created for Twitter, such as a part-of-speech tagger (Gimpel et al., 2011), a named entity recognizer (Ritter et al., 2011), and a dependency parser (Kong et al., 2014). Kong et al. (2014) have also annotated a small set of tweets with their parse trees to train and test the Twitter dependency parser, *Tweebo*. One property of Tweebo is that it is designed to ignore some tokens when parsing the input text, but simultaneously use the ignored tokens as features. We hypothesize that this property would be helpful for ungrammatical sentences specially when they have redundant words. To investigate our hypothesis, we evaluate the robustness of Tweebo parser on two domains of ungrammatical sentences in the next chapter.

2.2.4 Transcribed Conversation

2.2.4.1 Transcribed Conversation Corpora

Automatic Speech Recognition (ASR) transcripts of conversational speech offer another natural source of problematic data; however, annotated disfluency in spoken utterances tend to focus on removing extra fillers and repeated phrases (Rasooli and Tetreault, 2013; Honnibal and Johnson, 2014; Ferguson et al., 2015). These sentences are also typically shorter and simpler in their syntactic structures. An example of annotated disfluent utterance is Switchboard corpus (Godfrey et al., 1992), which contains transcribed conversation and parse trees annotated with edited nodes. The annotated edited nodes help to reconstruct the fluent version of the utterances, indicating what the speaker meant to say. The following is an example of observed utterance and its “cleaned” fluent version:

Disfluent Utterance: I want a flight to Boston, uh, I mean Denver

Fluent Version: I want a flight to Denver

A listener can often subconsciously filter out spoken disfluencies. However, these disfluencies negatively impact the accuracy of automated analysis performed on spoken utterances.

In this thesis, we primarily focus on written domains because ungrammatical sentences in these domains have a wider range of error types, such as missing phrases, insertion of unnecessary phrases, and incorrect phrasal ordering, than spoken domains.

2.2.4.2 NLP Research on Transcribed Conversation

Most of the NLP research on the speech utterances is to detect disfluencies (Georgila, 2009; Qian and Liu, 2013). It is then possible to use a pipeline system and give the fluent version of the utterances to the available NLP tools. Some approaches treat disfluency detection and parsing jointly (Rasooli and Tetreault, 2013; Honnibal and Johnson, 2014). They show that a joint system can improve the both disfluency detection and parsing of speech utterances.

2.2.5 Comparison of Ungrammatical Domains

We compare the four domains of ungrammatical sentences with the focus of syntactic parsing. We specifically address the following questions:

- Do the ungrammatical sentences have various error types, i.e. missing, unnecessary, and replacement word errors?

  ESL and MT sentences might have various grammatical mistakes. While Twitter data and ASR transcripts do not necessarily have missing terms or replacement word errors; but, they might contain redundant terms that need to be ignored while parsing them.

- Is it possible to collect the corrected version of ungrammatical sentences to obtain a parallel corpus of ungrammatical/grammatical sentences?

  It is often possible to reconstruct corrected version of ungrammatical sentences for ESL, MT, and ASR transcripts using the existing corpora for these domains. However, tweets represent the informal language which often might not be possible to build unique versions for them in the standard language of writing.

- Do the ungrammatical sentences have detailed correction annotations?

  The ESL and ASR transcripts are usually annotated with the detailed corrections showing the exact location and type of errors.

- Is there a manually created treebank for them?

  There are small dependency treebanks manually annotated for a subset of ESL and Twitter data. The Switchboard corpus is also a sizable constituency treebank for the spoken language.

- Is there a specialized parser designed for them?

  Tweebo parser is an adapted parser to handle parsing tweets. Researchers have also proposed to adapt parsers to jointly parse and detect disfluency in spoken utterances.

Table 1 summarizes the comparison of the ungrammatical domains. In this thesis, we focus on investigating parsing ESL and MT ungrammatical domains. Because, 1) the ultimate goal of these domains is often to make fluent and grammatical sentences, 2) they have a wide range of grammatical mistakes, and 3) no parser is designed for them. Furthermore, they have parallel

Property	ESL	MT	Twitter	ASR transcripts
Various error type	✓	✓	-	-
Parallel data	✓	✓	-	✓
Detailed error annotation	✓	-	-	✓
Treebank	✓ ($\sim$ 5000 sent.)	-	✓ ($\sim$ 900 sent.)	✓ ($\sim 110,500$ sent.)
Specialized parser	-	-	✓ (Tweebo)	✓ (Joint systems)

Table 1: Comparison of the ungrammatical domains.

corpora of ungrammatical sentences with their corrections that we can use for studying parsing in these domains.

2.3 SYNTACTIC PARSING

Syntactic parsing is the task of assigning a syntactic structure to a sentence. The syntactic structure characterizes the possible relations and orderings of words within the sentence. In this section, we give conceptual view on the various kinds of structures assign to sentences by categorizing them into two groups of fully-connected and locally-connected structures.

2.3.1 Full Parsing

There are two major syntactic representations: a constituency (phrase-based) and a dependency. Each representation produces a fully-connected structure for a sentence that encodes relationships between words, but the form of the structures varies considerably. We review the general properties of both representations to indicate that our proposed approaches can be generalized for both representations. For the purposes of this thesis, we focus on dependency representation.

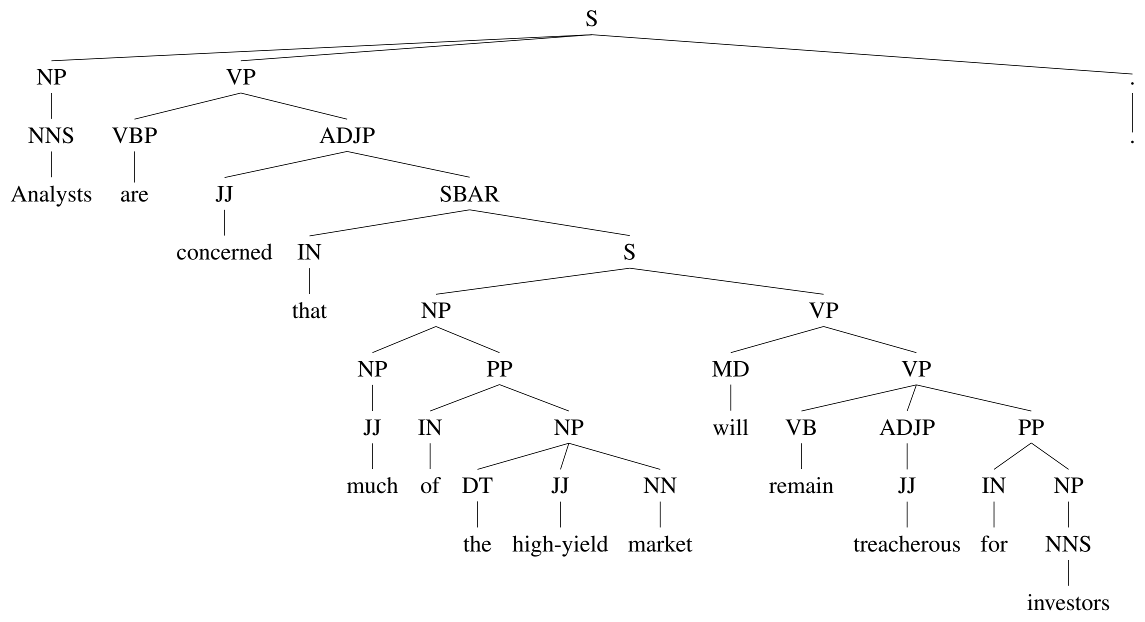


Figure 3: Example of a full constituency parse tree.

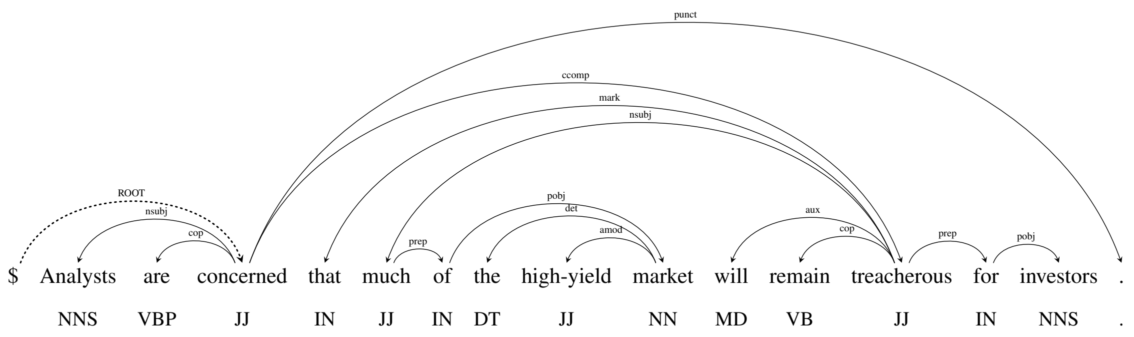
2.3.1.1 Constituency Parse Tree

A constituency (phrase-based) parse tree breaks a sentence into sub-phrases (Jurafsky and Martin, 2009). The interior nodes of the tree are types of phrases (e.g. noun phrase (NP) or verb phrase (VP)), while the leaf nodes are the words in the sentence. Figure 3 shows the constituency parse for the sentence *Analysts are concerned that much of the high-yield market will remain treacherous for investors*. In general, a phrase structure representation may be found more suitable for languages with rather clear constituency structures and fixed word order patterns.

Constituency representation is used as the syntactic formalism when annotating sentences with their parse trees in several large scale human annotation treebanks, such as Penn Treebank project (Marcus et al., 1993). These treebanks contain a collection of sentences with their manually annotated parse trees which can then be used to train and evaluate statistical parsers.

2.3.1.2 Dependency Parse Tree

A dependency parse tree connects words in a sentence. Each node in the tree represents a word, and each edge indicates the relationships between two words. The edges (or arcs) are called *dependency*



punct
ccomp
mark
nsubj
ROOT
nsubj
cop
pobj
det
amod
prep
aux
cop
prep
pobj
$ Analysts are concerned that much of the high-yield market will remain treacherous for investors .
NNS VBP JJ IN JJ IN DT JJ NN MD VB JJ IN NNS .

Figure 4: Example of a full dependency parse tree.

relations and are labeled by the type of the dependency (Jurafsky and Martin, 2009). Figure 4 shows an example of the dependency tree. The direction of the arrow is from the head/parent word to the dependent (modifier)/child word[6]. For instance, "Analysis" is the subject (nsubj) dependent of the head word "concerned". Often an artificial ROOT token ($ in our example) is added to the dependency tree to ensure that every word in the sentence has one associated head word.

Dependency parse trees contain fewer nodes than constituency parse trees; instead of focusing on phrase-structure rules and constituents (as in the constituency trees before), dependency structure of a sentence is only described in terms of binary relations between words, so they contain fewer nodes. Another characteristics of dependency trees that make them more common in the recent years is that they are more suitable for languages with free word order, such as Czech and Turkish.

To be able to utilize the constituency treebanks in dependency representation, various work has been done to convert constituency trees to dependency forms. These conversions leverage linguistic phenomena and are mostly deterministic (rule-based) transformations. In Section 3.4.2.1, we have used one of these methods to convert *Wall Street Journal* part of the Penn Treebank to dependency parse trees.

[6]Note that the direction we follow is a convention in graph theory, and this is the reverse of the convention in linguistics.

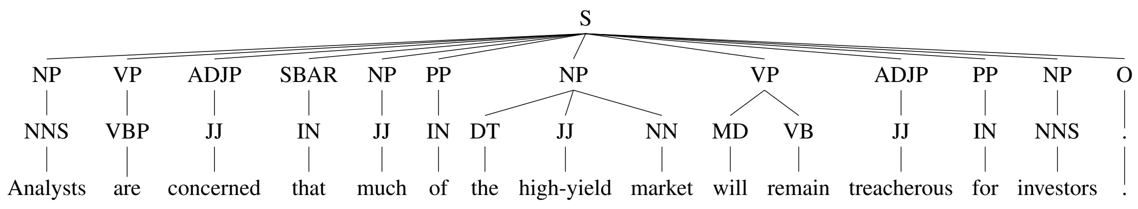


Figure 5: Chunking analysis of a sentence.

2.3.2 Partial Parsing

Many NLP applications may not require fully connected, complex parse trees. Partial parsing (or shallow parsing) is used in these situations to render a superficial syntactic analysis of a sentence. For example, consider a systems that deals with tweets and needs to process them in a short amount of time. Such text is problematic for either constituency or dependency parsers; the generated complete parse trees might not match the tweet and most importantly they are costly. While partial parsing is fast and still may give useful nuggets of information by omitting all but the most basic syntactic segments.

In this section, we provide an overview of three existing approaches that partially parse sentences: chunking, hedge parsing and vine parsing. Our proposed parse tree fragmentation framework is similar to these existing partial parsing methods in the sense that they all tend to break up parse trees to identify recognizable phrases. But the difference is that we break up the trees with regard to grammar mistakes to handle mismatches between the ungrammatical sentences and their syntactic structures. In the remaining of the section, we review the partial parsing methods to demonstrate their differences with our work.

2.3.2.1 Chunking

Chunking is an alternative style of partial parsing. It processes the text to identify and classify the flat non-overlapping segments. These segments correspond to the content-bearing parts of the sentence and typically include noun, verb, adjective and prepositional phrases. The identified constituents do not specify their internal structures or relations with other constituents in the sentence.

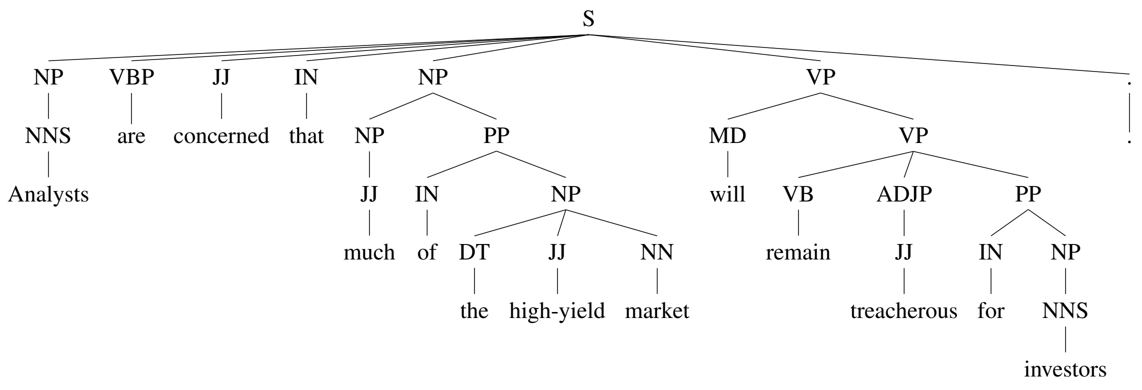


Figure 6: Example of Hedge parsing with maximum constituent span of 6.

Figure 5 shows an example of a chunk parse tree.

2.3.2.2 Hedge Parsing

Hedge parsing (Yarmohammadi et al., 2014) provides local internal hierarchical structure of phrases without requiring fully connected parses. Its goal is to find a less computationally demanding syntactic parser than a full parser but more expressive than a chunker. Hedge parsing discovers constituents of length up to some maximum span, i.e, the constituent nodes that cover more than some words are recursively elided. Then the hedges are sequentially connected to the top most nonterminal in the tree. Figure 6 shows an example of hedge parse tree for the full parse tree of Figure 3. The hedge parse tree keeps only full hierarchical annotations of structures within a local window and ignores global constituents outside that window. In the example, the constituents that cover less than 6 words are kept.

2.3.2.3 Vine Parsing

Similar pruning approaches have been used in dependency parsing known as vine parsing (Eisner and Smith, 2005; Dreyer et al., 2006). Vine parsing behaves like hedge parsing and has a set of constraints on arc lengths that considers only close words as modifiers. The assumption behind these constraints is that a word's dependents tend to fall near it in the sentence. Eisner and Smith (2005) proposed a vine parser that imposes a bound on the length of each dependency relation,

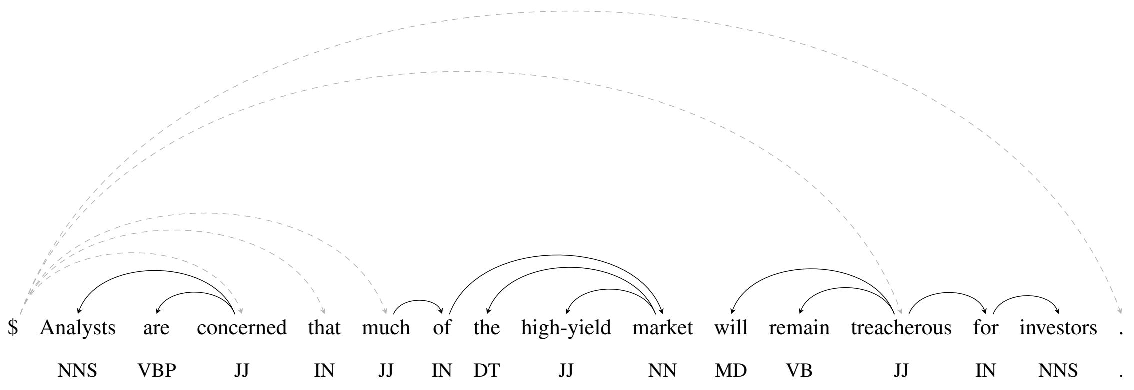


Figure 7: Example of Vine parsing retaining only tree dependencies of length less than 6. The root of the resulting parse fragments are now connected only by their dotted arcs "vine dependencies" to $.

which is the string distance between the child and its parent. This set of hard constraints completely ignores long dependencies in the parser. Figure 7 shows an example of a vine dependency tree for the full parse tree in Figure 4. The vine parser keeps only dependencies of length less than 6; thus five of the dependency relations are broken due to this length constraint. The modifiers of the broken arcs are then connected to the root of the tree.

2.3.3 Parsing Evaluation

A good syntactic parser is expected to produce an accurate parse tree for a sentence. This parse tree is typically used in a downstream NLP application. It is also to be expected that the performance of an NLP system degrades if the generated parse tree is incorrect. Thus, the generated parse trees can be evaluated in two main criteria (Resnik and Lin, 2010): *Intrinsic* evaluation and *Extrinsic* evaluation. Intrinsic evaluation would analyze the accuracy of the produced parse trees as a stand-alone system, whereas extrinsic evaluation would analyze the performance of the parse trees within downstream NLP applications.

2.3.3.1 Intrinsic Evaluation

The intrinsic evaluation of parsers proceeds by comparing the output of a parser against gold standard parse trees provided by human annotators. Depending on the representation of the parse tree whether constituency or dependency, the standard evaluation metrics are defined differently.

Constituency parse tree

The most widely used constituency evaluation techniques are called the PARSEVAL measures (Black et al., 1991). The PARSEVAL evaluates how much the constituents in the generated parse tree look like the constituents in a gold standard parse tree. The gold standard parse trees are generally drawn from a treebank such as the Penn Treebank.

A constituent in the output of parser is correct if there is a constituent in the gold standard parse with the same span of words and same non-terminal symbol. The labeled precision and recall are then calculated as:

$$\text{Precision} = \frac{\text{\# of correct constituents in generated parse}}{\text{Total number of constituents in output generated parse}}$$

$$\text{Recall} = \frac{\text{\# of correct constituents in generated parse}}{\text{Total number of constituents in gold standard parse}}$$

Often F-score is reported as the harmonic mean of precision and recall:

$$\text{F-score} = 2 \times \frac{\text{Precision} \times \text{Recall}}{\text{Precision} + \text{Recall}}$$

These metrics are used to evaluate the accuracy of various constituency parsing approaches such as chunking and hedge parsing.

Dependency parse tree

The standard metrics for evaluating dependency parsing are labeled and unlabeled attachment accuracy. Given a generated parse tree and a corresponding gold standard tree, labeled and unlabeled attachment accuracy are simply the percentage of correct assignments:

$$\text{Labeled Attachment Score (LAS)} = \frac{\text{\# of words with correct head and correct dependency label}}{\text{Total number of words}}$$

$$\text{Unlabeled Attachment Score (UAS)} = \frac{\text{\# of words with correct head (ignoring dependency label)}}{\text{Total number of words}}$$

These metric are used to evaluate the performance of dependency parsing approaches such as vine parsing.

2.3.3.2 Extrinsic Evaluation

It is important to know not only the accuracy of a parser but also the impact of the parser in a real NLP application. This is the goal of the extrinsic evaluation, where the parser is evaluated as an embedded component of an application. It is only with extrinsic evaluation that researchers can tell if a parsing technique is working in the sense of actually improving performance of a system.

On the other hand, while extrinsic evaluation gives a better sense of the impact of parsers, it requires integrating parsers into a complete working application which making it to be much more difficult and time-consuming to implement. Furthermore, an extrinsic evaluation analysis on one application may not generalize to other applications.

The parser's output is used in several downstream NLP taks, such as machine translation (Quirk and Corston-Oliver, 2006), information extraction (Miyao et al., 2008), and semantic dependencies (Dridan and Oepen, 2011). Just recently, a new shared task of Extrinsic Parser Evaluation (EPE)[7] is introduced. This shared task focuses on providing better estimates for different dependency representations on a variety of downstream applications that rely on the syntactic structure of sentences. The downstream applications that EPE supported are: biological event extraction (to recognize bio-molecular events that are mentioned in biomedical literature), fine-grained opinion analysis (to extract MPQA-style (Wiebe et al., 2005) opinion expressions from text), and negation resolution (to find scope of negated cue). In the next section, we discuss the NLP applications that we explore in this thesis to evaluate the fragmentation framework over the ungrammatical sentences.

[7]http://epe.nlpl.eu

2.4 SYNTACTIC PARSING APPLICATIONS

As we saw in the extrinsic evaluation section (Section 2.3.3.2), there is a wide range of NLP applications that leverage parsing as a component. In this section, we discuss the two applications that we explore in this thesis: 1) *sentence-level fluency judgment* as the task of predicting how much grammatical a sentence is, and 2) *semantic role labeling (SRL)* as the task of identifying semantic dependencies between words in a sentence. We choose fluency judgment application because it is the direct application of parsing that deals with ungrammatical sentences; we also choose SRL application because it is one of the basic tasks in semantic analysis and studying the behaviour of SRL systems on ungrammatical sentences could shed some light on this problem.

2.4.1 Sentence-Level Fluency Judgment

An automatic fluency judgment system detects whether a sentence is hard to read. It can be useful in various applications; for example it can be used in grammar checking systems to help both native and L2 speakers to improve their written communication; it can also be used to decide whether an MT output needs to be post-processed by a professional translator.

2.4.1.1 Fluency Judgment Task

Judging fluency of a sentence is a basic task that typically has to be performed before further analyzing the errors in the sentence. Gamon and Leacock (2010) describes fluency judgment as "a baseline task that any error detection and correction system needs to address." Fluency judgment could be either a binary task, i.e. to decide whether the sentence contains errors, or a finer-grained predication, i.e. to predict the degree of grammaticality of the sentence. Note that we use the terms fluency and grammaticality interchangeably; in some previous work this task is referred as grammaticality judgment (Wagner et al., 2009; Post, 2011).

One measure of grammaticality is to directly use the language model perplexities of sentences. However, they are not sufficient because they do not capture long distance dependencies. In addition, since grammaticality is a matter of syntax of a language, syntax-based features can be used to measure it. Therefore, various parse tree features have been incorporated to help grammaticality judgment task (Post, 2011; Post and Bergsma, 2013; Mutton et al., 2007).

2.4.1.2 Fleuncy Judgment Related Work

Fluency judgment has been studied vastly in two domains of human generated text and machine generated text. A recent resource in human generated text is introduced in Automated Evaluation of Scientific Writing (AESW)-2016[8] shared task (Daudaravicius et al., 2016). The task is to predict whether a given sentence requires editing to improve it. Thus the task evaluated as a binary classification. The machine generated text is studied more broadly in different NLP applications, such as summarization and machine translation. In the machine translation domain, there is a series of Quality Estimation (QE)[9] shared tasks organized by the WMT conference (Bojar et al., 2016). The QE task is to estimate the quality of machine translation output on real time given the input sentence in the source language. The scores produced by a QE system can be used to decide whether the machine translation output is good enough to be published or it needs further post-editing; the scores can also be used to choose between translations e.g. in translation reranking.

Automatic fluency judgment can be made in various ways. The basic approach is to estimate the grammaticality of a sentence using a language model. The simplest language models, n-grams, have been productive throughout natural language processing applications on both human and machine outputs; for instance, in automatic essay grading or in picking the best translation of a machine translation system. Although n-gram models are long-studies and easy to train, they are insufficient as models of language since they are unable to (easily) capture long distance linguistic phenomena. As a result, they are not able to detect grammatical issues in the sentences.

Since grammaticality judgment is a matter of syntax of a language, another approach for modeling grammaticality is to leverage syntactic features. The parse score and context-free grammars (CFGs) are used as features in fluency judgment classifiers and shown improvement upon n-gram baselines (Cherry and Quirk, 2008; Wagner et al., 2009; Wong and Dras, 2010; Heilman et al., 2014). A further successful approach in grammaticality task is to use Tree Substitution Grammars (TSGs) (Joshi and Schabes, 1997) which are generalized form of context-free grammars that allow nonterminals to rewrite as tree fragments of arbitrary size. Post (2011) demonstrated that larger

[8]http://textmining.lt/aesw/index.html
[9]http://www.statmt.org/wmt17/quality-estimation-task.html

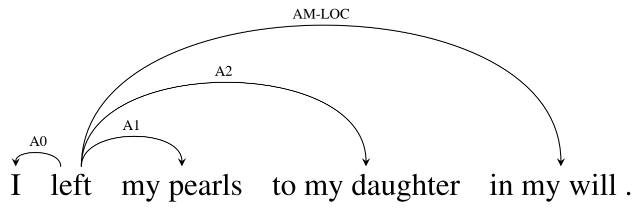


Figure 8: Example of semantic role labeling.

tree fragments of TSG are more natural units in grammatical sentences; thus they are less likely to fit into ungrammatical sentences. They learned TSGs automatically from a Treebank with a Bayesian model, then used TSG derivations as features for grammaticality classification. We use this model as one of our baselines in Section 7.2.

2.4.2 Semantic Role Labeling (SRL)

Semantic role labeling (SRL) is crucial to natural language understanding as it identifies the semantic relations in text. These relations provide a more stable semantic analysis across syntactically different sentences; as a result, they can be used in a range of NLP tasks such as information extraction and question answering (Shen and Lapata, 2007; Maqsud et al., 2014).

2.4.2.1 SRL Task

The goal of semantic role labeling task is to identify the roles of groups of words with respect to a particular verb in a sentence. Recognizing these roles is a key task for answering "what", "when", "who", "why", etc. questions in all NLP applications in which some kind of semantic interpretations is required, such as information extraction, question answering and summarization. For example, given a sentence *"I left my pearls to my daughter in my will."*, the goal is to detect arguments of the verb "left" and produce the semantic dependencies as in Figure 8. Here "I" is the leaver, "my pearls" is the thing left, "to my daughter" represents the beneficiary, and "in my will" indicates the location of the action. The semantic roles are commonly divided into core arguments (A0-A5) and additional common classes such as location, time, etc. These roles have different semantics for each verb, though A0 most often refers to agents, and A1 refers to patients. Table 22 in the Appendix shows more details about semantic roles. Different senses of arguments are specified in the frame files of the PropBank (Kingsbury and Palmer, 2002), which is an annotated text with roles for each argument.

2.4.2.2 Relation of Syntactic and Semantic Analyses

Syntactic parsing plays an important role in semantic role labeling; it provides various syntactic features, such as "path" between predicate and argument (proposed by (Gildea and Jurafsky, 2002)), that are mainstay of high performing semantic role labeling systems (FitzGerald et al., 2015; Roth and Woodsend, 2014; Foland and Martin, 2015). For example, as depicted in the top part of Figure 9, the semantic roles of the grammatical sentence overlaps with its dependency tree.[10] Although dependency parsing and semantic role labeling have different definitions (the former spans over a sentence, while the latter centers around individual predicates), their outputs often overlap. This is because the modifiers of the verbs in a parse tree tend to be its arguments in the semantic graph. Such overlaps corroborate the impact of syntactic parsing on the semantic role labeling.

In addition, for the purposes of this thesis, we investigate the impact of ungrammatical mistakes on the syntax of the sentence and thus on its semantic. For example, the bottom part of the Figure 9 shows an ungrammatical sentence written by an English-as-a-Second Language (ESL) learner. The ungrammatical sentence has two small mistakes (a missing comma and a phrase replacement error), but the impact of these mistakes is significant on the syntactic parse. Even though the parse tree of the ungrammatical sentence looks well-formed, the syntactic structure does not closely resemble the analysis for the corrected sentence (top part of figure): the head of the ungrammatical sentence is changed to "remember" from "known", and the "for ever" phrase has preposition relation instead of time adverb. The figure also shows the impact of grammatical mistakes on the interpretability of the semantic dependency graph, as compared to the correct version. Because of the mistakes in the sentence, the semantic graph of the ungrammatical sentence has some extra semantic dependencies: "remember→I" and "known→for". In this thesis, we will study the impact

[10]Dependency trees are produced by SyntaxNet parser (Andor et al., 2016) and semantic dependency graphs are produced by semantic role labeler of the Mate toolkit (Björkelund et al., 2009).

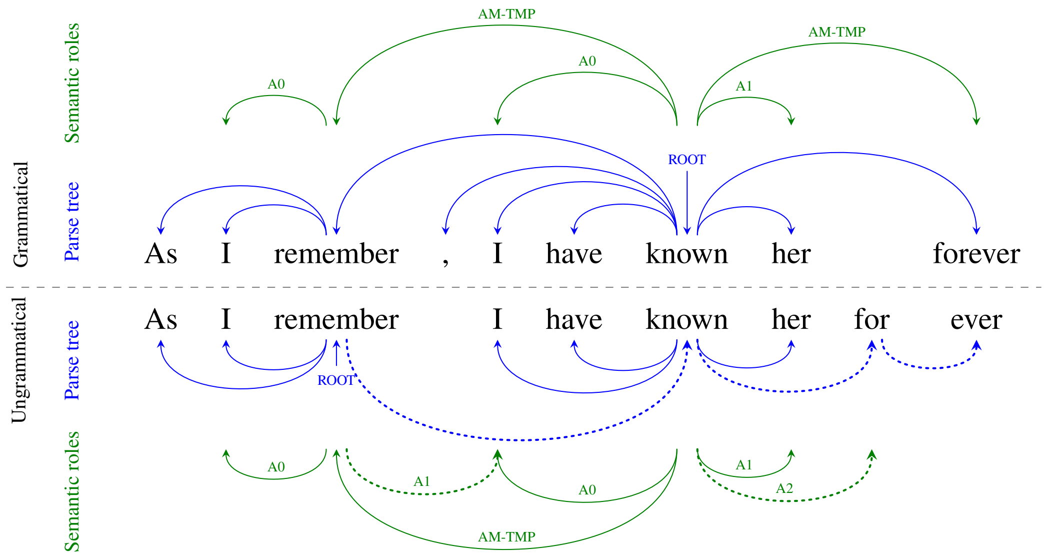


Figure 9: Syntactic (inner) and semantic (outer) analyses of an ungrammatical sentence (bottom) and its corrected version (top). The dotted arcs show mismatched dependencies of the ungrammatical sentence with the grammatical sentence.

of syntax in detecting these incorrect semantic dependencies (more details are given in Section 7.3).

2.4.2.3 SRL Related Work

The availability of resources such as PropBank corpus (Palmer et al., 2005) and organizing SRL shared tasks of CoNLL-2004 and CoNLL-2005[11], has enabled significant progress in SRL systems over the past decade. State-of-the-art SRL systems follow two main approaches. The first approach, which is widely used, employs a linear classifier with feature templates. A huge amount of efforts have been made to extract the best discriminative features. One of the most important set of features is defined based on syntactic parsing. Pradhan et al. (2005) and Punyakanok et al. (2008) used the generated parse trees and assigned semantic role labels to the constituents for each parse tree. They showed that combining features from different syntactic views brings large improvement for the SRL systems.

The second approach tries to solve SRL problem without feature engineering (Collobert et al., 2011; Zhou and Xu, 2015). Collobert et al. (2011) proposed a convolutional neural network model by initializing with word embeddings. Since convolution layer does not model long distance dependencies, they had to process the whole sequence for each given argument-predicate pair. Therefore, their introduced model is computationally expensive. Moreover, they also incorporated syntactic features of Charniak parser, in order to catch up with the performance of traditional methods.

[11]http://www.cs.upc.edu/~srlconll/

3.0 IMPACT OF UNGRAMMATICAL SENTENCES ON PARSING

3.1 INTRODUCTION

In this chapter, we investigate the impact of ungrammatical sentences on parsers by addressing the question: How much does a parser's performance degrade when dealing with ungrammatical sentences? If a parser can overlook problems such as grammar mistakes and produce a parse tree that closely resembles the correct analysis for the intended sentence, it is said that the parser is *robust* (Bigert et al., 2005; Kakkonen, 2007; Foster, 2007). For example, consider the following ESL sentence and its corresponding correction from the FCE dataset (Yannakoudakis et al., 2011):

ESL Sentence: This made me get bored.

Corrected ESL Sentence: This made me feel bored.

The only correction is the replacement of the verb "get" with "feel". Thus, we expect that a robust parser produces a similar syntactic structures for both sentences. However, parsing these sentences with the Turbo parser (Martins et al., 2013), we observe inconsistencies between generated parse trees of two sentences. Figure 10 shows the parse trees of the sentences. Although, the sentences have the same part of speech sequence, the parser generates different trees for them around the error word.

Because there is no explicit large-scale gold standard data (treebank) for various domains of ungrammatical sentence, such as machine translation outputs, we introduce a methodology for evaluating robustness of parsers when dealing with ungrammatical sentences. Moreover, to explore the impact of ungrammatical sentences on parsers, we report a set of empirical analyses of the leading dependency parsers on two domains of ungrammatical text.

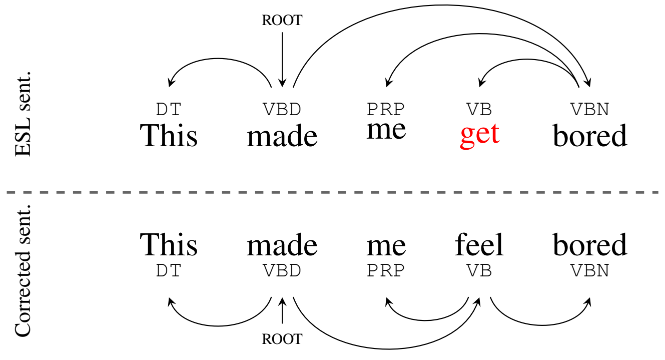


Figure 10: Parse trees of an ESL sentence and its corrected counterpart.

3.2 ASSESSING THE IMPACT OF UNGRAMMATICAL SENTENCES ON PARSERS

To explore the impact of ungrammatical sentences on parsers, we need to be able to evaluate their generated parse trees over the ungrammatical text. However, parser evaluation for ungrammatical text presents some domain-specific challenges. The typical approach to evaluate parsers is to compare parser outputs against manually annotated gold standards. But, these annotated treebanks are not available for all the ungrammatical domains. For example, there is no treebank for machine translation outputs, while there is a considerably large treebank for transcribed conversations Godfrey et al. (1992). For the ESL and social media domains, although there are a few small semi-manually constructed treebanks on learner text (Geertzen et al., 2013; Ott and Ziai, 2010) or tweets (Daiber and van der Goot, 2016), their size makes them unsuitable for the evaluation of parser robustness. Moreover, some researchers also raise valid questions over the merit of creating a treebank for ungrammatical sentences or adapting the annotation schema (Cahill, 2015; Ragheb and Dickinson, 2012). We, therefore, need to come up with an alternative approach be able to evaluate parsers' performances on various ungrammatical domains.

A "gold-standard free" alternative is to compare the parser output for each noisy sentence with the parse tree of the corresponding correct sentence. Foster (2004) used this approach over a small set of ungrammatical sentences and showed that parser's accuracy is different for different types of errors. A limitation of this approach is that the comparison works best when the differences between the noisy sentence and the correct sentence are small. This is not the case for some ungrammatical sentences (especially from MT systems). Another closely-related approach is to semi-automatically create treebanks from artificial errors. For example, Foster generated artificial errors to the sentences from the Penn Treebank for evaluating the effect of error types on parsers (Foster, 2007). In another work, Bigert et al. (2005) proposed an unsupervised evaluation of parser robustness based on the introduction of artificial spelling errors in error-free sentences. Kakkonen (2007) adapted a similar method to compare robustness of four parsers over sentences with misspelled words.

Our proposed evaluation methodology is the most similar to the "gold-standard free" approach; we compare the parser output for an ungrammatical sentence with the automatically generated parse tree of the corresponding correct sentence. In the next section, we discuss our evaluation metric to address the concerns that some ungrammatical sentences may be very different from their corrected versions. This allows us to evaluate parsers with more realistic data that exhibit a diverse set of naturally occurring errors, instead of artificially generated errors or limited error types.

3.3 PROPOSED GOLD-STANDARD FREE METHODOLOGY

For the purpose of parser robustness, we create pseudo gold parse tree for a problematic sentence and then compare the parser output for the corresponding problematic sentence against it.

3.3.1 Creating Pseudo Gold Parse Trees

We propose to create the gold parse tree of an ungrammatical sentence by taking the automatically produced parse tree of a well-formed sentence as "gold-standard". Even if the "gold-standard" is not perfectly correct in absolute terms, it represents the norm from which parse trees of problematic sentences diverge: if a parser were robust against ungrammatical sentences, its output for these sentences should be similar to its output for the well-formed ones. Our proposed gold standard procedure is based on three assumptions (Foster, 2007):

1. For every ungrammatical sentence, there is a grammatical sentence that has the same meaning as the ungrammatical sentence.
2. A state-of-the-art dependency parser produces parse trees of a grammatical sentence that reflects, to some extent, that sentence's correct syntactic structure.
3. The parse tree of an ungrammatical sentence should be as close as possible to the parse tree for its corresponding grammatical sentence.

In keeping with these assumptions, we create gold parse tree for an ungrammatical sentence by projecting the parse tree of its grammatical sentence to the ungrammatical sentence. Following are the steps that we take:

- **Step 1:** Running a state-of-the-art parser over the grammatical sentences.
- **Step 2:** Finding word alignments between ungrammatical and grammatical sentences.
- **Step 3:** Projecting directly the dependency arcs of grammatical sentence to the ungrammatical sentence using the alignments. For each dependency arc in the parse tree of grammatical sentence, if both the head and the modifier are aligned to two words of the ungrammatical sentence, we directly project the dependency arc to the aligned words in the ungrammatical sentence.

Figure 11 shows an example of projecting syntactic dependencies from the grammatical sentence to the ungrammatical sentence.

3.3.2 Evaluating Parse Trees

Determining the evaluation metric for comparing these trees, however, presents another challenge. Since the words of the ungrammatical sentence and its grammatical counterpart do not necessarily match (an example is given in Figure 12), we cannot use standard metrics such as Parseval (Black et al., 1991). We also cannot use adapted metrics for comparing parse trees of unmatched sentences (e.g., Sparseval (Roark et al., 2006)) because these metrics consider all the words regardless of the mismatches (extra or missing words) between two sentences. This is a problem for comparing ungrammatical sentences to grammatical ones because a parser is unfairly penalized when it assigns relations to extra words, and when it does not assign relations to missing words. Since a parser

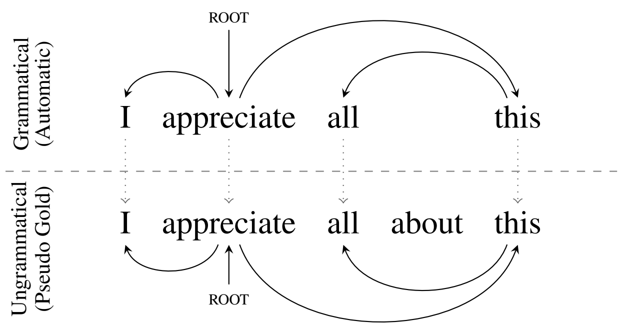


Figure 11: Projecting parse tree of the Grammatical sentence (top) to the Ungrammatical sentence (bottom) to create "gold standard" tree of the ungrammatical sentence.

cannot modify the sentence, we do not want to penalize these extraneous or missing relations; on the other hand, we do want to identify cascading effects on the parse tree due to a grammar error. For the purpose of evaluating parser robustness against ungrammatical sentences, we propose a modified metric in which the dependencies connected to unmatched (extra or missing) error words are ignored. A more formal definition is as follows:

- *Shared dependency* is a mutual dependency between two trees;
- *Error-related dependency* is a dependency connected to an extra word[1] in the sentence;
- Precision is (# of shared dependencies) / (# of dependencies of the ungrammatical sentence - # of error-related dependencies of the ungrammatical sentence);
- Recall is (# of shared dependencies) / (# of dependencies of the grammatical sentence - # of error-related dependencies of the grammatical sentence);
- Robustness F_1 is the harmonic mean of precision and recall.

Figure 12 shows an example in which the ungrammatical sentence has an unnecessary word, "about", so the three dependencies connected to it are counted as error-related dependencies. There are two matched dependencies between the trees, this results in a precision of $2/(5-3) = 1$, recall of $2/(4-0) = 0.5$ and F_1 of 66%.

[1]The extra word in the ungrammatical sentences is an unnecessary word error, and the extra word in the grammatical sentence is a missing word error.

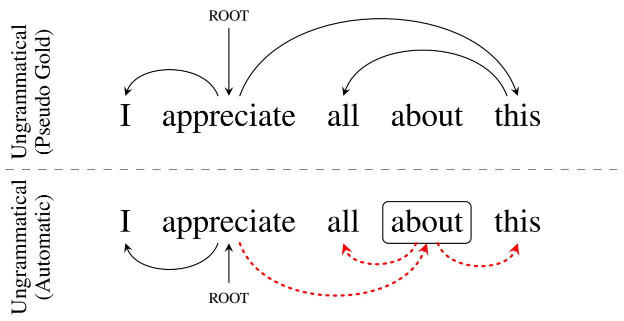

Ungrammatical (Pseudo Gold)
ROOT
I appreciate all about this
Ungrammatical (Automatic)
I appreciate all about this
ROOT

Figure 12: Example of evaluating robustness of an automatic parse tree (bottom) with the gold standard tree (top) of the Ungrammatical sentence. The dotted red arcs show error-related dependencies. The robustness F_1 is 66%.

3.4 EXPERIMENTAL SETUP

Our experiments are conducted over a wide range of dependency parsers that are trained on two different treebanks: Penn Treebank (PTB) and Tweebank. We evaluate robustness of parsers over three datasets that contain ungrammatical sentences: writings of English as a second language learners, machine translation outputs.

3.4.1 Parsers

Our evaluation is over eight state of the art dependency parsers representing a wide range of approaches. For all parsers we use their publicly available versions with the standard parameter settings.

- **Malt Parser** (Nivre et al., 2007)[2] A greedy transition-based dependency parser. We use LIBLINEAR setting in the learning phase.
- **Mate Parser** v3.6.1 (Bohnet, 2010)[3] A graph-based dependency parser that uses second-order maximum spanning tree.

[2]www.maltparser.org
[3]code.google.com/p/mate-tools

- **MST Parser** (McDonald and Pereira, 2006)[4] A first-order graph-based parser that searches for maximum spanning trees.

- **Stanford Neural Network Parser (SNN)** (Chen and Manning, 2014)[5] A transition-based parser that uses word embeddings. We use pre-trained word embeddings from Collobert et al. (2011) as recommended by the authors.

- **SyntaxNet** (Andor et al., 2016)[6] A transition-based neural network parser. We use the globally normalized training of the parser with default parameters.

- **Turbo Parser** v2.3 (Martins et al., 2013)[7] A graph-based dependency parser that uses dual decomposition algorithm with third-order features.

- **Tweebo Parser** (Kong et al., 2014)[8] An extension of Turbo Parser specialized to parse tweets. A new constraint is added to Turbo Parser's integer linear programming to ignore some Twitter tokens from parsing, but simultaneously uses them as parsing features.

- **Yara Parser** (Rasooli and Tetreault, 2015)[9] A transition-based parser that uses beam search training and dynamic oracle.

3.4.2 Data

We train all the parsers using two treebanks and test their robustness over two ungrammatical datasets.

3.4.2.1 Parser Training Data

We vary the types of training sources; the parsers are trained with the Penn Treebank (a treebank on news text) (Marcus et al., 1993) and Tweebank (a treebank on Tweets) (Kong et al., 2014). We

[4] seas.upenn.edu/~strctlrn/MSTParser/MSTParser.html
[5] nlp.stanford.edu/software/nndep.shtml
[6] github.com/tensorflow/models/tree/master/syntaxnet
[7] www.cs.cmu.edu/~ark/TurboParser
[8] github.com/ikekonglp/TweeboParser
[9] github.com/yahoo/YaraParser

choose Penn Treebank to be comparable with other studies, and Tweebank because it is a bit more like the test domain.

Penn Treebank (PTB)

We follow the standard splits of Penn Treebank, using section 2-21 for training, section 22 for development and 23 for testing. We transform bracketed sentences from PTB into dependency formats using Stanford Basic Dependency representation (De Marneffe et al., 2006) from Stanford parser v3.6. We assign POS tags to the training data using Stanford POS tagger (Toutanova et al., 2003) with ten-way jackknifing (with 97.3% accuracy).

Tweebank

Tweebank is a Twitter dependency corpus annotated by non-experts containing 929 tweets (Kong et al., 2014). Kong et al. (2014) used 717 of tweets for training and 201 for test[10]. We follow the same split in our experiments. We use pre-trained POS tagging model of Kong et al. (2014) (with 92.8% accuracy) over the tweets.

The elements in tweets that have no syntactic function (such as hashtags, URLs and emoticons) are annotated as unselected tokens (no tokens as the heads). In order to be able to use Tweebank in other parsers, we link the unselected tokens to the wall symbol (i.e. root as the heads). This assumption will generate more arcs from the root, but since we use the same evaluation setting for all the parsers, the results are comparable. We evaluate the accuracy of the trained parser on Tweebank with the unlabeled attachment F_1 score (same procedure as Kong et al. (2014)).

3.4.2.2 Robustness Test Data

To test robustness of parsers, we choose two domains of ungrammatical sentences that we discussed in Chapter 2: English learner and machine translation outputs. For fair comparison over test data, we automatically assign POS tags to the test data. When parsers are trained on PTB, we use Stanford POS tagger (Toutanova et al., 2003). When parsers are trained on Tweebank, we coarsen POS tags to be compatible with the Twitter POS tags using the mappings specified by Gimpel et al. (2011).

[10] `github.com/ikekonglp/TweeboParser/tree/master/Tweebank`

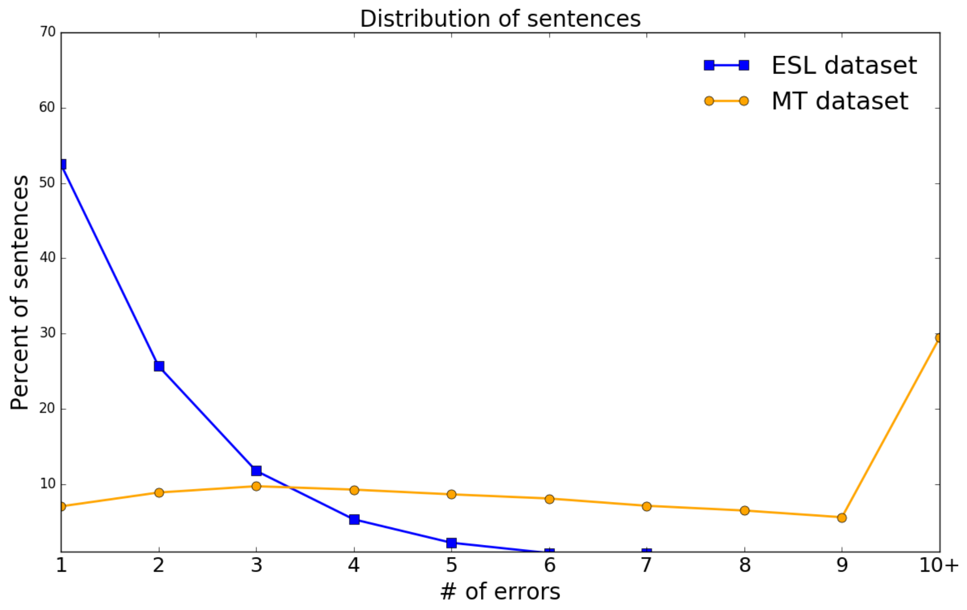
English as a Second Language corpus (ESL)

As discussed in Section 2.2.1.1, the ESL corpora contain writings of English as a second language learners and their corresponding error corrections. Given the errors and their corrections, we can easily reconstruct the corrected version of each ungrammatical ESL sentence. In this experiments, we use the First Certificate in English (FCE) dataset (introduced in Section 2.2.1.1) and from this corpus, we randomly select 10,000 sentences with at least one error; there are 4954 with one error; 2709 with two errors; 1290 with three; 577 with four; 259 with five; 111 with six; and 100 with 7+ errors.

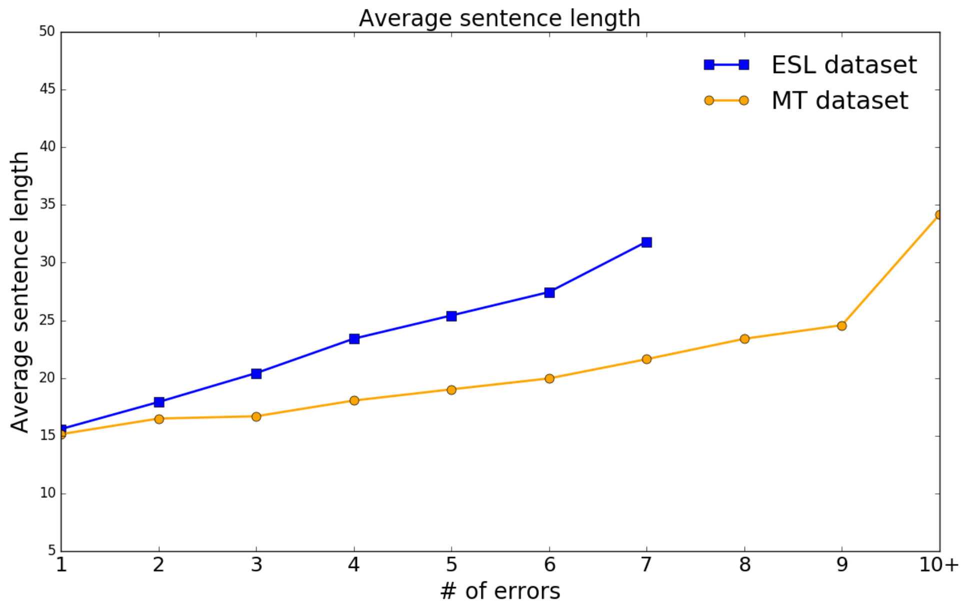
Machine Translation corpus (MT)

We also use machine translation outputs as anther domain of problematic sentences. From the LIG and LISMI's TRACE corpora (introduced in Section 2.2.2.1), we randomly select 10,000 sentences with at lease one edit distance (upon words) with their human-edited sentence. The distribution of the number of sentences with their edit distances from 1 to 10+ is as follows (beginning with 1 edit distance and ending with 10+): 674; 967; 1019; 951; 891; 802; 742; 650; 547; and 2752.

To better understand the sampled ESL and MT datasets, we further breakdown the sentences by the number of errors each contains. Figure 13 presents two graphs, plotting the number of sentences and the average sentence length against the number of errors for two datasets. In the ESL dataset, we observe that the number of sentences degrades with the increase of errors, which means most of the ESL sentences have only a few errors. While in the MT dataset, the number of sentences is constant by increasing the number of edits. The jump in the MT dataset when there are 10 or more errors shows that there are a considerable number of sentences that have more than 10 edits (2752 sentences). In terms of average sentence length, as number of errors increases, the average sentence length increases in both datasets. This is an intuitive observations, since longer sentences tend to have more errors. Note that, since there are very few ESL sentences with more than 7 errors, we do not plot their average sentence length.



(a) Distribution of sentences



(b) Distribution of sentence length

Figure 13: Some statistics of sampled ESL and MT datasets by number of errors.

3.4.3 Experimental Settings

In the robustness evaluation metric (Section 3.3), shared dependencies and error-related dependencies are detected based on alignments between words in the ungrammatical and grammatical sentences. We find the alignments in the ESL and MT data in a slightly different way. In the ESL dataset, in which the error words are annotated, the grammatical and ungrammatical sentences can easily be aligned. In the MT dataset, we use the TER (Translation Error Rate) tool (default settings)[11] to find alignments.

In our experiments, we present unlabeled robustness F_1 micro-averaged across the test sentences. We consider punctuations when parsers are trained with the PTB data, because punctuations can be a source of ungrammaticality. But we ignore punctuations when parsers are trained with the Tweebank data, because punctuations are not annotated in the tweets with their dependencies.

3.5 EXPERIMENTS

We have conducted a set of preliminary experiments using the proposed robustness metric to evaluate robustness of parsers in various conditions. This set of experiments aim to address the following questions given separate training and test data:

1. How do parsers perform on erroneous sentences? (Section 3.5.1)
2. To what extent is each parser negatively impacted by the increase in the number of errors in sentences? (Section 3.5.2)
3. To what extent is each parser negatively impacted by the interactions between multiple errors? (Section 3.5.3)
4. What types of errors are more problematic for parsers? (Section 3.5.4)

[11] www.cs.umd.edu/~snover/tercom

3.5.1 Overall Accuracy and Robustness

The overall performances of all parsers are shown in Table 2. Note that the Tweebo Parser's performance is not trained on the PTB because it is a specialization of the Turbo Parser, designed to parse Tweets. Table 2 shows that, for both training conditions, the parser that has the best robustness score in ESL domain has also high robustness for the MT domain. This suggests that it might be possible to build robust parsers for multiple ungrammatical domains. The training conditions do matter – Malt performs better when trained from Tweebank than from the PTB. In contrast, Tweebank is not a good fit with the neural network parsers due to its small size. Moreover, SNN uses pre-trained word embeddings and 60% of Tweebank tokens are missing.

Next, let us compare parsers within each train/test configuration for their relative robustness. When trained on the PTB, all parsers are comparably robust on ESL data, while they exhibit more differences on the MT data, and, as expected, everyone's performance is much lower because MT errors are more diverse than ESL errors. We expected that by training on Tweebank, parsers will perform better on ESL data (and maybe even MT data), since Tweebank is arguably more similar to the test domains than the PTB; we also expected Tweebo to outperform others. The results are somewhat surprising. On the one hand, the highest parser score increased from 93.72% (Turbo trained on PTB) to 94.36% (Malt trained on Tweebank), but the two neural network parsers performed significantly worse, most likely due to the small training size of Tweebank. Interestingly, although SyntaxNet has the lowest score on ESL, it has the highest score on MT, showing promise in its robustness.

3.5.2 Parser Robustness by Number of Errors

To better understand the overall results, we further breakdown the test sentences by the number of errors each contains. Our objectives are: (1) to observe the speed with which the parsers lose their robustness as the sentences become more error-prone; (2) to determine whether some parsers are more robust than others when handling noisier data.

Figure 14 presents four graphs, plotting robustness F_1 scores against the number of errors for all parsers under each train/test configuration. In terms of the parsers' general degradation of robustness, we observe that: 1) parsing robustness degrades faster with the increase of errors for

(a) Train on PTB §1-21

Parser	**UAS** PTB §23	**Robustness F_1** ESL	MT
Malt	*89.58*	93.05	76.26
Mate	**93.16**	93.24	77.07
MST	91.17	*92.80*	76.51
SNN	90.70	93.15	74.18
SyntaxNet	93.04	93.24	76.39
Turbo	92.84	**93.72**	**77.79**
Tweebo	-	-	-
Yara	93.09	93.52	*73.15*

(b) Train on Tweebank$_{train}$

Parser	**UAF$_1$** Tweebank$_{test}$	**Robustness F_1** ESL	MT
Malt	77.48	**94.36**	80.66
Mate	76.26	91.83	75.74
MST	73.99	92.37	77.71
SNN	*53.4*	88.90	*71.54*
SyntaxNet	75.75	*88.78*	**81.87**
Turbo	79.42	93.28	78.26
Tweebo	**80.91**	93.39	79.47
Yara	78.06	93.04	75.83

Table 2: Parsers performance in terms of accuracy and robustness. The best result in each column is given in bold, and the worst result is in italics.

the MT data than the ESL data; 2) training on the PTB led to a more similar behavior between the parsers than when training on Tweebank; 3) training on Tweebank does help some parsers to be more robust against many errors.

In terms of relative robustness between parsers, we observe that Malt, Turbo and Tweebo parsers are more robust than others given noisier inputs. The SNN parser is a notable outlier when trained on Tweebank due to insufficient training examples.

3.5.3 Impact of Error Distances

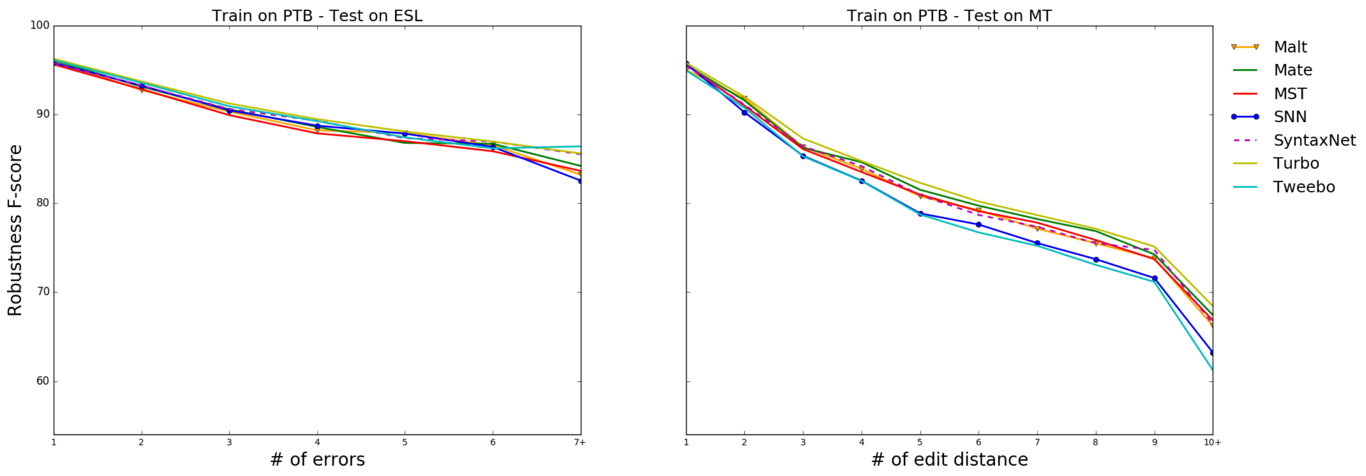
This experiment explores the impact of the interactivity of errors. We assume that errors have more interaction if they are closer to each other, and less interaction if they are scattered throughout the sentence. We define "near" to be when there is at most 1 word between errors and "far" to be when there are at least 6 words between errors.[12] We expect all parsers to have more difficulty on parsing sentences when their errors have more interaction, but how do the parsers compare against each other? We conduct this experiment using a subset of sentences that have exactly three errors; we compare parser robustness when these three errors are near to each other with the robustness when the errors are far apart.[13]

Table 3 presents the results as a collection of shaded bars. This aims to give an at-a-glance visualization of the outcomes. In this representation, all parsers with the same train data and *test domain* (including both the *near* and *far* sets) are treated as one group. The top row specifies the lowest score of all parsers on both test sets; the bottom row specifies the highest score. The shaded area of each bar indicates the *relative robustness* of each parser with respect to the lowest and highest scores of the group. An empty bar indicate that it is the least robust (corresponding to the lowest score in the top row); a fully shaded bar means it is the most robust (corresponding to the highest score in the bottom row). Consider the left-most box, in which parsers trained on PTB and tested on ESL are compared. In this group[14], Yara (near) is the least robust parser with a score of $F_1 = 87.3\%$, while SNN (far) is the most robust parser with a score of $F_1 = 93.4\%$; as expected, all parsers are less robust when tested on sentences with near errors than far errors, but

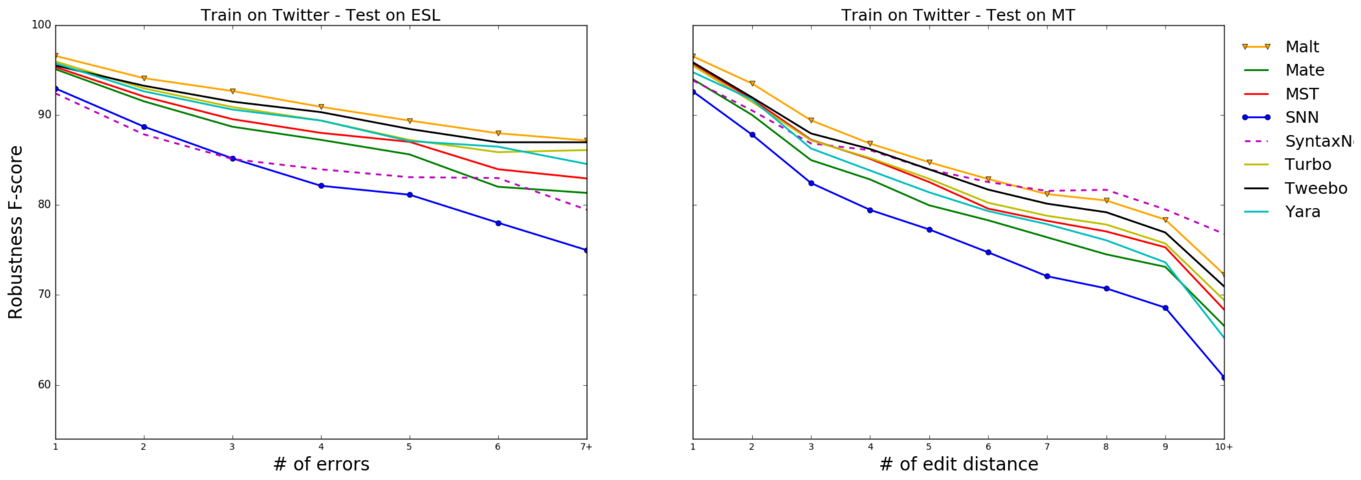
[12]We heuristically chose 1 and 6 numbers based on the amount of sentences that we have in each group.

[13]We chose the subset of sentences with three errors since we had considerable amount of sentences with exactly three errors.

[14]As previously explained, Tweebo is not trained on PTB, so it has no bars associated with it.



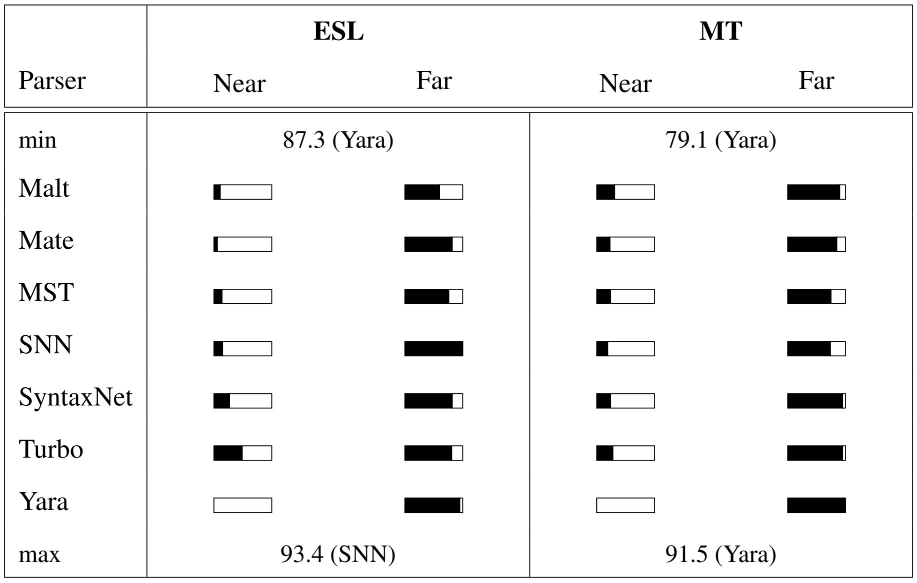
(a) Train on PTB §1-21



(b) Train on Tweebank$_{train}$

Figure 14: Variation in parser robustness as the number of errors in the test sentences increases.

(a) Train on PTB §1-21



(b) Train on Tweebank$_{train}$

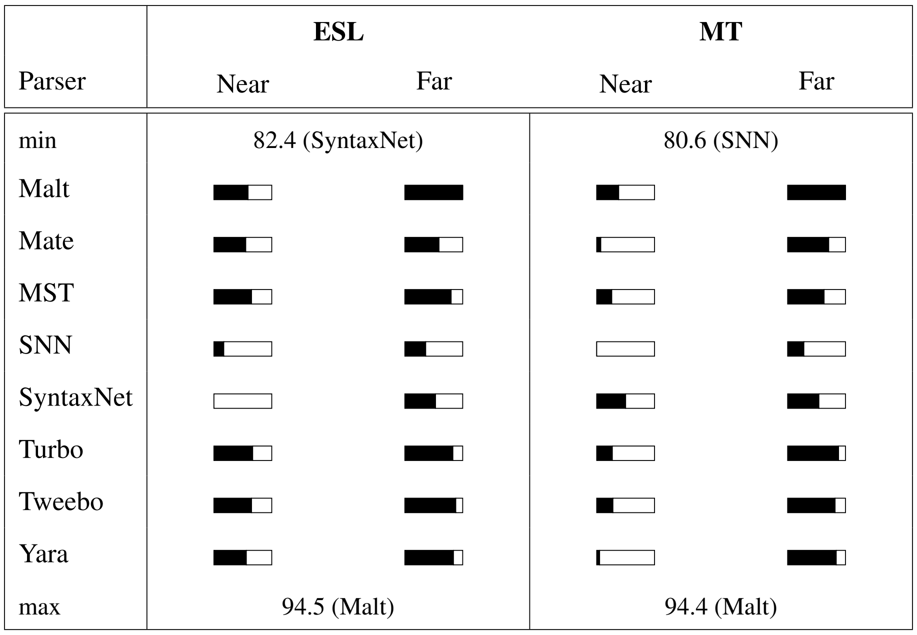


Table 3: Parser performance on test sentences with 3 near and 3 far errors. Each box represents one train/test configuration for all parsers and error types. The bars within indicate the level of robustness scaled to the lowest score (empty bar) and highest score (filled bar) of the group.

they do exhibit relative differences: Turbo parser seems most robust in this setting. Turbo parser's lead in handling error interactivity holds for most of the other train/test configurations as well; the only exception is for Tweebank/MT, where SyntaxNet and Malt are better. Compared to ESL data, near errors in MT data are more challenging for all parsers; when trained on PTB, most are equally poor, except for Yara, which has the worst score (79.1%) even though it has the highest score when the errors are far apart (91.5%). Error interactivity has the most effect on Yara parser in all but one train/test configuration (Tweebank/ESL).

3.5.4 Impact of Error Types

In the following experiments, we examine the impact of different error types. To remove the impact due to interactivity between multiple errors, these studies use a subset of sentences that have only one error. Although all parsers are fairly robust for sentences containing one error, our focus here is on the relative performances of parsers over different error types: We want to see whether some error types are more problematic for some parsers than others.

3.5.4.1 Impact of grammatical error types

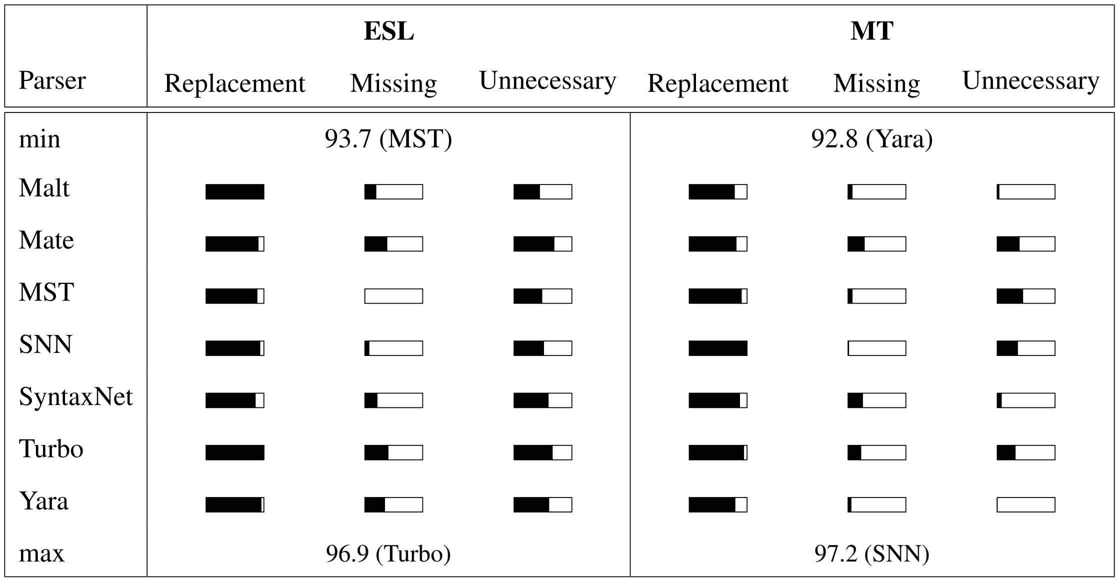
The three main grammar error types are replacement (a word need replacing), missing (a word missing), and unnecessary (a word is redundant). Our goal is to see whether different error types have different effect on parsers. If yes, is there a parser that is more robust than others?

As shown in Table 4, replacement word errors are the least problematic error type for all the parsers; on the other hand, missing word errors are the most difficult error type for parsers. This finding suggests that a preprocessing module for correcting missing and unnecessary word errors may be helpful in the parsing pipeline.

3.5.4.2 Impact of error word category

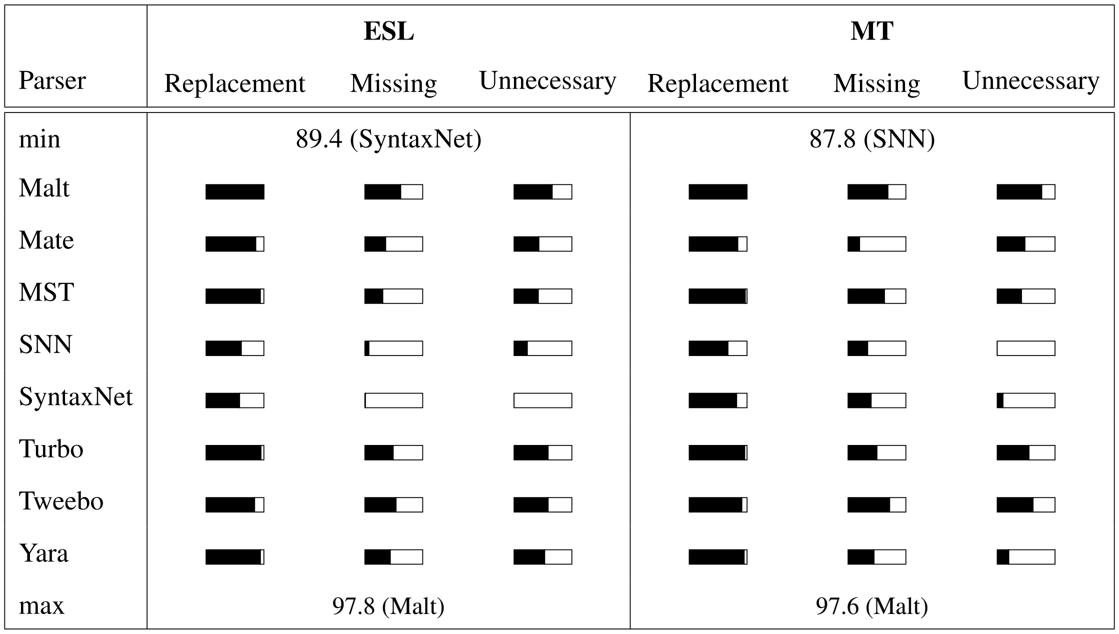
Another factor that might affect parser performances is the class of errors; for example, we might expect an error on a preposition to have a higher impact (since it is structural) than an error on an adjective. We separate the sentences into two groups: error occurring on an open- or closed-class word. We expect closed-class errors to have a stronger negative impact on the parsers because they contain function words such as determiners, pronouns, conjunctions and prepositions.

(a) Train on PTB §1-21



Parser	ESL Replacement	ESL Missing	ESL Unnecessary	MT Replacement	MT Missing	MT Unnecessary
min	93.7 (MST)			92.8 (Yara)		
Malt	■■■■■	■□□□□	■■□□□	■■■■□	□□□□□	□□□□□
Mate	■■■■□	■■□□□	■■■□□	■■■■□	■□□□□	■■□□□
MST	■■■■□	□□□□□	■■□□□	■■■■□	□□□□□	■■□□□
SNN	■■■■□	□□□□□	■■□□□	■■■■■	□□□□□	■■□□□
SyntaxNet	■■■■□	■□□□□	■■■□□	■■■■□	■□□□□	□□□□□
Turbo	■■■■■	■■□□□	■■■□□	■■■■□	■□□□□	■■□□□
Yara	■■■■□	■■□□□	■■■□□	■■■■□	□□□□□	□□□□□
max	96.9 (Turbo)			97.2 (SNN)		

(b) Train on Tweebank$_{train}$



Parser	ESL Replacement	ESL Missing	ESL Unnecessary	MT Replacement	MT Missing	MT Unnecessary
min	89.4 (SyntaxNet)			87.8 (SNN)		
Malt	■■■■■	■■■□□	■■■□□	■■■■■	■■■□□	■■■■□
Mate	■■■■□	■■□□□	■■□□□	■■■■□	■□□□□	■■■□□
MST	■■■■□	■■□□□	■■□□□	■■■■□	■■■□□	■■□□□
SNN	■■■□□	□□□□□	■□□□□	■■■□□	■■□□□	□□□□□
SyntaxNet	■■■□□	□□□□□	□□□□□	■■■■□	■■□□□	□□□□□
Turbo	■■■■□	■■■□□	■■■□□	■■■■□	■■■□□	■■■□□
Tweebo	■■■■□	■■■□□	■■■□□	■■■■□	■■■■□	■■■□□
Yara	■■■■□	■■□□□	■■■□□	■■■■□	■■□□□	■□□□□
max	97.8 (Malt)			97.6 (Malt)		

Table 4: Parser robustness on sentences with one grammatical error, each can be categorized as a replacement word error, a missing word error or an unnecessary word error.

Table 5 shows results. As expected, closed-class errors are generally more difficult for parsers. But when parsers are trained on PTB and tested on MT, there are some exceptions: Turbo, Mate, MST and Yara parsers tend to be more robust on closed-class errors. This result corroborates the importance of building grammar error correction systems to handle closed-class errors such as preposition errors.

3.5.4.3 Impact of error semantic role

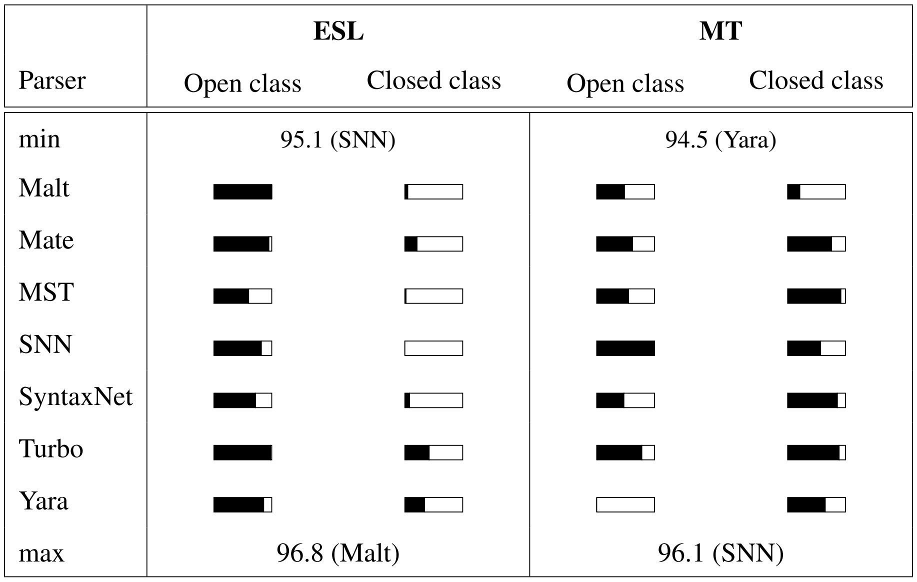
An error can be either in a verb role, an argument role, or no semantic role. We extract semantic role of the error by running Illinoise semantic role labeler (Punyakanok et al., 2008) on corrected version of the sentences. We then obtain the role of the errors using alignments between ungrammatical sentence and its corrected counterpart.

Table 6 shows the average robustness of parsers when parsing sentences that have one error. For parsers trained on the PTB data, handling sentences with argument errors seem somewhat easier than those with other errors. For parsers trained on the Tweebank, the variation in the semantic roles of the errors does not seem to impact parser performance; each parser performs equally well or poorly across all roles; comparing across parsers, Malt seems particularly robust to error variations due to semantic roles.

3.6 CHAPTER SUMMARY

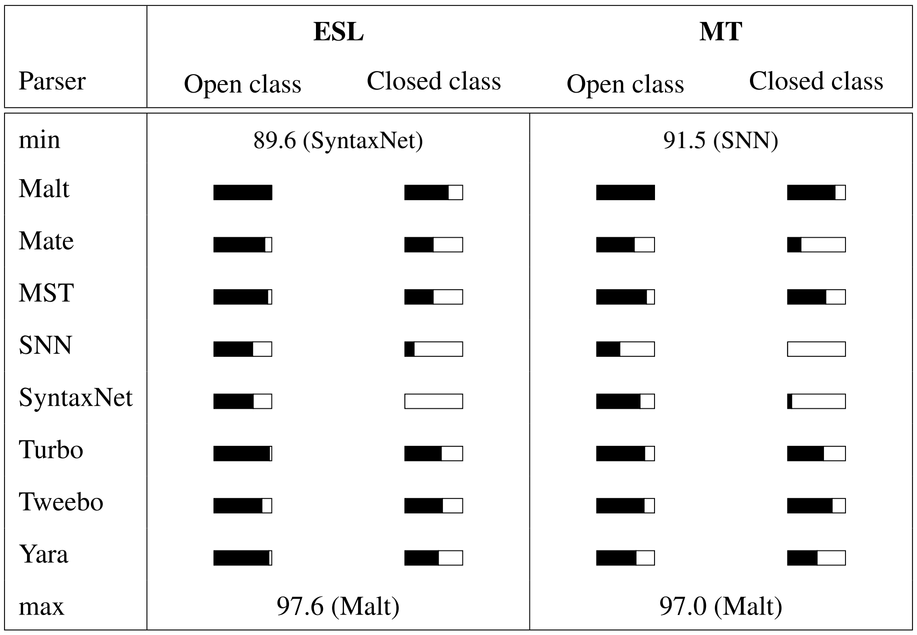
In this chapter, we have presented a set of empirical analyses on the robustness of processing ungrammatical text for several leading dependency parsers, using an evaluation metric designed for this purpose. We have found that parsers indeed respond differently to ungrammatical sentences of various types. Based on our experiments till now, we can make some recommendations for people who want to parse ungrammatical text in their applications. We recommend practitioners to examine the range of ungrammaticality in their input data (whether it is more like Tweets or has grammatical errors like ESL writings). If the input data contains noisy text more similar to Tweets (e.g. containing URLs and emoticons), Malt or Turbo parser may be good choices. If the input data is more similar to the machine translation outputs; SyntaxNet, Malt, Tweebo and Turbo parser

(a) Train on PTB §1-21



Parser	ESL Open class	ESL Closed class	MT Open class	MT Closed class
min	95.1 (SNN)		94.5 (Yara)	
Malt	█████	░░░░░	██░░░	█░░░░
Mate	█████	█░░░░	███░░	████░
MST	███░░	░░░░░	███░░	█████
SNN	████░	░░░░░	█████	███░░
SyntaxNet	████░	░░░░░	██░░░	████░
Turbo	█████	██░░░	████░	█████
Yara	████░	██░░░	░░░░░	███░░
max	96.8 (Malt)		96.1 (SNN)	

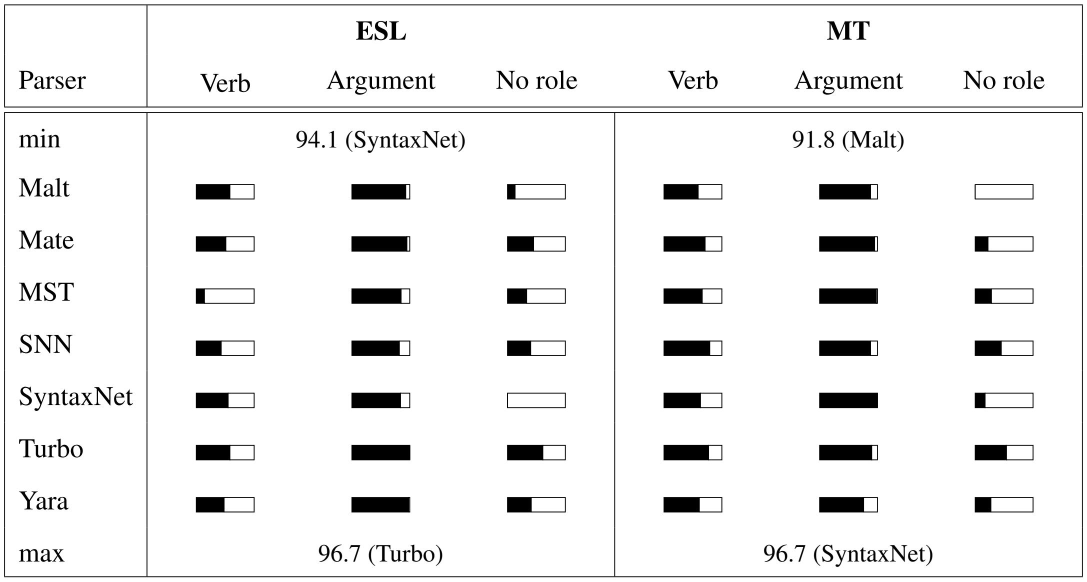
(b) Train on Tweebank$_{train}$



Parser	ESL Open class	ESL Closed class	MT Open class	MT Closed class
min	89.6 (SyntaxNet)		91.5 (SNN)	
Malt	█████	████░	█████	████░
Mate	████░	██░░░	███░░	█░░░░
MST	█████	██░░░	████░	███░░
SNN	███░░	█░░░░	██░░░	░░░░░
SyntaxNet	███░░	░░░░░	████░	░░░░░
Turbo	█████	███░░	████░	███░░
Tweebo	████░	███░░	████░	████░
Yara	█████	███░░	███░░	███░░
max	97.6 (Malt)		97.0 (Malt)	

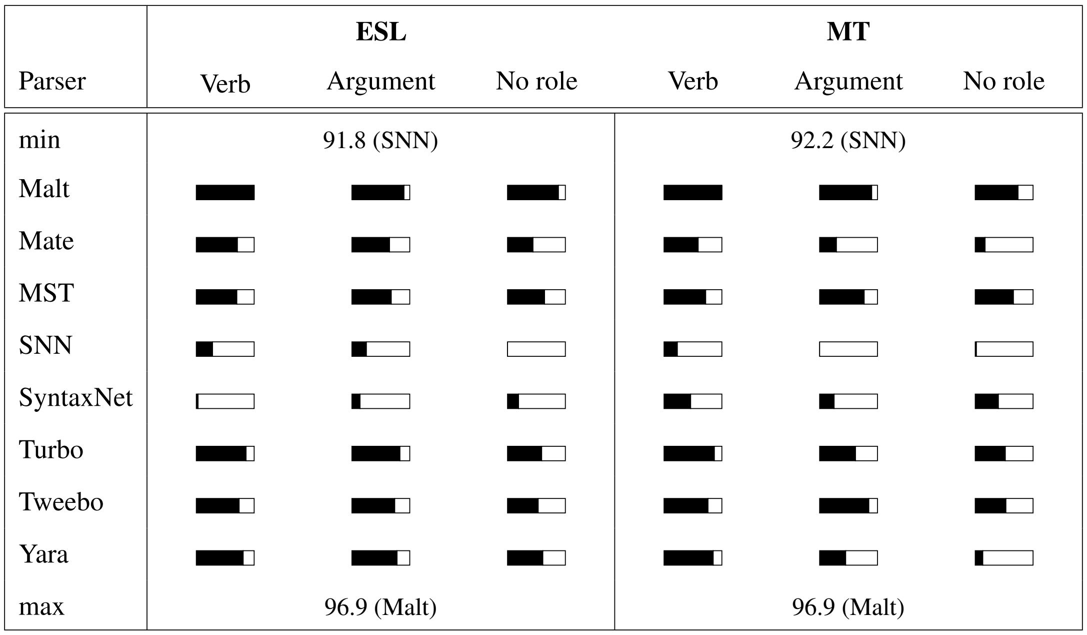
Table 5: Parser robustness on sentences with one error, where the error either occurs on an open-class (lexical) word or a closed-class (functional) word.

(a) Train on PTB §1-21



Parser	ESL Verb	ESL Argument	ESL No role	MT Verb	MT Argument	MT No role
min	94.1 (SyntaxNet)			91.8 (Malt)		
Malt	▬	▬	▬	▬	▬	▭
Mate	▬	▬	▬	▬	▬	▬
MST	▬	▬	▬	▬	▬	▬
SNN	▬	▬	▬	▬	▬	▬
SyntaxNet	▬	▬	▭	▬	▬	▬
Turbo	▬	▬	▬	▬	▬	▬
Yara	▬	▬	▬	▬	▬	▬
max	96.7 (Turbo)			96.7 (SyntaxNet)		

(b) Train on Tweebank$_{train}$



Parser	ESL Verb	ESL Argument	ESL No role	MT Verb	MT Argument	MT No role
min	91.8 (SNN)			92.2 (SNN)		
Malt	▬	▬	▬	▬	▬	▬
Mate	▬	▬	▬	▬	▬	▬
MST	▬	▬	▬	▬	▬	▬
SNN	▬	▬	▭	▬	▭	▭
SyntaxNet	▭	▬	▬	▬	▬	▬
Turbo	▬	▬	▬	▬	▬	▬
Tweebo	▬	▬	▬	▬	▬	▬
Yara	▬	▬	▬	▬	▬	▬
max	96.9 (Malt)			96.9 (Malt)		

Table 6: Parser robustness on sentences with one error where the error occurs on a word taking on a verb role, an argument role, or a word with no semantic role.

are good choices.

Furthermore, the results show that when ignoring erroneous parts of the ungrammatical sentences, parsers are doing reasonably well on finding syntactic structures of the remaining grammatical parts of the sentences. Therefore, an alternative reasonable approach to parse ungrammatical sentences would be to identify well-formed syntactic structures of those parts of the sentences that do make sense. The omission of the problematic structures may also help to prevent models that learn from syntactic structures from degrading due to incorrect syntactic analysis.

4.0 PARSE TREE FRAGMENTATION OF UNGRAMMATICAL SENTENCES

4.1 INTRODUCTION

The previous chapter showed that ungrammatical sentences present challenges for statistical parsers and the well-formed trees they produce may not be appropriate for these sentences. The experiments also showed that when ignoring erroneous parts of the ungrammatical sentences, parsers did reasonably well on finding syntactic structures of the remaining grammatical parts of the sentences. Therefore, in this chapter, we introduce a framework for reviewing the parses of ungrammatical sentences and extracting the coherent parts whose syntactic analyses make sense. We call this task *parse tree fragmentation*.

One approach for obtaining these partially completed structures is to use chunking (Abney, 1991; Sha and Pereira, 2003; Sun et al., 2008) (more details are given in Section 2.3.2) to identify recognizable low-level constituents, but this excludes higher-level complex structures. Instead, we propose to review the full parse tree generated by a state-of-the-art parser and identify the parts of it that are plausible interpretations for the phrases they cover. We call these isolated parts of the parse tree *fragments*, and the process of breaking up the tree, *parse tree fragmentation*.

In prior work, breaking up dependency arcs has been explored primarily in the form of vine parsing (Eisner and Smith, 2005; Dreyer et al., 2006), where a hard constrain on arc lengths considers only close words as modifiers (as discussed in Section 2.3.2.3). Our approach differs from vine parsing in that we do not have any limit on arc lengths; we identify the incorrect arcs with regard to grammar mistakes. Similar pruning approaches have been used in constituency parsing known as hedge parsing (Yarmohammadi et al., 2014). Hedge parsing behaves like vine parsing and discovers every constituent of length up to some span and prune other constituents. We also do not try to correct grammar mistakes (Sakaguchi et al., 2017), since error detection methods mostly work for

ESL error categories and non-ESL mistakes are not easily fixable; we aim to salvage well-formed syntactic structures form ungrammatical sentences in general for downstream applications that use syntactic relationships. Our task also differs from disfluency detection in spoken utterances, which focuses on removing extra fillers and repeated phrases (Honnibal and Johnson, 2014; Rasooli and Tetreault, 2013; Ferguson et al., 2015); ungrammatical sentences written by non-native speakers or generated by machines have a wider range of error types, such as missing phrases and incorrect phrasal ordering.

In the remaining of the chapter, we first define the parse tree fragmentation task in two syntactic representation (constituency and dependency) to indicate that our proposed framework can be generalized for both representations. We then present a methodology for creating gold standard data for training and evaluating parse tree fragmentation methods without using a task-specific annotated corpus.

4.2 A FRAMEWORK FOR PARSE TREE FRAGMENTATION

The goal of parse tree fragmentation is to take a sentence and possibly its tree as input and extract a set of partial trees that are well-formed and appropriate for the phrases they cover. To define this framework, we need to address some fundamental problems:

1. What kind of partial trees are considered to be well-formed and appropriate? (Section 4.2.1)
2. How do we obtain enough examples of appropriate ways to fragment the trees? (Two methods are proposed in Section 4.3)
3. How to automatically fragments the trees? (Three approaches are introduced in Chapter 5)
4. How should this task be evaluated? (Intrinsic and extrinsic evaluations are conducted in Chapters 6 and 7)

In this section, we address the first problem by defining the parse tree fragmentation task and discuss its challenges. We address the remaining problems in the next section and chapters by introducing the steps that we take to tackle the challenges.

4.2.1 Ideal Fragmentation

One factor that dictates how fragmentation should be done is how the fragments will be used in a downstream application. For example, a one-off slight grammar error (e.g., number agreement) probably will not greatly alter a parser output. For the purpose of information extraction, this type of slight mismatches should probably be ignored; for the purpose of training future syntax-based computational models, on the other hand, more aggressive fragmentation may be necessary to filter out unwanted syntactic relationships.

Even assuming a particular downstream application choice (sentential fluency judgment or semantic role labeling in our case), the ideal fragmentation may not be obvious, especially when the errors interact with each other. Consider the following output from a machine translation system:

The <u>members of the vote</u> <u>opposes</u> <u>any him</u>.

The sentence contains three problem areas (underlined):

i. *members of the vote*: unusual subject noun phrase
ii. *members ... opposes*: number disagreement between subject and the verb
iii. *any him*: unusual bigram

Figure 15 shows the parsers' outputs for this sentence in constituency and dependency syntactic representations. The parse trees look well-formed but they are inappropriate for the sentence. For example, both Stanford and SyntaxNet parsers group *any* and *him* into a clause to serve as the object of the main verb. In the constituency tree, this problem is more evident, since the Stanford parser assigns a sentential clause (*S*) to the *any him* phrase.

To tackle the inappropriate parse trees of ungrammatical sentences, we propose the parse tree fragmentation task which extracts a set of partial trees that are appropriate for the phrases they cover. But, which fragments should be salvaged from these parse trees? Someone who thinks the sentence says: *The members of the voting body oppose any proposal by him* might produce the coherent fragment sets shown in Figure 15. On the other hand, if they think it says: *No parliament members voted against him*, they might have opted to not keep the *PP* (*of the vote*) intact.

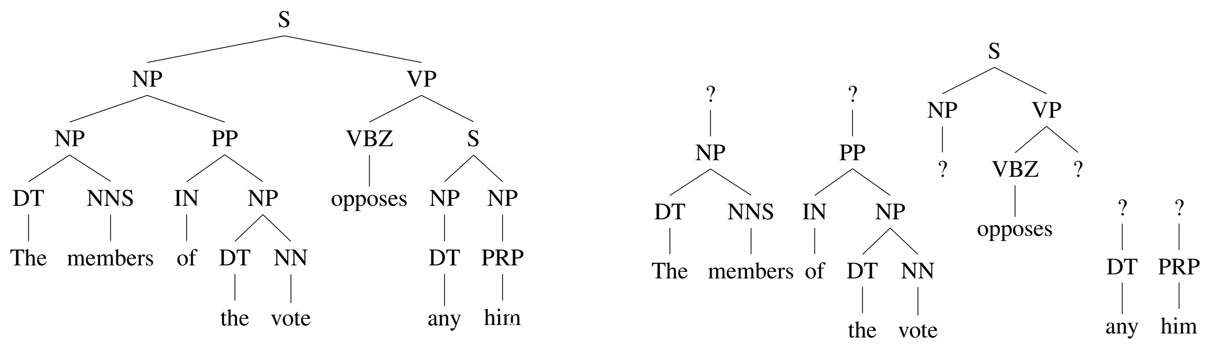
This example illustrates that fragmentation decisions are influenced by the amount of information we glean from the sentence. With only a sentence and an automatically generated tree for it, we may mentally error-correct the sentence in different ways. If we are also given an acceptable paraphrase for the sentence, the fragmentation task becomes more circumscribed because we now know the intended meaning. An example data source of this type is an MT evaluation corpus, which consists of machine-translated sentences and their corresponding human-translated references. Furthermore, if we not only have access to a closely worded paraphrase but also an explanation for each change, the fragmentation decisions are purely deterministic (e.g., whenever a phrase is recommended for deletion, the tree over it is fragmented). An example data source of this type is an ESL learner's corpus, which consists of student sentences and their detailed corrections.

4.2.2 Dependency Tree Fragmentation

The constituency tree fragmentation is analogous to dependency tree fragmentation (as shown in Figure 15), but it has other challenges because of the internal structure of trees. For example, in the constituency tree, the *any him* phrase contains three constituents: *S*, *NP* and *NP*. While in the dependency tree, it only has one dependency relations: *any* $\rightarrow$ *him*. Therefore, in this thesis, we focus on fragmenting dependency trees, whose head-modifier representation offers a clearer linguistic interpretation when dealing with ungrammatical sentences and a closer resemblance to semantic relations. In addition to dependency fragmentation described here, we have also explored fragmentation over constituency trees in Hashemi and Hwa (2016).

4.3 DEVELOPING A FRAGMENTATION CORPUS

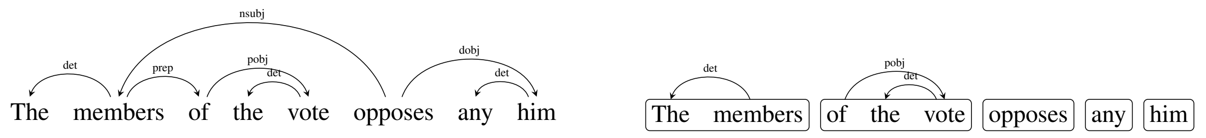
Our goal is to develop a sizable tree fragmentation gold standard corpus. Ideally, this corpus would be a collection of trees of ungrammatical sentences and their corresponding sets of tree fragments extracted by knowledgeable annotators who agree with each other. However, since the definition of an ideal fragmentation depends on multiple factors (e.g., the intended use and the context in which the original sentences were generated), this task is not well-suited for a large-scale



(i) Stanford parse tree

(ii) Coherent fragments

(a) Constituency tree fragmentation



(i) SyntaxNet parse tree

(ii) Coherent fragments

(b) Dependency tree fragmentation

Figure 15: Example of an ungrammatical sentence that gets a complete well-formed but inappropriate parse trees in two syntactic representations (right), and a set of coherent tree fragments that might be extracted from the full parse tree (left).

human annotation project. Instead, we propose to develop our fragmentation corpus by leveraging existing data sources previously mentioned (an ESL learner's corpus and an MT evaluation corpus). We exploit two types of parallel corpora to create our gold standard corpora by introducing two approaches: Pseudo gold fragmentation and Reference fragmentation.

4.3.1 Pseudo Gold Fragmentation (PGold)

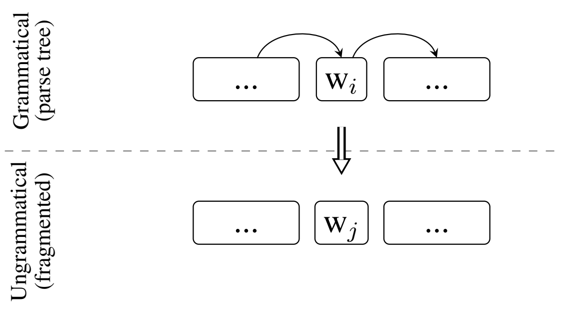
An ESL learner's corpus in which every sentence has been hand corrected by an English teacher is ideal for our purpose. We identified sentences that are marked as containing word-level mistakes: *unnecessary*, *missing* or *replacing* word errors. Given the positions and error types, a grammatical sentence can be reconstructed and reliably parsed. The parse tree of the grammatical sentence can then be iteratively fragmented according to the error types that occur in the original ungrammatical sentence. The resulting sets of fragments approximate an explicitly manually created fragmentation corpus; however, since a parser may make mistakes even on a grammatical sentence, we call these fragments *pseudo gold*.

We first parse the grammatical sentence with a state-of-the-art dependency parser. We then fragment it based on the errors in the original ungrammatical sentence. For each error, we apply the following procedure to the tree of grammatical sentence to reconstruct the ungrammatical sentence and its fragments:

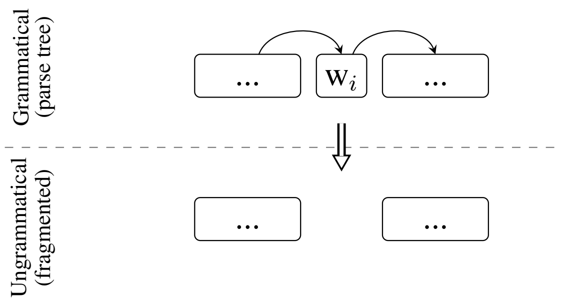
- Prune the dependency arcs based on the type of the error (see Figure 16):
  - If the error is a word replacement, prune the dependency arcs to and from the error word.
  - If the error is a missing word, remove the word and the dependencies to and from to it.
  - If the error is an unnecessary word, add the extra word as a separate fragment.
- Find the immediate right and left words of the error word in the sentence, if there is an arc to or from the right or left words that passes over the error word, prune it.

Figure 17 shows an example of PGold fragmentation for a sentence written by an English-as-a-Second Language (ESL) learner[1]. There are two grammar mistakes in the sentence: a missing comma and a phrase replacement word error ("for ever" should be replaced with "forever"). Our

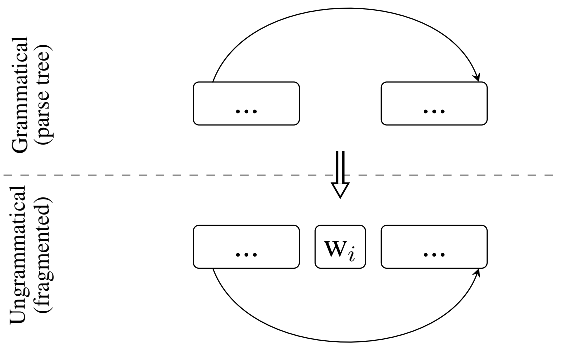
[1]Dependency tree is produced by SyntaxNet parser (Andor et al., 2016)



(a) Replacing word error

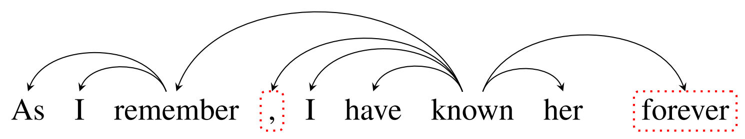


(b) Missing word error

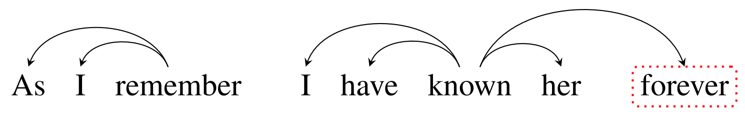


(c) Unncessary word error

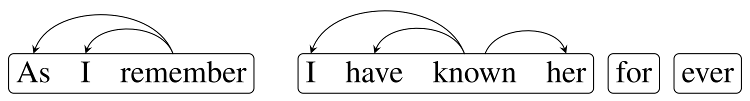
Figure 16: Creating pseudo gold fragments. The upper parts of figure are parse tree of grammatical sentences and the lower parts are their transformation after applying errors.



(a) Grammatical sentence and its parse tree.



(b) Reconstructing the ungrammatical sentence by applying the first error, missing comma.



(c) Reconstructing the ungrammatical sentence by applying the second error, replacement word error.

Figure 17: Example of PGold fragmentation of an ungrammatical sentence. There are two errors in the sentence: a missing comma and a replacement word error. Starting from the grammatical sentence and its parse tree, PGold reconstructs the ungrammatical sentence and its fragments.

goal is to identify the dependency arcs of the ungrammatical sentences that are related to grammar mistakes. Using the PGold procedure, the parse tree fragments of the ungrammatical sentence is iteratively constructed, given the position and type of errors.

4.3.2 Reference Fragmentation (Reference)

Even if we do not have detailed information about why certain parts of a sentence are problematic, we can construct an almost-as-good fragmentation if we have access to a fluent paraphrase of the original. We call this a *reference* sentence, borrowing the terminology from the MT community, where it is used to refer to human translations against which MT systems are evaluated. In a language tutoring scenario, the reference would be a teacher's revision of a student's original attempt. Given a parallel corpus of ungrammatical sentences and their grammatical versions, we first parse the ungrammatical sentence with a state-of-the-art dependency parser. Next, we find its grammar mistakes based on alignments between words in the ungrammatical and grammatical sentences. Then for each grammar mistake, we apply the following restrictive pruning rules (which might be modified depending on a downstream application):

- Prune the dependency arc to the error word.
- Prune all the dependency arcs from the error word.
- Find the immediate right and left words of the error word in the sentence, if there is an edge to or from the right or left words that passes over the error word, prune it.

Although these rules are restrictive, they simplify our argument for the use of tree fragments and, at the same time, they still help us to validate the usefulness of fragmentation in downstream applications. Figure 18 shows an example of the Reference method. In this example, the word "for" is not aligned, therefore the dependencies to and from it are pruned. The comma in the grammatical sentence is also a missing word error, thus the dependency arc from its left word that passes over the missing comma, "remember $\rightarrow$ known", is pruned.

4.3.3 Comparing PGold and Reference

While both PGold and Reference made use of additional information to create reliable tree fragments, they serve different purposes. PGold tree fragments represent the most linguistically plausi-

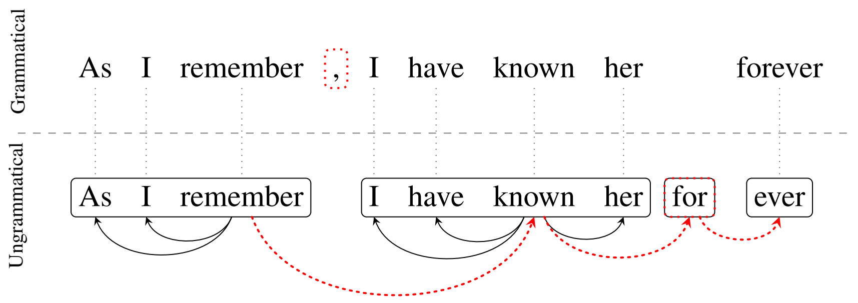


Figure 18: Example of Reference fragmentation of an ungrammatical sentence. The dotted red arcs are cut dependencies based on the two word error. It results four fragments.

ble interpretation of the original (ungrammatical) sentence because we can construct the intended well-formed sentence and obtain the fragments from its corresponding well-formed tree. In contrast, an automatic alignment between an original sentence and a reference sentence may not be as linguistically plausible (e.g., an error could be fixed via a substitution or via an insertion plus a deletion). Therefore, the Reference tree fragments are formed from the automatically parsed tree of the original sentence, and they represent an upperbound on what a real fragmentation algorithm could achieve. Thus, we are able to use Reference fragments to train automatic fragmentation algorithms.

4.4 CHAPTER SUMMARY

We have introduced parse tree fragmentation as a way to address the mismatch between ungrammatical sentences and statistical parsers that are not trained to handle them. We have defined the parse tree fragmentation framework on the dependency formalism with the goal of identifying and pruning the syntactic dependency arcs of the ungrammatical sentences that are related to the grammar mistakes. The result of breaking up the trees is a set of tree fragments that are linguistically appropriate for the phrases they cover. Since there is not a sizable corpus with gold standard annotations of tree fragments for ungrammatical sentence, we have devised methods for extracting gold standard tree fragments using evaluative parallel corpora available for other NLP applications. The gold standard corpus enables us to train and evaluate automatic fragmentation methods.

5.0 AUTOMATIC METHODS OF PARSE TREE FRAGMENTATION

5.1 INTRODUCTION

In this chapter, we propose some fragmentation strategies to automatically produce parse tree fragments for ungrammatical sentences. The goal of these approaches is to automatically identify and prune the syntactic dependency arcs of the ungrammatical sentences that are related to the grammar mistakes.

5.2 FRAGMENTATION METHODS

We propose three automatic methods of fragmentation by assuming the availability of a gold standard training corpus. In the first method, we propose a post-hoc process on the outputs of off-the-shelf parsers for the ungrammatical sentences; we then formulate this problem as a binary classification task to decide which arcs of a dependency tree should be cut. We also propose two fully end-to-end data-driven approaches to directly build the parse fragments for ungrammatical sentences. The methods jointly learn to parse and fragment ungrammatical sentences to avoid cascading parsers' errors on these sentences. In our second method, we adapt a parser with ungrammatical inputs by building a treebank of ungrammatical sentences. In the third proposed method, we cast the problem of parse tree fragmentation as a sequence-to-sequence mapping problem. Inspired by the recent works in neural network-based sequence-to-sequence learning (Sutskever et al., 2014; Bahdanau et al., 2014; Cho et al., 2014), we use a state-of-the-art LSTM-based recurrent neural network.

The automatic fragmentation methods are developed based on a parallel corpus of ungrammatical sentences and their corrections. Using this parallel corpus, we build the Reference corpus (described in Section 4.3.2) as the gold standard training corpus. We exploit Reference tree fragments, because they are formed from the automatically parsed tree of the ungrammatical sentences, thus they represent an upperbound on what a real fragmentation algorithm could achieve.

5.2.1 Classification-based Parse Tree Fragmentation (Classification)

As we saw in Chapter 3, when ignoring error-related dependency arcs of ungrammatical sentences, parsers are doing reasonably well on finding syntactic structures of the remaining grammatical parts of the sentences. Thereby, a straight-forward approach to automatically extract reliable parse tree fragments from ungrammatical sentences is to find the error-related dependency arcs. Along this line, we propose a post-hoc process to review the generated parse trees by off-the-shelf parsers. Given the generated parse trees, a system needs to discriminate between the right and wrong contexts from some head-modifier dependencies. We formulate this as a binary classification problem: for each dependency arc in the tree indicates whether the arc should be kept or cut. Using parse trees that were fragmented by the Reference method as examples, we train a Gradient Boosting Classifier (Friedman, 2001) that learns to fragment trees in a similar manner as Reference. The trained classifier can then make predictions on the branches of unseen parse trees. The tree fragments obtained in this post-hoc manner are referred to as *Classification*.

Because the number of kept arcs is far greater than the cut ones, when constructing the training set, we randomly sample equal number of the kept and cut arcs. The following features are extracted from each head-modifier dependency arc:

- Depth and height of the head and the modifier when the dependency tree is traversed in depth-first order. Figure 19 shows depth and height features for "known $\rightarrow$ for" arc in depth-first traversal of the dependency tree in Figure 18. The depth and height of the head word "known" are 2 and 3 respectively. The depth and height of the modifier word "for" are 3 and 2 respectively.
- Part-of-speech tags of the head, modifier, and the parent of the head word. For example in the Figure 19, for the arc of "known$\rightarrow$for" the POS tags of "known", "for", and "remember" are extracted.

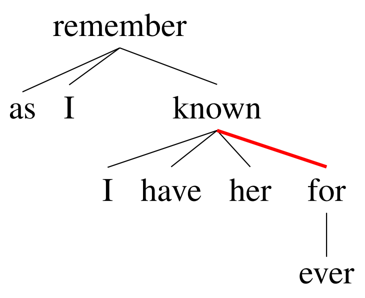


Figure 19: Depth and height features for the dependency arc of "known $\rightarrow$ for".

- Word bigrams and trigrams corresponding to the arc (as shown in Figure 20). Denoting w_h $(h = 1, 2, ..)$ as the head word and w_m as the modifier word, the bigram feature are calculated for the pairs of $w_h w_m$ ($w_m w_h$ if $m < h$), $w_{m-1} w_m$, and $w_m w_{m+1}$. The trigram features are calculated for the triples of $w_{m-1} w_m w_{m+1}$, $w_{m-2} w_{m-1} w_m$, and $w_m w_{m+1} w_{m+2}$. We use both raw counts and pointwise mutual information of the N-grams. To compute the N-gram counts, we use Agence France Press English Service (AFE) section of English Gigaword (Graff et al., 2003).

5.2.2 Parser Adaptation Parse Tree Fragmentation (Parser)

Parsing ungrammatical sentences can be considered as an instance of domain adaptation, in which the goal is to adapt a standard parser to accurately process the ungrammatical text (Foster et al., 2008). The ungrammatical text might be considered as the target domain that contains the language that is not covered by the parser's grammar. We propose to adapt parsers with ungrammatical sentences by building a treebank of these sentences and their parse tree fragments. In the following, we first briefly describe the approaches to collect data for parser domain adaptation. Next, we describe our proposed approach to create a treebank of ungrammatical sentences with the goal of building an end-to-end data-driven parse tree fragmentation method.

5.2.2.1 Parser Domain Adaptation

One of the challenges of parser adaptation is the lack of training data for the target domain. There-

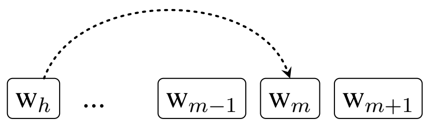


Figure 20: Word N-gram features for the dotted arc. Rectangles are words. Word bigrams associated to the dotted arc are: $w_h w_m$, $w_{m-1} w_m$ and $w_m w_{m+1}$.

fore, various approaches have been proposed to automatically label data in the target domain to use as training data. These approaches include self-training (McClosky et al., 2006), parser ensemble (Sagae and Tsujii, 2007; Baucom et al., 2013), selecting source sentences that are most similar to a target domain (McClosky et al., 2010), and building a treebank to retrain a parser (Foster, 2007; Kong et al., 2014; Foster et al., 2011b; Berzak et al., 2016). Foster (2007) builds a treebank for ungrammatical sentences by automatically generating errors to grammatical sentences. She iteratively applies the error creation procedure to the parse tree of the grammatical sentence to adapt it to the ungrammatical sentence. It is noteworthy to mention that our proposed pseudo gold fragmentation in Section 4.3.1 is inspired by her work in which we iteratively fragment parse trees according to error types. Kong et al. (2014) and Berzak et al. (2016) also introduce annotation guidelines and create treebanks for tweets and ESL writings, respectively. The sizes of these treebanks is small since they manually annotated sentences with their parse trees. Having the treebanks of ungrammatical sentences, they retrained parsers to make specialized parsers for the new domains.

The task of parse tree fragmentation can also be considered as an approach for parser adaptation with ungrammatical inputs. Therefore, we first introduce an approach to create a treebank of ungrammatical sentences and their parse tree fragments. We then train a new specialized fragmentation parser of ungrammatical sentences. One of the advantages of this approach is that it jointly learns to parse a sentence and fragment it considering grammatical errors that might exist in the sentence.

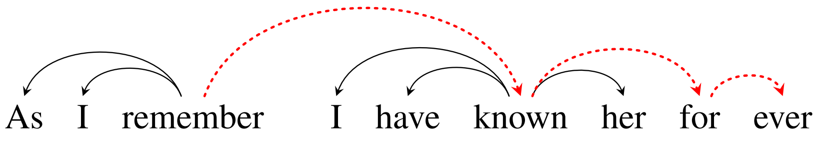


Figure 21: Example of a fragmented dependency tree. The dotted red arcs are cut dependencies based on the mistakes in the sentence.

5.2.2.2 Creating a Treebank of Tree Fragments

For the purpose of creating a treebank for ungrammatical sentences, we use their dependency trees that are fragmented by the Reference method. We adapt the dependency tree of the ungrammatical sentence by setting the head of the pruned arcs to be the wall symbol (i.e. root as the heads). The created treebank is in the CoNLL format. An example of the CoNLL based format for the dependency tree in Figure 21 with its pruned arcs is:

1	As	IN	3
2	I	PRP	3
3	remember	VB	0
4	I	PRP	6
5	have	VB	6
6	known	VB	0
7	her	PRP	6
8	for	IN	0
9	ever	RB	0

The first column shows the word number in the sentence; the second and the third columns contain the original words and their part-of-speech tags respectively. The last column (which is the focus of the parser to learn) shows the head of the word, i.e., the parent of the word which can be another word or the wall symbol. For example, the head of the first word “As” is the third word “remember”. In the standard CoNLL format of a dependency tree, each word should have a head and only one word in the sentence has the wall symbol as its head. For the purpose of adapting the parse trees of ungrammatical sentences with parse tree fragmentation, we assume that several words can have the wall symbol as their heads. To build the treebank, we first find the pruned arcs by the gold standard method. Next, we set the head of the pruned arc to be the wall symbol. For instance, in Figure 21, the arc "remember$\rightarrow$ known" is cut; therefore the head of the "known" is set to be 0 in the CoNLL format.

Using this new ungrammatical treebank that are created by the Reference method as examples, we train a statistical state-of-the-art parser that learns to prune dependency arcs in a similar manner as Reference. The trained parser can then both parse and prune error-related arcs on the unseen input sentences. We retrain SyntaxNet parser (Andor et al., 2016) with this ungrammatical treebank, and the obtained tree fragments in this manner are referred to as *Parser*.

5.2.3 Sequence-to-Sequence Parse Tree Fragmentation (seq2seq)

Many tasks in natural language processing can be casted as finding an optimal mapping from a source sequence to a target sequence including machine translation (Bahdanau et al., 2014), sentence compression (Filippova et al., 2015), grammar error correction (Schmaltz et al., 2016), dialogue systems (Serban et al., 2015), image or video captioning (Venugopalan et al., 2015; Xu et al., 2015). Theoretically, Recurrent Neural Networks (RNN) were always a potential tool to be used for learning a complex and highly non-linear seq2seq mapping. However, due to the problem of vanishing and exploding gradient, RNNs were far away from being practical. Recent advancements of deep structure RNNs are based on using Long Short-Term Memory (LSTM) (Hochreiter and Schmidhuber, 1997) units, addressing the gradient vanishing and the gradient exploding problem; therefore RNNs have rapidly become a versatile tool in natural language processing.

We also formulate the parse tree fragmentation task as finding an optimal sequence-to-sequence mapping, in which the source sequence is simply the ungrammatical input sentence and the target sequence is a linearized one-to-one mapping of the associated dependency tree with pruned arcs. Similar to the Parser method, the seq2seq method jointly parse and fragment ungrammatical sentences to avoid cascading parsers' errors on these sentences. In the following, for the sake of completeness, we first briefly describe the idea of sequence-to-sequence learning with deep neural networks. Next, we describe how we represent the tree fragments in a linear form as the target sequence of the seq2seq problem. The tree fragments obtained with sequence-to-sequence learning

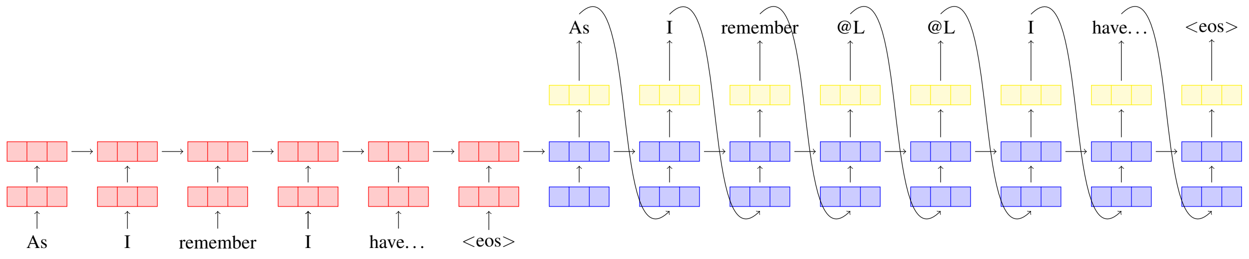


Figure 22: Schematic view of seq2seq model for parse tree fragmentation. The input words are first mapped to word vectors and then fed into a recurrent neural network (RNN). The final time step initializes an output RNN, upon seeing the <eos> symbol.

are referred to as *seq2seq*.

5.2.3.1 Seq2Seq Using Deep Neural Nets

We follow the dominant approach to train a seq2seq framework, which employs conditional language model and a cross-entropy loss function to maximize the conditional likelihood of a successive target word in the target sequence given the the input sequence and a history of target words. Following the past practice of the state-of-the-art seq2seq deep neural network models, in our network architecture, we use a stack of LSTM recurrent networks to encode the input sequence (or to be more accurate, a word embedding of the input sequence) into a latent representation that would be useful in finding the target sequence. Another stack of LSTM recurrent neural networks is used to decode the encoded latent representation of the input sequence to the target output sequence. For the training, in each step, the error signal generated by the cross-entropy loss function will be back-propagated through the network for tuning the weights to minimize the corresponding empirical risk on a batch of data. Figure 22 shows the schematic view of our neural arc pruning seq2seq model on our running example of Figure 21. More detailed information about the seq2seq deep neural network models can be found in Sutskever et al. (2014) and Wiseman and Rush (2016).

The deep neural RNN based seq2seq models require an effective representation for the input and the output to yield good performance (Vinyals et al., 2015a). We therefore utilize an interleaved arc-standard transition actions to represent the arc pruned dependency trees, that is described in the following sections.

5.2.3.2 Sequence Representation of a Fragmented Dependency Tree

We treat parse tree fragmentation as a seq2seq task by attempting to map from an input sentence to a linear form of arc pruned dependency tree. Using the ungrammatical sentences and their dependency trees that are pruned by the Reference method, we can train a seq2seq model. But the challenge is to represent arc pruned dependency trees in their linear forms. To tackle this problem, we follow the representation of Wiseman and Rush (2016) to linearize dependency trees, by inserting arc-standard reduce actions (Nivre, 2004) interleaved with the sentence words. Table 7 illustrates an example of arc-standard representation of a parse tree from the initial configuration (when the buffer contains the sentence and stack is empty) to a terminal one (when the buffer is empty and the stack contains only one word which will be connected to the ROOT symbol). To represent a parse tree, the arc-standard system defines three types of transition actions:

- `Shift`: moves the first word in the buffer to the top of the stack.
- `Left-arc`: adds an arc from the first word to the second word in the stack and removes the second word in the stack.
- `Right-arc`: adds an arc from the second word to the first word in the stack and removes the first word in the stack.

A dependency tree can be represented with a unique set of arc-standard actions. For example, the third column of Table 7 shows the set of actions for the dependency tree of Figure 21. This representation is particularly beneficial for our task, since each dependency arc is equivalent to a `Left-arc` or `Right-arc` action, hence we can annotate the pruned arcs accordingly. The last column of Table 7 shows the generated output sequence with annotated fragmented arcs. In particular, we try to map the input sentence to the output sequence:

Input: As I remember I have known her for ever

Output: As I remember @L @L I have known @L @L her @R for ever @RCUT @RCUT @RCUT

We use unlabeled arcs and show the actions with @L as the `Left-arc` action, and @R as the `Right-arc` action. The pruned arc is denoted by @LCUT or @RCUT action whether it was originally a `Left-arc` or a `Right-arc` action. The `Shift` actions are also replaced with the sentence words.

A trained seq2seq model with this representation would be able to prune error-related arcs of an ungrammatical sentence while parsing the remaining grammatical parts of the sentence. Another strength of this seq2seq model is that it learns the output vocabularies as well, even though we do not constraint the output to obey the stack constraint to use the same vocabulary of the input sentence.

In order to evaluate the seq2seq method, we then convert back the output of seq2seq which is in the form of interleaved arc-standard actions to a CoNLL format of dependency tree (similar to the example in the previous section).

Alternative Representation

We have also linearized the dependency parse trees by traversing them in depth-first, similar to the representation introduced in Vinyals et al. (2015b) for constituency trees. As the input representation, we both considered giving a sentence and also giving the linear form of the full dependency tree. As the output representation, we represented the fragmented dependency tree by marking the beginning and end of fragments with brackets; we also considered both keeping the vocabularies in the output and also replacing vocabularies with "X" (to simplify the task for the model to only learn the fragmented parts not the vocabularies). But the seq2seq model learned with these representation were not performing well, thus we do not report them in the dissertation. Here is an example of the the linearized form of the dependency trees that we tried but did not work:

Input: (remember As I known (I have her for (ever)))

Output: { (remember As I { known (I have her { for } ({ ever })) }) }

Buffer	**Stack**	**Action**	**Sequence**
As I remember I have known her for ever			
I remember I have known her for ever	As	`Shift`	As
remember I have known her for ever	As I	`Shift`	I
I have known her for ever	As I remember	`Shift`	remember
I have known her for ever	As remember	`Left-arc`	@L
I have known her for ever	remember	`Left-arc`	@L
have known her for ever	remember I	`Shift`	I
known her for ever	remember I have	`Shift`	have
her for ever	remember I have known	`Shift`	known
her for ever	remember I known	`Left-arc`	@L
her for ever	remember known	`Left-arc`	@L
for ever	remember known her	`Shift`	her
for ever	remember known	`Right-arc`	@R
ever	remember known for	`Shift`	for
	remember known for ever	`Shift`	ever
	remember known for	`Right-arc`	@RCUT
	remember known	`Right-arc`	@RCUT
	remember	`Right-arc`	@RCUT

Table 7: An example of the transition sequence of the arc-standard actions for the dependency tree of Figure 21. The last column shows the generated output sequence with annotated fragmented arcs. We use this linear form of arc pruned dependency trees to train the seq2seq model.

5.3 COMPARISON OF FRAGMENTATION METHODS

The three proposed fragmentation methods employ different strategies: one uses a binary classifier to distinguish the error-related dependency arcs, the second one utilizes parser technology by creating a fragmented treebank, and the third method exploits the recent advances in neural networks to jointly learns to parse and fragment ungrammatical sentences. We summarize the strengths and weaknesses of each fragmentation method in Table 8.

The proposed methods can be used by the practitioners based on their available ungrammatical data. If they have a small set of ungrammatical sentences as the training data and a high quality dependency parser, the Classification method may be a good choice. If they have a reasonably high quality parallel data and can tune a dependency parser, Parser method may be a good choice. Finally, if they have a large amount of parallel data and access to a good computational power, seq2seq method would be a better choice (we will discuss the performances of the methods in the next chapters).

5.4 CHAPTER SUMMARY

We have proposed three practical methods for extracting parse tree fragments of the ungrammatical sentences: a classifier-trained method, a deterministic parser retraining method, and a sequence-to-sequence method. These methods can be trained with the gold standard tree fragments to automatically produce tree fragments of the unseen ungrammatical sentences. Each of the devised fragmentors has specific characteristics and can be adapted to other domains based on the available resources.

Method	Strength	Weakness
Classification	• A couple of thousand sentences is enough for training.	• It needs feature engineering. • It post-processes parser outputs, so parser's errors might propagate.
Parser retraining	• Jointly learns to parse and fragment. • Theoretically any dependency parser can be trained.	• It needs high quality or a huge amount of training data. • In practice, parsers' implementations matter. Because they perform differently even though they have the same underlying design.
seq2seq	• Jointly learns to parse and fragment. • No need for feature engineering. • No need for high quality annotated data, even noisy training data would be helpful.	• It needs a huge amount of parallel training data which might not be available for some ungrammatical domains.

Table 8: Comparison of the proposed automatic fragmentation methods.

6.0 EMPIRICAL EVALUATION OF PARSE TREE FRAGMENTATION

6.1 INTRODUCTION

We introduced parse tree fragmentation framework to review parsers of ungrammatical sentences and identify well-formed syntactic structures of the parse trees that do make sense. We also proposed three automatic fragmentation methods that learns to fragment using the gold standard fragmentation methods. In this chapter, we perform a set of empirical evaluations to determine the performance of the automatic fragmentation methods with respect to the gold standard fragments. We evaluate tree fragments of two domains with ungrammatical sentences: writings of English-as-a-Second Language (ESL) learners and the MT outputs.

6.2 EVALUATION OF PARSE TREE FRAGMENTATION

The typical approach to evaluate NLP tasks is to compare the outputs of automatic systems against manually annotated gold standards. Therefore, in order to evaluate parse tree fragmentation methods, we seek a collection of gold standard fragments for ungrammatical sentences. However, as we discussed in Section 4.3, the fragmentation task is not well-suited for a large-scale human annotation project because the definition of an ideal fragmentation depends on many factors. Thus, instead we created near gold fragmentation corpora using existing data sources (more details in Chapter 4). In this chapter, we aim to evaluate the automatic fragmentation methods by comparing them to the gold fragments. This type of evaluation task is called *intrinsic evaluation* and it will tell us how closely an automatic tree fragmentation method might approach the gold fragments. In the next chapter, we will evaluate the potential uses of tree fragments in downstream applications which is called *extrinsic evaluation*. It will tell us whether the fragmentation is helpful, by evaluating the downstream applications once with fragmentation and once without it.

6.3 EXPERIMENTAL SETUP

6.3.1 Data

The experiments that we conduct in this thesis are over two domains of ungrammatical sentences: English as a second language learners and machine translation outputs. We choose datasets for which the corresponding correct sentences are available (or easily reconstructed); thus, given these parallel corpora of ungrammatical sentences and their grammatical versions, we can deterministically build the gold standard fragments. In this section, we discuss the data that we use for both this chapter and the next chapter.

6.3.1.1 English as a Second Language corpus (ESL)

We use English learners corpora that contain ungrammatical sentences and their corresponding error corrections. Given the location and type of the errors, a corrected version of each ungrammatical ESL sentence can be reconstructed. For example, in a sentence "He talk with a friend" the teacher would annotate that "talk" should be replaced by "talks" because it has the wrong number agreement. In most cases, knowing the errors and their corrections makes it possible for us to determine the appropriate fragments. However, some corrections are more complicated, involving phrase-to-phrase replacement due to multiple problems. For example, suppose a teacher recommended replacing "have a talk" with "talked". This edit involves both a semantic shift as well as a tense change. On a more micro-level, should the corrected verb "talked" be aligned with the original noun "talk" (because they are more semantically similar) or the original verb "have" (because they are more syntactically similar)? Due to ambiguity in the phrase-to-phrase corrections, we filter them out in experiments.

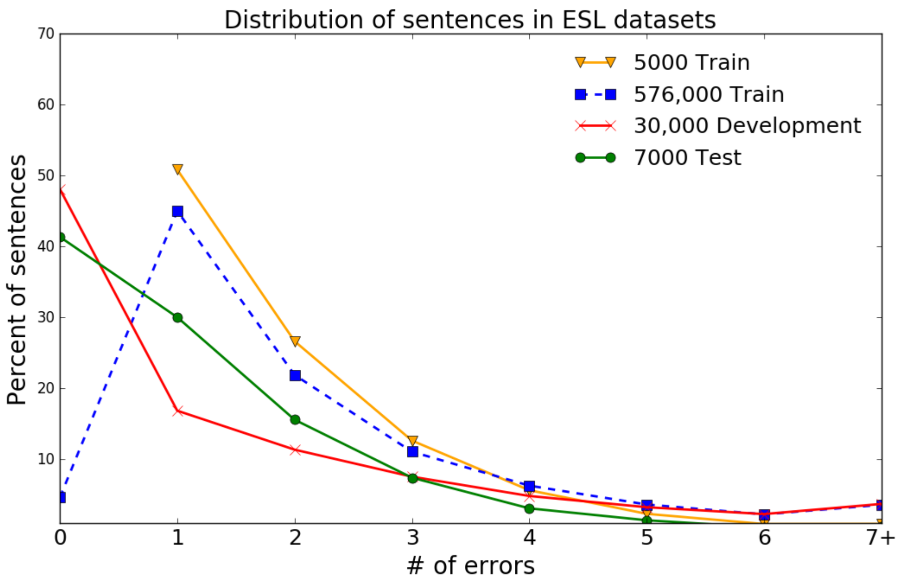
Our sampled ESL datasets

For the purpose of training and testing the fragmentation methods, we sample non-overlapping sets from the ESL corpora that we introduced in Section 2.2.1.1. The following datasets will serve as the training, development and test sets in our experiments:

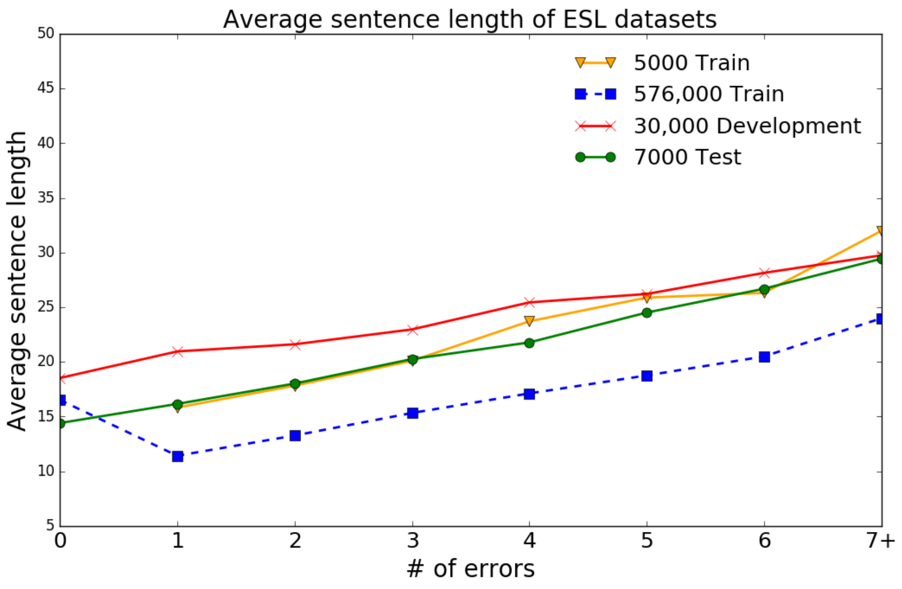
- **5000 Train**: From the FCE corpus, we randomly select 5000 sentences with at least one error for training the Classification fragmentation method.
- **576,000 Train**: From all the three corpora, we randomly select 576,238 sentences as the training set of Parser and seq2seq methods.
- **30,000 Development**: From the FCE and NUCLE datasets, we then randomly select non-overlapping 30,000 sentences as the development set of Parser and seq2seq methods.
- **7000 Test**: From the FCE corpus, we create a non-overlapping dataset for the intrinsic and extrinsic evaluation. It consists of 7000 sentences and is representative of the corpus's error distribution; there are 2895 sentences with no error; 2103 with one error; 1092 with two errors; and 910 with 3+ errors.

To better understand the sampled ESL datasets, we further breakdown the sentences by the number of errors each contains. Figure 23 presents two graphs, plotting the number of sentences and the average sentence length against the number of errors for all the sampled datasets. In terms of number of sentences (as shown in Figure 23(a)), we observe that the number of sentences degrades with the increase of errors, which means most of the ESL sentences have only a few errors. The four datasets have similar behavior, the only exception is the few number of sentences with no errors in the 576,000 Train dataset. This happens because 576,000 Train dataset is sampled over a million sentences with at least one errors and only a few thousand sentences without any errors.

In terms of average sentence length (Figure 23(b)), as number of errors increases, the average sentence length increases. This is an intuitive observations, since longer sentences tend to have more errors. We also observe that ESL sentences of 576,000 Train dataset are on average shorter than other datasets. This shows a characteristic of the EFCAMDAT dataset which contains submitted sentences to an online website; it might happen because students tend to write shorter sentences on websites than on exams.



(a) Distribution of number of ESL sentences. For example, 41% sentences of the 7000 Test dataset sentences have no errors and 30% of sentences have 1 error.



(b) Distribution of ESL sentence length.

Figure 23: Some statistics of sampled ESL datasets by number of errors.

6.3.1.2 Machine Translation corpus (MT)

Unlike the ESL corpus, in the MT corpus, we only have access to the human-edited sentences. We cannot create PGold fragmentation (Section 4.3.1) for the MT data because we are not certain about positions or types of the errors. We can only build Reference fragments (Section 4.3.2) for MT by comparing the parse tree of the bad sentence with that of the good sentence, making splitting point decisions on the parse tree of the bad sentence.

Human-targeted Translation Edit Rate (HTER) score

In our experiments on the MT corpus, we use the HTER (Human-targeted Translation Edit Rate) score (Snover et al., 2006) as the fluency score of MT outputs. This score is also used in Workshop on Statistical Machine Translation (WMT)[1] for the sentence-level quality estimation task. Thus, we use this score to be consistent with the machine translation works. HTER is defined as the minimal rate of edits needed to change the machine translation to its manually post-edited version:

$$\text{HTER} = \frac{\text{\# of edits}}{\text{\# of words in the grammatical sentence}}$$

HTER ranges between 0 and 1 (0 when no word is edited and 1 when all words are edited). We use TER (default settings)[2] to compute HTER scores.

Our sampled MT datasets

We sample the following non-overlapping datasets from the MT corpora that we introduced in Section 2.2.2.1 as the training, development and test sets:

- **4000 Train**: From the LIG corpus, we randomly select 4000 sentences with HTER score more than 0.1 fro training the Classification fragmentation method.
- **9000 Train**: From the two corpora, we randomly select 9000 sentences as training data for the Parser fragmentation method. This training data has overlap with the 4000 Train dataset.
- **2000 Development**: From the two corpora, we randomly select 2000 sentences as development data for training the Parser fragmentation method. This training data does not have

[1] http://www.statmt.org/wmt17/quality-estimation-task.html
[2] http://www.cs.umd.edu/~snover/tercom/

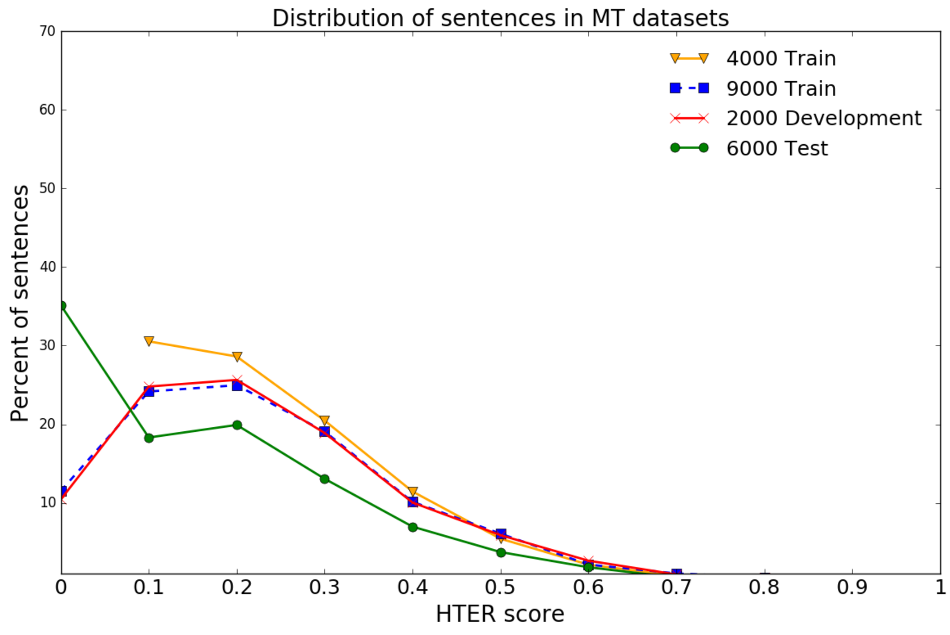
overlap with the 9000 Train datset, but has overlap with the 4000 Train dataset.

- **6000 Test**: From the LIG corpus, we create a non-overlapping dataset for the intrinsic and extrinsic evaluation. It consists of 6000 sentences and is representative of the corpus's error distribution. The HTER score of 2109 sentences are within $[0, 0.1)$; 1099 sentences within $[0.1, 0.2)$; 1195 sentences within $[0.2, 0.3)$; 784 sentences within $[0.3, 0.4)$; and 813 sentences have scores more than 0.4.

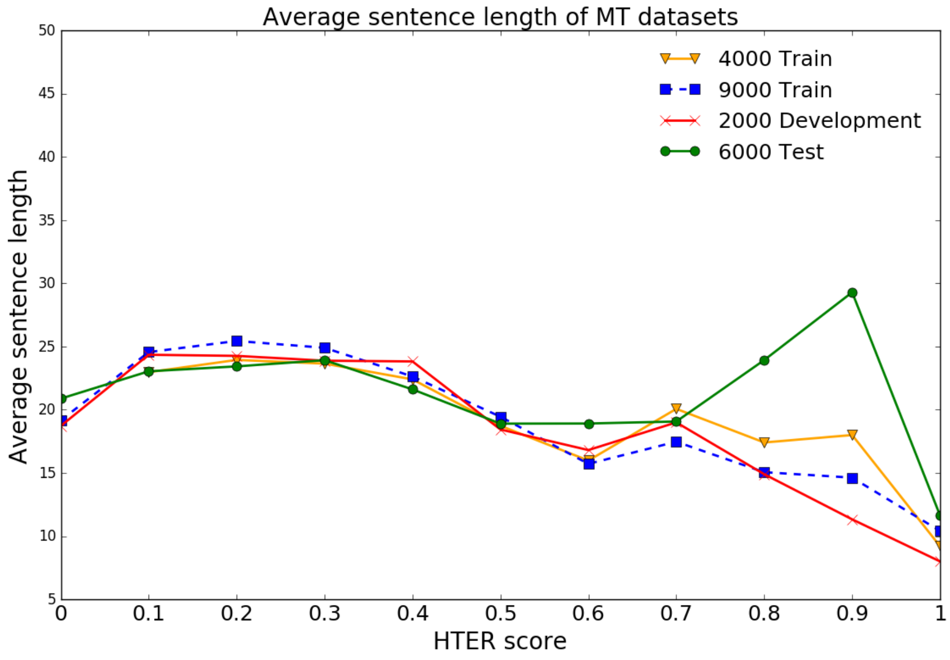
To train the seq2seq method, we need a huge amount of parallel data. However, to our knowledge, there are not available any other larger MT corpora containing English translations and their human-edited sentences. Therefore, we use the trained model over ESL data and test it on the MT data. This experimental setup helps to investigate how we can transfer learning on different ungrammatical domains.

We further analyze the sampled MT datasets by separating the sentences with their HTER scores. Figure 24 shows two graphs, plotting the number of sentences and the average sentence length against the HTER score for all the sampled datatsets. In terms of number of sentences (as shown in Figure 24(a)), we observe that the number of sentences degrades with the increase of HTER score, which means most of the MT outputs have only a few edits with respect to their total number of words. In terms of average sentence length (Figure 24(b)), as HTER score increases, the average sentence length gradually decreases. Although there are few MT outputs with large HTER scores, these few sentences are on average shorter than other sentences. This is because HTER score shows the ratio of edits with respect to the number of words in the sentence, thus a short sentence with only a few edits will have a high HTER score.

In order to better understand the MT datasets, we also breakdown the sentences by the number of edits each contains. Using the raw number of edits helps us to compare the MT datasets with the ESL datasets in which we investigate the number of errors in the sentences. Figure 25 presents two graphs, plotting the number of sentences and the average sentence length against the number of edits for all the sampled datasets. In terms of number of sentences (as shown in Figure 25(a)), we observe that the number of sentences is almost the same with the increase of number of edits. The statistics indicates that there are quite a large number of MT outputs that have many edits; for instance around 30% of the MT sentences have more than 8 edits. While in the ESL datasets, less than 1% of sentences have more than 8 errors. In terms of average sentence length (Figure 25(b)),

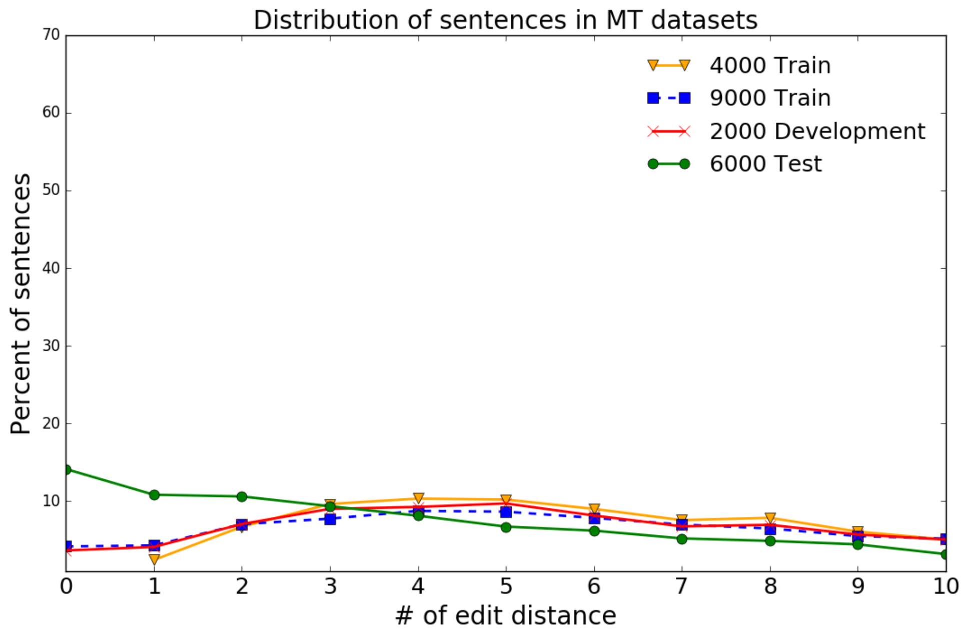


(a) Distribution of MT sentences as the HTER score.

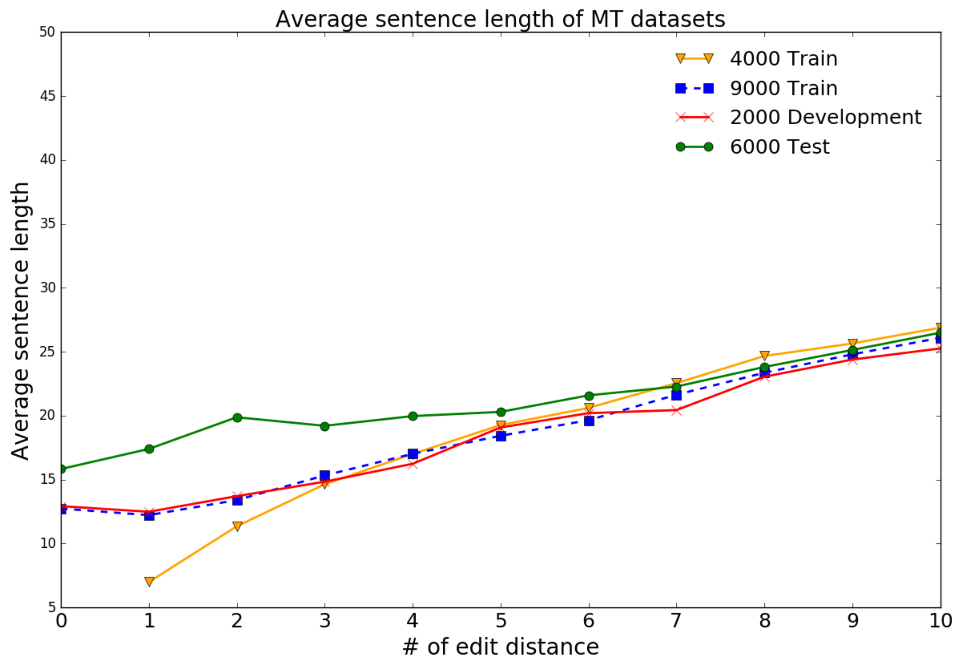


(b) Distribution of MT sentence length as the HTER score.

Figure 24: Some statistics of sampled MT datasets by HTER score.



(a) Distribution of MT sentences as the number of edit distance.



(b) Distribution of MT sentence length as the number of edit distance.

Figure 25: Some statistics of sampled MT datasets by number of edits.

as number of edit distance increases, the average sentence length increases. This is an intuitive observations, since longer sentences tend to have more edits.

6.3.2 Experimental Tools

The pre-trained SyntaxNet POS tagger and parser (Andor et al., 2016)[3] is used to generate dependency parses for all the sentences.

6.3.2.1 Reference Settings

To create the Reference training data, all the grammatical and ungrammatical sentences are first parsed. Then grammar mistakes are detected based on alignments between words in the ungrammatical and grammatical sentences. We use the TER (Translation Error Rate) tool (default settings)[4] to find alignments. Then the Reference method are run over the trees to detect the arcs that should cut.

6.3.2.2 Classification Settings

For the Classification binary classification, we train the standard Gradient Boosting Classifier (Friedman, 2001) in the scikit-learn toolkit (Pedregosa et al., 2011).[5] We tune Gradient Boosting parameters with a 3-fold cross validation on the training data: `learning_rate` over the range $\{0.0001 \dots 100\}$ by multiples of 10 and `max_depth` over the range $\{1 \dots 5\}$.

6.3.2.3 Parser Retraining Settings

We create a treebank of our ESL data using the Reference method (as described in Section 5.2.2). We then train the SyntaxNet parser (Andor et al., 2016) which is a transition-based neural network parser and use its globally normalized training with default parameters. We train the parser on the train set and pick the model with the best unlabeled attachment score on the development set.

[3] github.com/tensorflow/models/tree/master/syntaxnet/syntaxnet/models/parsey_mcparseface

[4] www.cs.umd.edu/~snover/tercom

[5] We have also tried SVMs with LibLinear toolkit (Fan et al., 2008), but gradient boosting learners obtained the best results.

6.3.2.4 seq2seq Settings

To train the sequence-to-sequence model, we use OpenNMT[6] (Klein et al., 2017) package, which is a neural machine translation system utilizing the Torch mathematical toolkit. In our implementation of seq2seq RNNs, we use 2-layer LSTMs with 750 hidden units in each layer both for decoding and encoding modules. We train the network with a batch size of 48 and a maximum sequence length of 62 and 123 for the source and target sequences, respectively. The sequence length is chosen in a way to cover the 5 standard deviations range from the mean of the length of the source and target sequence. The parameters of the model are uniformly initialized in $[-0.1, 0.1]$, and the L2-normalized gradients are constrained to be ≤ 5 to prevent the gradient exploding effect. In the training phase, the learning rate schedule starts at 1 and halves the learning rate after each epoch beyond epoch 10, or once the validation set perplexity no longer improved. We train the network for up to 30 epochs choosing the model with the lowest perplexity on the validation set as the final model.

6.3.3 Evaluation Metrics

One way to evaluate an automatic arc pruning method is to compare its resulting dependency tree against the Reference tree. We use three metrics for this comparison: the usual dependency tree attachment score, accuracy of the cut arcs, and an adapted version of F-score for set-to-set comparison. Another way of evaluating whether the fragmentation methods make sense is to perform an extrinsic evaluation (which will be discussed in the next chapter).

6.3.3.1 Unlabeled Attachment Score (UAS)

The usual dependency tree attachment score is used to compare the resulting dependency trees against Reference trees by calculating unlabeled attachment score (UAS). UAS calculates the percentage of words that have the correct head:

$$\text{Unlabeled Attachment Score (UAS)} = \frac{\text{\# of words with correct heads}}{\text{Total number of words}}$$

The head could be either another word or the wall symbol (i.e. a cut arc). Therefore, UAS

[6] `github.com/opennmt/opennmt`

measures the *total* performance of an automatic method considering both kept and cut arcs.

6.3.3.2 Accuracy of Cut Arcs

To measure how well a fragmentation method cuts arcs, we evaluate its accuracy only on the cut arcs. Precision and recall (and F-score) are calculated as the percentage of correct pruned dependency arcs in the resulting parse tree and the Reference tree respectively:

$$\text{Precision}_{cut} = \frac{\text{\# of correct cut arcs}}{\text{Total number of cut arcs by an automatic fragmentation method}}$$

$$\text{Recall}_{cut} = \frac{\text{\# of correct cut arcs}}{\text{Total number of cut arcs by the Reference method}}$$

$$\text{F-score}_{cut} = 2 \times \frac{\text{Precision}_{cut} \times \text{Recall}_{cut}}{\text{Precision}_{cut} + \text{Recall}_{cut}}$$

6.3.3.3 Set-2-Set F-score

Another way of evaluating an automatic fragmentation method is to compare its resulting fragments against the gold standard fragments by adapting the usual tree-to-tree precision and recall metrics for set-to-set. First, each fragment of the candidate set is mapped to a fragment of the gold standard set with which it has a maximum number of shared arcs. (If there are two candidate fragments but only one gold fragment, both candidates would be mapped to the same gold fragment.) Second, precision and recall (and F-score) are calculated as the number of shared arcs between all the mapped fragments divided by the total number of arcs in the candidate and the gold fragment sets respectively:

$$\text{Precision}_{set-to-set} = \frac{\text{\# of shared arcs}}{\text{Total number of arcs in the automatic fragment sets}}$$

$$\text{Recall}_{set-to-set} = \frac{\text{\# of shared arcs}}{\text{Total number of arcs in the gold fragment sets}}$$

$$\text{F-score}_{set-to-set} = 2 \times \frac{\text{Precision}_{set-to-set} \times \text{Recall}_{set-to-set}}{\text{Precision}_{set-to-set} + \text{Recall}_{set-to-set}}$$

We report macro-averaged precision, recall and F-score over the test sentences.

6.4 EVALUATION

To measure how well the proposed automatic fragmentation methods perform, we have conducted a series of intrinsic evaluations. We first validate each fragmentation method using standard measures for parsing and classification; we then compare its tree fragments against those produced by other fragmentation methods.

6.4.1 Performance of Each Fragmentation Method

Given an ungrammatical sentence, our proposed automatic fragmentation methods produce dependency parse trees for it with some pruned arcs. Table 9 shows the performance of the produced dependency trees against the Reference trees with the unlabeled attachment score (UAS) over both ESL and MT sentences. The *No cut* method serves as a baseline that does not break any tree; thus, its UAS shows the similarity of the complete trees with the Reference fragments. Its results corroborate the fact that the Reference method cuts a small percentage of the dependency arcs; 84.6% and 65.14% of the dependency arcs are not pruned in the ESL and MT domains respectively.

In the FCE dataset, the UAS suggests that the dependency trees produced by the seq2seq method are more similar to the Reference trees than the Classification and the Parser methods'. It shows that the seq2seq method not only learns to parse but also learns to prune dependency arcs in a completely automatic regime. Evaluating the accuracy of only the pruned arcs also suggests that the seq2seq method is making reasonable decisions in opting to cut an arc while parsing the sentence.

In the MT dataset, the Classification method produces the most similar fragments to the Reference method's. The seq2seq method is not performing well, it is because it is trained on the ESL data and tested on the MT data. In order to further investigate the cross domain effect of the training data, we apply the Classification method when trained on ESL over the test sentences of

(a) ESL dataset

Automatic Method	UAS	Accuracy of cut arcs		
		Precision$_{cut}$	Recall$_{cut}$	F-score$_{cut}$
Classification	61.36	0.35	0.79	0.48
Parser	63	0.35	0.53	0.42
seq2seq	82.4	0.71	0.57	**0.63**
No cut	84.6	-	-	-

(b) MT dataset

Automatic Method	UAS	Accuracy of cut arcs		
		Precision$_{cut}$	Recall$_{cut}$	F-score$_{cut}$
Classification	60.67	0.49	0.66	**0.56**
Parser	50.55	0.43	0.70	0.54
seq2seq (trained on ESL)	58.82	0.68	0.16	0.26
Classification (trained on ESL)	62.23	0.51	0.52	0.51
No cut	65.14	-	-	-

Table 9: Performance of automatic fragmentation methods by comparing their resulting dependency trees against Reference fragmented trees as their training data. The *No cut* method serves as a baseline and does not break any tree.

MT. Even though the seq2seq method is not performing well when transferring the models, the Classification method trained on ESL is doing well and somewhat comparable to the Classification method trained on MT. The reason is that the seq2seq method is conservative in pruning the arcs (it has high precision$_{cut}$ but low recall$_{cut}$), while the Classification method is pruning more arcs (it has high recall$_{cut}$) in both domains; therefore, since the MT sentences have more errors than ESL sentence, the Reference method cuts more dependency arcs, as a result the Classification method that prunes more arcs is showing more similarity to the Reference in this cross domain setup. These results suggest that the difference between the training and testing data, and the characteristics of the fragmentation method (e.g. whether it is conservative in pruning) are important factors in transferring the models.

6.4.2 Performance of the Classification Method

The Classification method runs a binary prediction model over parse tree arcs, deciding whether to keep an arc or cut it. The ground-truth labels come from the Reference fragments. We performed a 10-fold cross validation for the two domains of ESL and MT. Note that while the Classification training data is balanced, the test data is not; thus, a baseline of never cutting any arc would result in a high classification accuracy (84% on ESL and 65% on MT). To take the skewed class distribution into account, we evaluate classifiers with the AUC measure (the area under the receiver operating characteristic curve) (Hanley and McNeil, 1982). AUC estimates how probable it is that a classifier might give a higher rank to a randomly cut-arc compared to a randomly not-cut-arc. In our experiments, the AUC of the Classification on ESL and MT is 0.75 and 0.63 respectively whereas the AUC of the baseline (cutting no arc) is 0.5 for both. The AUC of the Classification when trained on the ESL data and tested on the MT data is also 0.61. The AUC scores suggest that Classification method is making reasonable decisions, opting to cut an arc when it is certain.

6.4.3 Evaluation of Tree Fragmentation Methods

In the next experiment, we evaluate the fragmentation methods by how well their resulting tree fragments match the gold tree fragments. To perform the comparison, we use an adapted version of the usual precision and recall metrics for set-to-set (as described in Section 6.3.3). Table 9(a) summarizes the comparison of different fragmentation methods over the ESL dataset in terms of their average number of fragments, average fragment size, and F-score against PGold and Reference fragments. We see that the Reference fragments are the most similar to PGold. This validates our choice of using Reference fragments as the training data for automatic fragmentation methods. The average number and size of fragments indicate how much the method fragments the tree in comparison with the gold fragments. We see that the Classification method over-prune the dependency trees; as a result, it shows less similarity to the Reference. On the other hand, the Parser method is cautious in breaking the trees which results in fewer fragments. One reason is that the SyntaxNet is a transition-based parser which is designed to assign root as the head to the last remaining words in the stack. Even though we train the parser with a large treebank of ungrammatical sentences with multiple words with root as their heads, the parser still tends not to prune arcs. This result suggests that some adaptations may be necessary for the parser; one possible modification is to add a new action to the transition-based dependency parser that marks pruned arcs without removing the modifiers from the stack (because we need the modifiers to obtain the internal syntactic structure of fragments).

The set-2-set F-score similarity of seq2seq to Reference is 0.83, which indicates it has learned useful signals from the Reference method. But, the seq2seq has on average fewer fragments; which shows it prunes less arcs than the Reference method. The results of Tables 8(a) and 9(a) highlight that the seq2seq is conservative on pruning the error-related arcs but when it makes decisions on pruning an arc, it is almost certain.

Table 9(b) compares the fragmentation methods over the MT dataset. The Classification and the Parser methods are making more fragments than the Reference method. The seq2seq is producing much fewer fragments for the MT sentences since it is trained on the ESL data in which it is learned to make fewer fragments. On the other hand, the Classification trained on the ESL data is relatively breaking the trees into right number of fragments; the fragments even show higher similarity to the Reference. This results suggests the helpfulness of the transfer learning in the MT domain for the Classification method. In this thesis, throughout the experiments we perform the cross domain analysis over the seq2seq and the Classification methods to compare and observe the transfer learning behaviour in different experimental setups.

(a) ESL dataset

Method	Avg. #of Fragments	Avg. Size of Fragments	set-2-set P/R/F_1 to PGold	set-2-set P/R/F_1 to Reference
PGold	3.51	8.61	-	-
Reference	3.51	8.60	0.97/0.97/0.97	-
Classification	7.29	2.40	0.89/0.57/0.66	0.90/0.57/0.67
Parser	1.8	13.62	0.75/0.81/0.76	0.77/0.82/0.77
seq2seq	2.92	9.36	0.84/0.83/**0.82**	0.85/0.85/**0.83**
No cut	1	16.46	0.75/0.88/0.8	0.76/0.89/0.81

(b) MT dataset

Method	Avg. #of Fragments	Avg. Size of Fragments	set-2-set P/R/F_1 to Reference
Reference	9.66	5.36	-
Classification	12.96	2.09	0.71/0.57/0.60
Parser	15.61	2.38	0.63/0.37/0.41
seq2seq (trained on ESL)	2.29	18.70	0.54/0.72/0.59
Classification (trained on ESL)	9.80	2.88	0.67/0.64/**0.62**
No cut	1	24.82	0.52/0.76/0.60

Table 10: Similarity of fragmentation methods with gold fragments.

Comparing the two domains of ESL and MT, we see several differences. First, the Reference method produces more fragments in the MT data than the ESL data. This comes from the fact that MT outputs contain more edits than ESL sentences; thus, the Reference method breaks more the MT parse trees. Second, the Parser method behaves differently in the MT than ESL; it makes very few fragments in the ESL data, while it makes many fragments in the MT data. One reason is that the sizes of their training data are different. The parser is trained over 576k ESL sentences and 11k MT sentences, respectively. Thus, it suggests that as the number of training data grows, the parser tends to cut less arcs. To further study the behaviour of the Parser considering the size of the training data, we train the SyntaxNet with the 5000 train ESL dataset instead of 576k train dataset. We observe that the average number of fragments increases to 5.37 with the average size of 4.86; but the similarity of the Parser's fragments to the Reference's with the set-2-set F-score drops to 0.69. This observation also confirms that the Parser's performance depends on the size of the training data; when training the SyntaxNet with the smaller training data, we saw that it fragments more. Having small training set might not be enough to make a parser to be a good fragmentor; on the other hand, having a large training set might also not be optimal since the parser will perform more like a normal parser than a fragmentor. Therefore, it is important to find an optimal parameter in this spectrum. Since the focus of this thesis is on introducing the parse tree fragmentation and proposing practical approaches, we leave finding the optimal training size of parsers with respect to their performance for the future work.

6.4.4 Relationships between Fragments Statistics

To further evaluate the fragmentation methods, we analyze the relationships between the simple statistics of the produced fragments with the Reference fragments. The results in Table 10 reports the average number and size of each fragmentation method; however, the average might not best reflect the differences between the fragments, as it gives an aggregate but not the trend or the differences. To get a better insight on the relationships between the fragments, we further report the Pearson's r correlation and the root mean square error (RMSE) between the number and size of produced fragments and the Reference fragments. Table 11 summarizes the results. We observe that the Classification method has the highest correlation with Reference in terms of number of fragments and their sizes, but its RMSE numbers are far from the Reference fragments. This results suggest that even though the Classification does not break the trees into right number of fragments, its trend in breaking the trees is similar to Reference; when the Reference breaks more, the Classification also breaks more, and vice versa. On the other hand, the seq2seq method has the lowest RMSE numbers which shows its preciseness in fragmenting. In the MT dataset, the Classification method trained on ESL is making more accurate fragments and the results are along the line of the results in Table 9(b). These intrinsic evaluations suggest that different fragmentation methods might be useful for different NLP tasks that deal with ungrammatical sentences; the choice of fragmentation method might depend on a downstream application whether it benefits more from the number of fragments or the accuracy of the fragmentation.

6.5 CHAPTER SUMMARY

We have performed a set of empirical evaluations to investigate the impact of parse tree fragmentation. We compared the automatic fragmentation methods that we proposed in Chapter 5 with the gold standard fragments. We find that automatic fragmentation methods have different responses to ungrammatical sentences of various types. Our results suggest that given the domain of ungrammatical data and the size and type of the available resources, one can select an appropriate automatic fragmentation method.

(a) ESL dataset

Method	# of Fragments		size of Fragments	
	Pearson r	RMSE ($\downarrow$)	Pearson r	RMSE ($\downarrow$)
Classification	**0.453**	5.086	**0.299**	0.543
Parser	0.092	3.946	0.076	0.545
seq2seq	0.407	**3.068**	0.281	**0.444**

(b) MT dataset

Method	# of Fragments		size of Fragments	
	Pearson r	RMSE ($\downarrow$)	Pearson r	RMSE ($\downarrow$)
Classification	**0.646**	7.433	**0.377**	0.335
Parser	0.527	11.135	0.223	0.364
seq2seq (trained on ESL)	0.012	10.212	-0.011	0.654
Classification (trained on ESL)	0.589	**6.169**	0.326	**0.327**

Table 11: Relationship of fragmentation methods with Reference fragments over the number and size of fragments.

7.0 EVALUATION OF PARSE TREE FRAGMENTATION IN NLP APPLICATIONS

7.1 INTRODUCTION

The previous chapter on intrinsic evaluation only tells us how closely an automatic tree fragmentation method might approach the gold fragments. Since even the gold fragments are automatically created, we evaluate the potential utility of tree fragments in external NLP applications. We believe that the resulting fragments may still provide some useful information for downstream NLP applications that use parsing and deal with ungrammatical sentences in some way. Such applications are information extraction (IE), machine translation (MT), and automatic evaluation of text (e.g., generated by MT or summarization systems or human second language learners). One might also note that different applications may try to use tree fragments differently; and since the extrinsic evaluation is indirect, the results might depend on a selected application and its settings (i.e. different results might be obtained with different applications). This indicates that an extrinsic evaluation analysis on one application may not generalize to other application, as shown previously on extrinsic evaluation of parsers (Miyao et al., 2008; Elming et al., 2013; Oepen et al., 2017).

In this thesis, we verify the utility of tree fragments for two distinct NLP applications that use parsing in different levels, one on the sentence-level and the other on the word-level; therefore, we would be able to investigate different aspects of the parse tree fragmentation:

i. Sentence-level fluency judgment, in which a system automatically predicts how "natural" a sentence might sound to a native-speaker human. An automatic fluency judge can be used to decide whether an MT output needs to be post-processed by a professional translator; it can also be used to help grading student writings. We choose fluency judgment application since it is the direct application of parsing that deals with ungrammatical sentences.

ii. Semantic role labeling (SRL), in which a system identifies semantic roles of groups of words with respect to a particular verb in a sentence. A semantic role labeler can be used to understand sentences better; it can also be used to build knowledge bases for question answering systems. We choose semantic role labeling application since it is one of the basic tasks in semantic analysis of sentences, and studying semantic analysis of ungrammatical sentences could shed some light on this problem.

We hypothesize that if the fragmentation were helpful, the downstream applications should perform better with it than without it. For both applications, we consider two domains with ungrammatical sentences: writings of English-as-a-Second Language (ESL) learners and the MT outputs.

7.2 EXTRINSIC EVALUATION: FLUENCY JUDGMENT

There have been several previous work on sentence-level fluency judgment. Researchers have found that language model metrics alone are not sufficient, and various syntax-based features have been proposed to be incorporated into the fluency metric (Mutton et al., 2007; Post, 2011; Post and Bergsma, 2013). However, in order for these features to work well, they ought to be extracted from appropriate parse trees. Given that statistical parsers have difficulties with ungrammatical sentences, mis-interpreted parse trees may degrade the predictive power of the features. *We hypothesize that through parse tree fragmentation, major syntactic problems can be identified; thus, tree fragments should be useful for judging sentence fluency.*

7.2.1 Fluency Judgment Tasks

There are many different ways to set up a fluency judgment task; typically the desired granularity of the judgment differs depending on the application. We evaluate both binarized and ordinal level of grammaticality of sentences, because some applications might benefit more from binary classification of grammatical/ungrammatical sentences than a fine-grained judgment. For example, a systems that decides whether an ESL sentence needs to be corrected benefits from the binary fluency judgment, and a systems that helps grading ESL writings benefits more from a fine-grained judgment. Hence, we report two fluency judgment conditions: a binary classification and a regression formulation.

7.2.1.1 Binary Task

For the binary classification task, we train a classifier to distinguish between sentences that have virtually no error and those that have many errors. Thus, an ESL sentence is labeled 0 if it has no errors, and it is labeled 1 if it has three or more errors; an MT output is labeled 0 if its HTER score is less than 0.1, and it is labeled 1 if its HTER score is greater than 0.4. Although the setup is a little artificial, this study tells us how well each method performs on the extreme cases.

7.2.1.2 Regression Task

In contrast, the regression task is more challenging because the systems have to make finer distinctions of fluency. For the ESL dataset, the system has to predict the number of errors in each sentence (0, 1, 2, or 3+); for the MT dataset, the HTER score (a real number between 0 and 1).

7.2.2 Feature Sets

7.2.2.1 Our feature set

We extract four simple features from the output of each fragmentation method for each sentence:

i. Number of fragments
ii. Average size of fragments
iii. Minimum size of fragments
iv. Maximum size of fragments

7.2.2.2 Contrastive feature sets

We compare the proposed fragmentation approach against several contrastive baselines. In addition to typical language model features, we especially focused on previous work that rely on parse information:

- Sentence length (l).

- **LM (Language Modeling)**. An N-gram precision for $1 \leq N \leq 5$ is computed as a fraction of N-grams appearing in the reference text (we used the Agence France Press English Service (AFE) section of the English Gigaword Corpus (Graff et al., 2003).
- **C&J (Charniak&Johnson)** . This set of features is based on the complete set of parse tree reranking features of (Charniak and Johnson, 2005)[1] from Stanford parser's output version 3.2.0 (Klein and Manning, 2003). These features have been used previously for predicting grammaticality and are shown to perform well (Post and Bergsma, 2013). The feature set contains more than 60,000 features.
- **TSG (Post)**. This set of features is based on the tree substitution grammar (TSG) derivation counts from constituency tree (Post, 2011)[2]. This approach extracts more than 6000 features from the parse trees.

7.2.3 Experimental Setup

For all binary classification or regression tasks, we use the test datasets of ESL and MT which are containing 7000 and 6000 sentences respectively (discussed in Section 6.3.1). We run a 10-fold cross validation with the standard Gradient Boosting Classifier or Regressor (Friedman, 2001) in the scikit-learn toolkit (Pedregosa et al., 2011).[3] We tune Gradient Boosting parameters with a 3-fold cross validation on the training data: `learning_rate` over the range $\{0.0001 \ldots 100\}$ by multiples of 10 and `max_depth` over the range $\{1 \ldots 5\}$.

Since the test datasets are imbalanced, it is important to choose proper evaluation measures. For the binary classification, we report the standard accuracy metric that shows the percentage of correct predictions, and the AUC metric to take imbalanced test set into account. AUC estimates how probable it is that a classifier might give a higher rank to a randomly fluent sentence to a randomly disfluent one. The AUC of a random system is 0.5, while the its accuracy might be as high as the portion of skewed class. For example, in the ESL dataset, the accuracy of a system that tells all the sentences are fluent is 76% while its AUC in 0.5. The reported metrics for the regression task are root mean square error (RMSE) and Pearson's r correlation coefficient between

[1]https://github.com/mjpost/extract-spfeatures

[2]https://github.com/mjpost/post2011judging

[3]We have also tried SVMs with LibLinear toolkit (Fan et al., 2008), but gradient boosting learners obtained the best results.

the predicted and expected values.[4] RMSE penalizes the errors more than the mean absolute error (because of the square of distance); it is also shown to be a robust metric for ordinal evaluation of imbalanced data (Baccianella et al., 2009). A lower RMSE value indicates a better prediction system.

7.2.4 Results

Table 12 summarizes a comparison of different fluency judgment feature sets. Accuracy and AUC measures are reported for binary classification, root mean square error (RMSE) and Pearson's r are reported for regression.

The first block reports the baselines. For the ESL domain, the length of a sentence is a good indicator of the fluency of a sentence; longer sentences tend to have more errors than shorter sentences, but sentence length is not as strongly correlated with HTER score in the MT domain.

The second block of feature sets in the table shows that the four features extracted from parse tree fragments are correlated with the fluency quality of sentences. While it is expected that features based on PGold and Reference fragments should correlate strongly with fluency, Classification and seq2seq features also correlate with fluency better than C&J and TSG features in both domains. Moreover, they have different model sizes: the Classification and seq2seq feature sets consist of only 4 simple extracted features from the tree fragments, while C&J has more than 60k features and TSG has more than 6k features.

The Classification method significantly outperforms other methods (using a two-sided paired t-test with $> 95\%$ confidence from the 10 folds) on both domains. Also it performs comparable with the seq2seq in the binary task of ESL dataset. Although the seq2seq method makes more accurate pruning decisions (as we observed in the intrinsic evaluations of the previous chapter), it is not performing better than the Classification method in the fluency judgment application. This is because of the setup of the task, which uses only four simple features from fragments; especially since Classification produces more fragments, its number-of-fragments feature becomes a good indicator in the regression error prediction. Table 13 shows the Pearson r correlation of the extracted features with the fluency of the sentences in the regression task. We observe

[4] We have also evaluated the regression task with Kendall's τ and Spearman's ρ. Since the general trend of the results was similar to Pearson's r, we only report Pearson's r.

(a) ESL dataset

	Binary		**Regression**	
Feature Set	Acc.(%)	AUC	RMSE (↓)	r
Chance	76.1	0.5	1.249	
length (l)	77.3	0.75	0.994	0.304
LM	76.7	0.73	0.963	0.279
LM+l	80.6	0.84	0.933	0.417
C&J (Charniak&Johnson)	76.3	0.74	1.179	0.318
TSG (Post)	77.3	0.74	1.153	0.285
PGold	*100*	*1*	*0.537*	*0.889*
Reference	*100*	*1*	*0.557*	*0.879*
Classification	**80.7**	**0.82**	**0.905**	**0.411**
Parser	77.6	0.73	1.035	0.3
seq2seq	**81.3**	0.75	0.947	0.377

(b) MT dataset

	Binary		**Regression**	
Feature Set	Acc.(%)	AUC	RMSE (↓)	r
Chance	72.2	0.5	1.308	
length (l)	72	0.5	0.171	0.018
LM	74.4	0.71	0.163	0.307
LM+l	74.2	0.71	0.163	0.306
C&J (Charniak&Johnson)	68.3	0.6	0.186	0.136
TSG (Post)	69.8	0.59	0.179	0.105
Reference	*98.8*	*1*	*0.085*	*0.865*
Classification	**73.3**	**0.68**	**0.166**	**0.228**
Parser	71.8	0.56	0.171	0.077
seq2seq (trained on ESL)	71.9	0.52	0.171	0.06
Classification (trained on ESL)	72.4	0.66	**0.167**	0.207

Table 12: Fluency judgment results over two datasets containing ungrammatical sentences using binary classification and regression. Accuracy and AUC measures are reported for binary classification, and RMSE and Pearson's r are reported for regression. PGold and Reference as the upper bounds are given in italics, and the best result among automatic fragmentation methods is given in bold.

that the number of fragments from the Classification method trained on ESL data has the highest correlation with the number of errors in the sentences. Note that, however, seq2seq fragmentation method is completely automatic without any feature engineering to cut arcs and more importantly it learns to parse the sentences as well as pruning the arcs; while the Classification method uses hand engineered features for a binary classifier to decide which arcs of a given dependency tree to cut.

As a simpler fragmentation method, Parser is not as competitive for fluency judgment, especially with Classification and seq2seq methods. But it is still comparable to the other baseline methods. This suggests that Parser has learned some useful signals from the Reference training examples.

7.3 EXTRINSIC EVALUATION: SEMANTIC ROLE LABELING

To further verify parse tree fragmentation utility, we apply it in another downstream NLP application which benefits from syntactic parsing: semantic role labeling (SRL). The goal of SRL task is to identify the relations between group of words with respect to a particular verb in the sentence. These relations can then be used to understand the sentence better and help other NLP tasks such as question answering. Traditionally syntactic parsing plays an important role in SRL systems. Extracted features from parse trees are one of the main sets of features to detect semantic dependencies between parts of a sentence (Punyakanok et al., 2008). In this section, we aim to address performance of SRL systems on the ungrammatical sentences. Furthermore, we investigate whether extracted fragments from ungrammatical sentences might help to detect *incorrect semantic dependencies* in these sentences. As an example, Figure 26 shows an ungrammatical sentence and its automatically produced semantic dependencies. Because of the mistakes in the sentence, the SRL system assigns two incorrect semantic dependencies: “remember→I” and “known→for”. *We hypothesize that through parse tree fragmentation, major syntactic problems can be identified; thus, tree fragments should be useful to detect incorrect dependencies of semantic role labeling.*

Detecting *incorrect semantic dependencies* is crucial for systems that high accuracy is desirable. An example of these systems is modern search engines. To satisfy users’ information

(a) ESL dataset

Method	# of fragments	Avg. size	Min size	Max size
Reference	*0.842*	*-0.822*	*-0.765*	*-0.766*
Classification	**0.409**	**-0.317**	-0.178	**-0.241**
Parser	0.099	-0.093	-0.084	-0.063
seq2seq	0.285	-0.241	**-0.215**	-0.177

(b) MT dataset

Method	# of fragments	Avg. size	Min size	Max size
Reference	*0.662*	*-0.608*	*-0.476*	*-0.77*
Classification	0.155	-0.122	-0.047	-0.171
Parser	0.081	-0.056	-0.042	-0.082
seq2seq (trained on ESL)	0.076	-0.077	**-0.073**	-0.058
Classification (trained on ESL)	**0.191**	**-0.148**	-0.06	**-0.179**

Table 13: Correlation between the extracted features from each fragmentation method with the fluency of the sentence in the regression task. Reference as the upper bound is given in italics, and the best result in each column is given in bold.

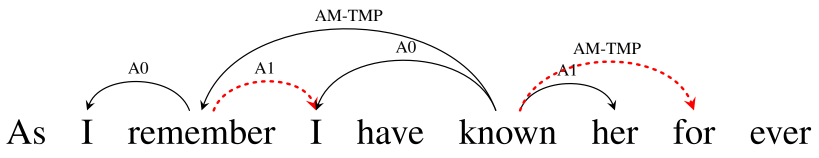


Figure 26: Automatically produced semantic dependency graph of an ungrammatical sentence. The red dotted relations show incorrect semantic dependencies.

needs, they go beyond retrieving relevant documents and display a concise answer to the user's query. For example, the query "barack obama wife", which asks for factual information, would return Michelle Obama as the answer. Thus search engines not only require a deep understanding of the user's query, but also need an accurate knowledge base to retrieve the correct answer. A Knowledge base is a graph of entities and their relations to provide answers to questions. They are typically automatically built by processing unstructured natural language text. One way to build a knowledge base is by adding semantic dependencies of a SRL system. Therefore, in order to have accurate knowledge base, it is important to add only correct semantic dependencies. It would not be acceptable if the search engine returns an incorrect answer, e.g. displays someone else's name as the Obama's wife. While it is still satisfactory if the search engine does not display any answer, since it would still retrieve some relevant documents based on the query words. Thus, adding noise to knowledge bases has negative consequences that should be avoided.

7.3.1 Semantic Role Labeling of Ungrammatical Sentences

Semantic role labeling evaluation for ungrammatical texts presents some domain-specific challenges. Similar to parsing, the typical approach to evaluate SRL systems is to compare extracted semantic dependencies against manually annotated gold standards. The available gold standard corpora with semantic roles are often created over grammatical text. A commonly used corpora are CoNLL-2005 and CoNLL-2009 shared task datasets which consist of the information on predicate-argument structures extracted from the PropBank corpus (Palmer et al., 2005) for the sections of the *Wall Street Journal* part of the Penn TreeBank (Marcus et al., 1993). In order to evaluate performance of the SRL system on ungrammatical sentences, we need to have a dataset with annotated semantic roles for these noisy sentences. Although there exists 300 machine translation outputs manually annotated with their semantic roles (Birch et al., 2013), its size makes it unsuitable for our extrinsic evaluation.

A "gold-standard free" alternative is to compare semantic roles for each noisy sentence with the semantic roles of the corresponding correct sentence. This approach is similar to our proposed parser robustness metric in Chapter 3. Here instead of comparing parse trees, we compare semantic graphs. We, therefore, can build a large amount of near gold annotated semantic dependencies for ungrammatical sentences as long as we have a parallel corpus of problematic sentences with their corrected versions. These parallel data can be either ESL writings with their corrected versions, or machine translation outputs with their human post-editions. A limitation of this approach is that the comparison works best when the differences between the problematic sentence and the correct sentence are small. This is not the case for some ungrammatical sentences (especially from MT systems).

One difference between parsers and SRL systems is that parsers perform slightly better than SRL systems. Although there has been a huge progress on SRL systems, the overall F_1 score of state-of-the-art SRL systems is around 87%; while accuracy of state-of-the-art parsers is more than 93%. One reason of lower performance of SRL systems is that they typically use parsers's outputs as features. Therefore, errors in parser's output may propagate through semantic role detection. Despite of the lower performance of SRL systems, there have been several previous works that used state-of-the-art SRL systems to build gold annotations. On a series of work by Akbik et al. (Akbik et al., 2015; Akbik and Li, 2016), they used an English off-the-shelf SRL system to project SRL annotations of an English sentence to its translations in other languages. In this way, they were able to automatically construct annotated corpora with semantic dependencies for several languages. Our proposed evaluation methodology is similar to their approach of projecting semantic roles; instead we project the semantic dependencies of grammatical sentence to its corresponding ungrammatical sentence.

7.3.2 Creating Pseudo Gold Semantic Dependencies for Ungrammatical Sentences

For the purpose of evaluating semantic dependencies of an ungrammatical sentence, we take the automatically produced semantic relations of a grammatical sentence as "gold standard" and compare the SRL output for the corresponding ungrammatical sentence against it. Our proposed gold standard procedure is based on three assumptions:[5]

i. For every ungrammatical sentence, there is a grammatical sentence that has the same meaning as the ungrammatical sentence.

ii. A state-of-the-art SRL system produces semantic dependencies of a grammatical sentence that reflect, to some extent, that sentence's intended meaning.

iii. The semantic dependencies of an ungrammatical sentence should be as close as possible to semantic dependencies for its corresponding grammatical sentence.

In keeping with these assumptions, we create gold semantic dependencies for an ungrammatical sentence by projecting the semantic dependencies of its grammatical sentence to the ungrammatical sentence. Following are the steps that we take:

- **Step 1:** Running a state-of-the-art SRL system over the grammatical sentences. We use Mate SRL toolkit (Björkelund et al., 2009) (see Section 7.3.5 for details).
- **Step 2:** Finding word alignments between ungrammatical and grammatical sentences. The word alignment in semantic role labeling is slightly different from parsing as in our proposed robustness evaluation metric in Section 3.3. Finding word to word alignments in the ESL dataset is pretty straightforward, because we have the error corrections. But alignment of MT data is more challenging. In the parsing evaluation, we used word edit distance to find MT word alignments. It was a reasonable choice for parsing evaluation, because we wanted to investigate the impact of each single error. While in the SRL, we do not want to penalize the SRL system when the errors come from semantic differences, for example replacement of two synonyms "commence" and "start". Thus, for the SRL word alignment, we use a state-of-the-art monolingual word alignment system (Sultan

[5] Similar assumptions have been introduced by Foster (2007) for parse trees.

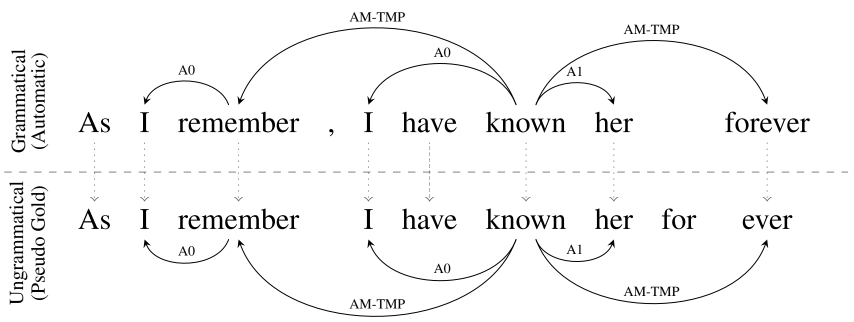

Grammatical (Automatic)
Ungrammatical (Pseudo Gold)
As I remember , I have known her forever
As I remember I have known her for ever
AM-TMP
A0
A0
A1
AM-TMP
A0
A0
A1
AM-TMP
AM-TMP

Figure 27: Projecting semantic dependencies of the Grammatical sentence (top) to the Ungrammatical sentence (bottom) to create "gold standard" semantic dependencies of the ungrammatical sentence.

et al., 2014) which aligns related words in the two sentences by exploiting the semantic and contextual similarities of the words.

- **Step 3:** Projecting directly SRL annotations of grammatical sentence to the ungrammatical sentence using the alignments. If a word in the ungrammatical sentence is aligned to a word in the grammatical sentence, we directly project the semantic role of the word in the grammatical sentence to the word in the ungrammatical sentence.

Figure 27 shows an example of projecting semantic dependencies from the grammatical sentence to the ungrammatical sentence. Even if the process of projecting "gold standard" semantic dependencies is not perfectly correct, it presents the norm from which semantic dependencies of ungrammatical sentences diverge: if two sentences have the same meaning, their semantic dependencies for these sentences should be similar. Therefore, we assume that SRL annotations of the ungrammatical sentence are the same as their corresponding grammatical sentence.

7.3.3 Applying Fragmentation to Automatic SRL Annotations

Our goal is to investigate the impact of parse tree fragmentation on detecting incorrect semantic dependencies of ungrammatical sentences. For this purpose, we propose two approaches to utilize parse tree fragmentation of ungrammatical sentences. In one, we introduce a heuristic rule-based method to detect incorrect semantic dependencies. In the second one, we introduce a classification model that finds incorrect dependencies based on fragmentation features.

7.3.3.1 Approach 1: Rule-based

One way to detect incorrect semantic dependencies of an ungrammatical sentence is to assume that any semantic dependency that crosses the sentence's parse tree fragments is not correct and should be removed. Although this is a restrictive assumption, it simplifies our argument for the use of fragmentation and, at the same time, it still helps us to evaluate the usefulness of fragmentation by counting the number of detected incorrect semantic dependencies.

Given an ungrammatical sentence, the steps that we take to apply parse tree fragments to the semantic dependencies are as below:

- **Step 1:** Finding parse tree fragments of the ungrammatical sentence using one of the fragmentation methods introduced in the Chapters 4 and 5.
- **Step 2:** Running a state-of-the-art SRL system over the ungrammatical sentence.
- **Step 3:** Removing semantic dependencies that cross between fragments, i.e. when the predicate and the argument of a semantic dependency are in different parse tree fragments.
- **Step 4:** Comparing the resulting semantic dependencies with the gold standard (projected) dependencies of the ungrammatical sentence (which will be discussed in Section 7.3.4).

Figure 28 shows an example of the fragmented semantic dependencies of an ungrammatical sentence. The ungrammatical sentence has four parse tree fragments. Using the rule-based approach, all the three cross-fragment semantic dependencies are removed. Two of the dependencies are correctly removed, but the relation "known→remember" is a correct semantic dependency that should not be removed. To address the issue of cutting correct relations, in the next section we propose a smarter approach to learn when to cut semantic dependencies using fragmentation features.

7.3.3.2 Approach 2: Machine-Learning-based (ML)

Yet another way to detect incorrect semantic dependencies is to train a classifier to discriminate

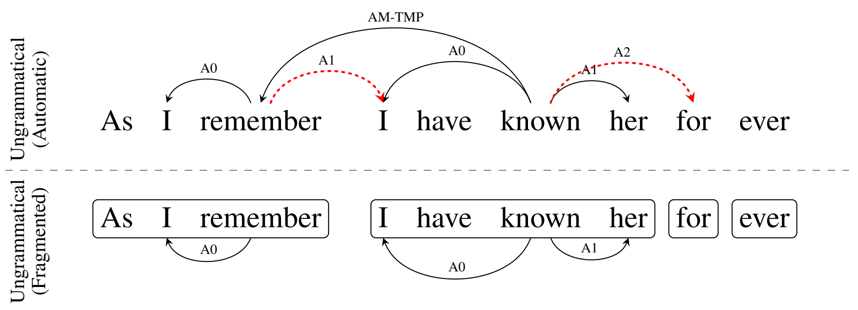

Ungrammatical (Automatic)
AM-TMP
A0
A1
A0
A2
A1
As I remember I have known her for ever
Ungrammatical (Fragmented)
As I remember
A0
I have known her
A0
A1
for
ever

Figure 28: Applying fragmentation to automatic semantic dependencies of an ungrammatical sentence using the rule-based approach.

between the right and wrong contexts for some semantic dependencies. We formulate this as a binary classification problem: for each semantic dependency generated by an automatic SRL system indicates whether the dependency is correct or incorrect. Using projected SRL annotations as examples, we train a Gradient Boosting Classifier that learns to detect incorrect semantic dependencies. The trained classifier can then make prediction on the unseen semantic graphs of ungrammatical sentences.

Because the number of correct semantic dependencies is greater than the incorrect ones, we make a balanced training set by randomly sampling equal numbers of the correct and incorrect dependencies[6]. We extract the following features for each semantic dependency in an automatically generated semantic graph of an ungrammatical sentence:

- A binary feature that denotes whether the semantic role crosses between parse tree fragments. For example the semantic dependency of "known→for" in the Figure 28 crosses two fragments, while the semantic dependency of "known→her" does not cross parse tree fragments. This feature value is extracted for each parse tree fragmentation method separately.
- Type of the semantic dependency (e.g. A0, A1, A2 or AM-LOC). This feature is also dependent to each parse tree fragmentation method.

[6]We have followed a similar approach in the binary classification discussed in Section 5.2.1

The next sets of features are independent from the fragmentation method and are the adapted versions of the features of the parse trees described in Section 5.2.1:

- Depth and height of the predicate and argument of semantic dependency when the SRL graph is traversed in depth-first order. Similar example for parse trees is given in Figure 19.
- Part-of-speech tags of the predicate, argument, and the parent of the predicate word. For example in the Figure 19, for the arc of "known→for" the POS tags of "known", "for", and "remember" are extracted.
- Word bigrams and trigrams corresponding to the arc (a similar example for parse trees is shown in Figure 20). Denoting w_h $(h = 1, 2, ..)$ as the predicate word and w_m as the argument word, the bigram feature are calculated for the pairs of $w_h w_m$ ($w_m w_h$ if $m < h$), $w_{m-1} w_m$, and $w_m w_{m+1}$. The trigram features are calculated for the triples of $w_{m-1} w_m w_{m+1}$, $w_{m-2} w_{m-1} w_m$, and $w_m w_{m+1} w_{m+2}$. We use both raw counts and pointwise mutual information of the N-grams. To compute the N-gram counts, we use Agence France Press English Service (AFE) section of English Gigaword (Graff et al., 2003).

7.3.4 Evaluating Automatic SRL Annotations of Ungrammatical Sentences

Given a set of "gold standard" semantic dependencies for an ungrammatical sentence, we can evaluate performance of an automatic SRL system or fragmented semantic graph of an ungrammatical sentence. We focus on evaluating argument identification and labeling because these are the steps which have been previously believed to require syntactic information (Punyakanok et al., 2008). For a given semantic dependency, the head of an argument span is connected to the predicate and labeled with a semantic role (e.g. A0 or A1). For example as depicted in the Figure 29, the verb "known" is the predicate and "her" is one of its arguments, representing A1 (described as patient or theme) relation.

In order to compare the SRL annotations[7] of ungrammatical sentences with the gold standard SRL annotations (i.e. projected annotations, introduced in Section 7.3.2), we use the standard

[7]The SRL annotations could be either the output of the automatic SRL system or the fragmented SRL graph by the methods introduced in Section 7.3.3.

CoNLL-2009 evaluation scrip[8]. The script computes the confusion matrix between the automatic and gold semantic dependencies. In our evaluation, the four values of confusion matrix are defined as below:

- True Positive (TP): Correctly identified semantic dependencies by both automatic system and the gold standard.
- False Positive (FP): Incorrectly identified semantic dependencies by automatic system, while there are not semantic dependencies in the gold standard. This type of error is also called false alarm or *Type I error*.
- True Negative (TN): Correctly identified no semantic dependencies by both automatic system and the gold standard.
- False Negative (FN): Incorrectly identified no semantic dependencies by the automatic system, while there are semantic dependencies in the gold standard. This type of error is also called miss or *Type II error*.

In this research, we do not have any control on adding new semantic dependencies; we can only remove the incorrect semantic dependencies since applying fragmentation methods over the automatic SRL annotations cuts some of the semantic dependencies. Therefore, monitoring the number of False Positives is a crucial measure to evaluate the helpfulness of fragmentation methods to detect incorrect semantic dependencies. A method of measuring Type I error is *False Discovery Rate* (FDR) (Murphy, 2012) which defined as:

$$\text{False Discovery Rate (FDR)} = \frac{\text{False Positive}}{\text{False Positive + True Positive}}$$

In another words, FDR is the ratio of incorrect semantic dependencies out of all the identified semantic dependencies by an automatic SRL system. In our experiments, to compare the performance of SRL systems, we report their False Discovery Rates. The smaller the FDR number, the better the system performs in detecting incorrect semantic dependencies. Figure 29 shows an example of evaluating automatic semantic dependencies of an ungrammatical sentence against its projected (gold standard) semantic dependencies. There are two incorrect semantic dependencies in the automatic SRL system which result in False Discovery Rate of $2/6 \approx 33\%$.

[8]http://ufal.mff.cuni.cz/conll2009-st/eval09.pl

Since applying fragmentation methods may remove mistakenly some correct semantic dependencies as well as the incorrect ones, we also report the overall number of the missing semantic dependencies by measuring False Negatives. A method of measuring False Negatives is False Negative Rate (FNR) (Murphy, 2012) which defined as:

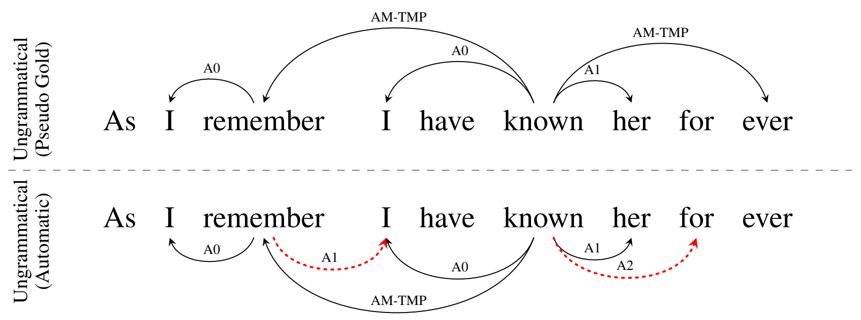
$$\text{False Negative Rate (FNR)} = \frac{\text{False Negative}}{\text{False Negative + True Positive}}$$

FNR measures the ratio of missing semantic dependencies by an automatic SRL system out of all the semantic dependencies of the gold standard. Therefore, the smaller FNR number, the better the system performs in preserving correct semantic dependencies. In the example of Figure 29, there is one missing semantic dependency in the automatic semantic dependencies when comparing with the pseudo gold dependencies ("known→ever"), which results in having one False Negative. Thus, the False Negative Rate is calculated as $1/(1+4) = 0.2$.

In this research, we are less concerned with the False Negatives because we do not have any control over adding new semantic dependencies – applying fragmentation methods will only cut semantic dependencies. While the fragmentation methods may cut some correct semantic dependencies, thus introducing false negative cases, that is less problematic than leaving in incorrect semantic dependencies. Detecting incorrect semantic dependencies is crucial for applications that need high accuracy e.g. by building accurate knowledge bases. Therefore, we mainly monitor the number of false positives using the FDR metric to evaluate the helpfulness of fragmentation methods when detecting incorrect semantic dependencies.

7.3.5 Experimental Setup

We use the test datasets of ESL and MT (that are discussed in Section 6.3.1) and parse the sentences using the SyntaxNet parser. We then run the semantic role labeler of the Mate toolkit (Björkelund et al., 2009). Mate toolkit has achieved state-of-the-art semantic F-score in the semantic role labeling task of the CoNLL-2009 shared task (Hajič et al., 2009), and has been used as an off-the-shelf SRL system since (Akbik et al., 2015; Akbik and Li, 2016). The Mate SRL system is implemented as a sequence of local logistic regression classifiers for the four steps of predicate identification, predicate classification, argument identification and argument classification. It uses a standard



Ungrammatical (Pseudo Gold)
AM-TMP
A0
AM-TMP
A0
A1
As I remember I have known her for ever
Ungrammatical (Automatic)
As I remember I have known her for ever
A0
A1
A0
A1
A2
AM-TMP

Figure 29: Evaluating the **automatic** semantic dependencies (bottom) with the gold standard/projected semantic dependencies (top) of the Ungrammatical sentence. The dotted red relations show produced false positive relations by the automatic SRL. The False Discovery Rate (FDR) is $2/6 \approx 33\%$.

feature set of lexical and syntactic features. In addition, it reranks sets of local predictions by implementing a global reranker.

For the machine-learning-based method of applying fragmentation on SRL annotations (discussed in Section 7.3.3.2), we train the standard Gradient Boosting Classifier (Friedman, 2001) in the scikit-learn toolkit. We use the 10-fold cross validation over the test data.

7.3.6 Results

The experiments aim to address the usefulness of the parse tree fragmentation methods to detect incorrect semantic dependencies of ungrammatical sentences. Specifically, we are interested in answering the following questions:

- How do fragmentation methods perform on detecting incorrect semantic dependencies of erroneous sentences? (Section 7.3.6.1)
- To what extent detecting incorrect semantic dependencies negatively impacted by the increase in the number of errors in sentences? (Section 7.3.6.2)
- To what extent detecting incorrect semantic dependencies negatively impacted by the interactions between multiple errors? (Section 7.3.6.3)

- What types of errors are more problematic for detecting incorrect semantic dependencies? (Section 7.3.6.4)

7.3.6.1 Overall Performances

In this section, we address the first question by exploring the overall performance of fragmentation methods on detecting incorrect semantic dependencies in terms of False Discovery Rate (FDR) and False Negative Rate (FNR). We also evaluate the overall performance of the machine-learning-based method.

Overall False Discovery Rates

The overall False Discovery Rates (FDR) of the fragmentation methods in detecting incorrect semantic dependencies are shown in Table 14. The "0+" columns indicate the experiments over the original test datasets in which sentences are randomly selected from each domain and might contain no errors. Since more than 40% of the ESL sentences, and 35% of the MT sentences do not have any or very few changes (as shown in Figures 23(a) and 24(a)), to remove the impact of these sentences, we also report the overall SRL results on the sentences with at least one error i.e. "1+". The performance of detecting incorrect semantic dependencies are reported with respect to the metric of False Discovery Rate (FDR). Note that the smaller FDR indicates lower rate of type I error. The FDR$_{rule}$ and FDR$_{ML}$ columns show the performance of fragmentation methods when applied on the output of automatic SRL system using two approaches of rule-based and machine-learning-based respectively (discussed in Section 7.3.3).

The first row of the table is the baseline method a.k.a Basic. The Basic method compares the projected semantic dependencies (as the gold standard) with the automatically produced semantic dependencies on the ungrammatical sentences. In both ESL and MT datasets, the Basic method shows how well the automatic SRL system performs when processing domains that contain ungrammatical sentences. As expected, the FDR numbers are higher in the 1+ dataset, as it is because the sentences with no errors are ignored and so the total number of semantic dependencies are reduced which makes the ratio of incorrect dependencies to the total dependencies increases.

Table 14 shows that, for both datasets, applying fragmentation methods reduces the False Discovery Rates. This suggests that tree fragments are useful in decreasing the rate of incorrect se-

(a) ESL dataset

Method	0+ errors FDR$_{Rule}$	0+ errors FDR$_{ML}$	1+ errors FDR$_{Rule}$	1+ errors FDR$_{ML}$
Basic	12.81		22.68	
Reference	*3.82*	*3.65*	*9.51*	*9.19*
Classification	**7.07**	7.40	19.57	14.87
Parser	12.24	7.88	22.68	15.01
seq2seq	9.24	**7.32**	**17.26**	**14.11**

(b) MT dataset

Method	0+ HTER FDR$_{Rule}$	0+ HTER FDR$_{ML}$	0.1+ HTER FDR$_{Rule}$	0.1+ HTER FDR$_{ML}$
Basic	33.51		39.51	
Reference	*16.98*	*16.16*	*21.79*	*20.72*
Classification	**26.35**	26.96	**37.30**	32.42
Parser	29.29	26.72	38.40	32.54
seq2seq (trained on ESL)	32.86	**26.43**	38.61	**31.93**
Classification (trained on ESL)	28.78	26.84	38.61	31.91

Table 14: Overall performance of fragmentation methods in detecting incorrect semantic dependencies in terms of False Discovery Rates (FDR). The "0+" columns indicate the experiments over the sentences with zero or more errors, and the "1+" columns reports the results on the sentences with at least one error. Reference as the upper bound is given in italics, and the best result among automatic arc pruning methods is given in bold.

mantic dependencies of ungrammatical sentences. The Reference method is outperforming other tree fragmentation methods as it uses extra source of information to identify major syntactic problems. When applying fragmentation, the machine-learning-based approach is mostly performing better than the rule-based approach. Moreover, The machine-learning-based approach uses other features than fragmentation features to detect incorrect semantic dependencies, so this makes it pretty much robust among the automatic fragmentation methods, i.e. the FDR$_{ML}$ is similar for the Classification, Parser and seq2seq fragmentation methods. However, on the sentences with at least one error, the seq2seq method gets the best overall results. Since the machine-learning-based approach outperforms the rule-based approach, we use the machine-learning-based approach for the rest of the experiments.

Overall False Negative Rates

In this experiment, we evaluate the fragmentation methods by how well they preserve the correct semantic dependencies from removing. Although our main goal is to evaluate the performance of fragmentation methods in detecting incorrect semantic dependencies, we are also interested to see what percentage of semantic dependencies are missed by each method. We evaluate the percentage of missing semantic dependencies in terms of False Negative Rate (FNR). Table 15 shows the overall FNR of the fragmentation methods. As we expected, the fragmentation methods have higher FNRs than the Basic method, because they are designed to remove semantic dependencies so they may remove semantic dependencies mistakenly which results in having higher False Negatives as well as lowering True Positives. The Reference method is also performing better than other fragmentation methods since it uses extra source of information so it serves as the upper bound for the automatic methods. But even among the automatic fragmentation methods, the seq2seq method outperforms other methods in the ESL data which shows it is a practical fragmentation method that both learns to parse and fragment ungrammatical sentences. The FNR scores in the MT data are higher than the ESL data which shows MT sentences are more challenging than ESL.

Performance of Machine-Learning-based methods

Machine-Learning-based approach runs a binary classification modal over semantic dependencies, deciding whether a dependency is correct or incorrect. The ground-truth labels come from the projected semantic dependencies. We performed a 10-fold cross validation over the ESL and MT

(a) ESL dataset

Method	**0+ errors** FNR$_{ML}$	**1+ errors** FNR$_{ML}$
Basic	5.76	12.03
Reference	*23.12*	*32.63*
Classification	38.30	46.20
Parser	40.37	46.52
seq2seq	**34.48**	**42.87**

(b) MT dataset

Method	**0+ HTER** FNR$_{ML}$	**0.1+ HTER** FNR$_{ML}$
Basic	17.13	21.70
Reference	*42.03*	*47.16*
Classification	53.07	**55.37**
Parser	**52.48**	55.60
seq2seq (trained on ESL)	52.55	55.63
Classification (trained on ESL)	52.84	55.68

Table 15: Overall False Negative Rates (FNR) of fragmentation methods. Reference as the upper bound of fragmentation methods is given in italics, and the best result among automatic arc pruning methods is given in bold.

(a) ESL dataset

Method	**0+ error** AUC	**1+ error** AUC
Reference	**0.815**	**0.755**
Classification	0.68	0.65
Parser	0.67	0.648
seq2seq	0.698	0.666

(b) MT dataset

Method	**0+ error** AUC	**1+ error** AUC
Reference	**0.747**	**0.721**
Classification	0.617	0.607
Parser	0.619	0.602
seq2seq (trained on ESL)	0.619	0.608
Classification (trained on ESL)	0.616	0.608

Table 16: Performance of binary classification models of machine-Learning-based approach (Section 7.3.3.2) using fragmentation features to detect incorrect semantic dependencies.

test data. Note that while we make the train data to be balanced (using 9 folds), the test data (the 10th fold) is not; thus, a baseline of never detecting incorrect dependencies would result in a high classification accuracy (84% on ESL and 57% MT "0+ error" datasets). Similar to the other imbalanced test sets in this thesis, in order to take the skewed class distribution into account, we evaluate classifies with the AUC measure. The AUC estimates how probable it is that a classifier might give a higher rank to a randomly incorrect dependency compared to a randomly correct one. Table 16 presents the AUC of the classifiers with the features from different fragmentation methods. The AUC of the classifiers with the Reference features are higher than other classifiers. However, all the classifiers are performing better than the baseline (detecting no incorrect semantic dependencies) which is 0.5. The AUC scores suggest that the Machine-Learning-based classifiers with the fragmentation features are making reasonable decisions to detect incorrect semantic dependencies of ungrammatical sentences.

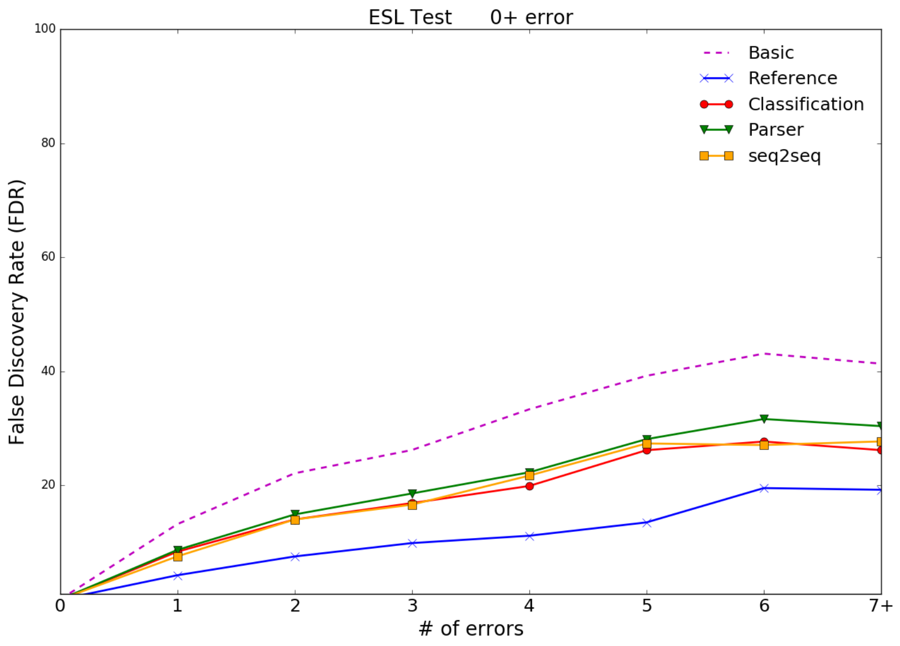
7.3.6.2 Impact of Number of Errors

We further analyze the results by separating the test sentences by the number of errors each contains. Our objectives are: (1) to observe the speed with which the rates of false positives increases as the sentences become more error-prone; (2) to determine the differences between fragmentation methods and the basic SRL system when handling noisier data.

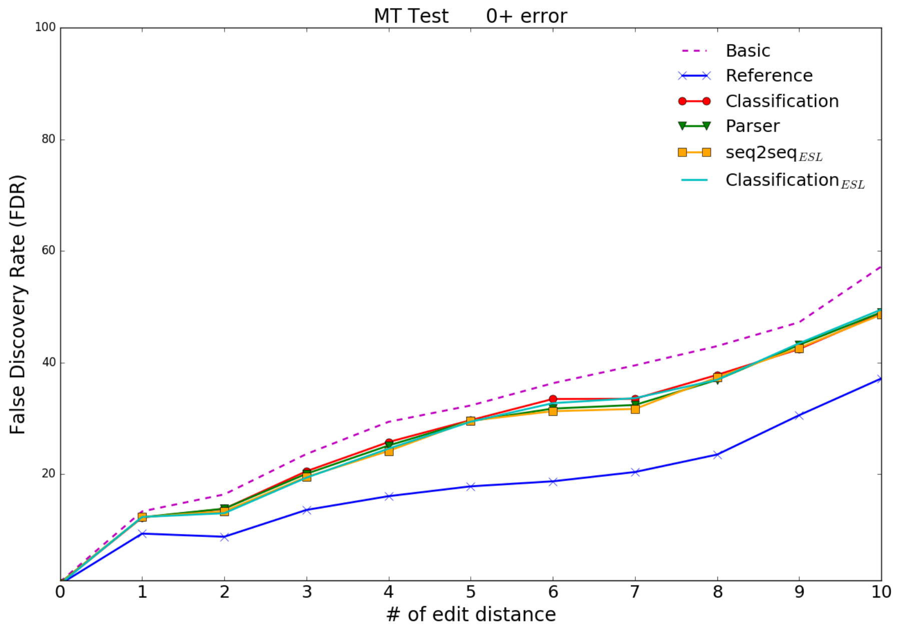
Figure 30 presents two graphs, plotting False Discovery Rates against the number of errors for two test datasets of ESL and MT. We observe that 1) the FDR score is increasing more rapidly for the Basic method than the Reference method; 2) using fragmentation features to detect incorrect semantic dependencies led to a similar behavior between the fragmentation methods. In both datasets, the FDR increases gradually with increasing number of errors; therefore, the fact of detecting incorrect semantic dependencies becomes more crucial for the noisier sentences.

7.3.6.3 Impact of Error Distances

In this experiment, we explore the impact of the interactivity of errors. Similar to the experiments in Section 3.5.3, we assume that errors have more interaction if they are closer to each other, and less interaction if they are scattered throughout the sentence. We define "near" to be when there is at most 1 word between errors and "far" to be when there are at least 6 words between



(a) ESL dataset



(b) MT dataset

Figure 30: Variation in False Discovery Rates as the number of errors in the test sentences increases.

errors. We expect that the SRL system and the fragmentation methods have more difficulty on detecting incorrect semantic dependencies when the errors have more interaction. We conduct this experiment using a subset of sentences that have exactly two errors; we compare False Discovery Rate of the methods when the two errors are near each other and when the errors are far apart.[9]

Table 17 presents the results using our representation of the shaded bars. Each dataset is treated as one group. The top row specifies the lowest FDR and the bottom row specifies the highest FDR. The shaded area of each bar indicates the relative FDR of each method with respect to the lowest and highest FDR scores of the group. Note that the lower FDR is desirable, so the emptier bar indicates the system that detects lower ratio of incorrect semantic dependencies. In all the datasets, the Reference method has the least ratio of incorrect semantic dependencies (indicating the empty bar) and the Basic method has the highest ratio of incorrect dependencies (indicating the fully shaded bar). As expected, the Basic method shows more difficulty with near errors than far errors (the ratio of its False Positives is higher with near errors). In the ESL dataset, the near errors are less challenging for the fragmentation methods; they only exhibit minor differences between near and far errors. Compared to ESL data, near errors in MT data are more challenging for the fragmentation methods; they all have more problems in detecting incorrect semantic dependencies when the errors are near.

The results of error interactivity in detecting incorrect semantic dependencies are consistent with the error interactivity in parser robustness. They both show that the near errors are more problematic for both parsers and SRL systems. With respect to the SRL results, the fragmentation methods are helpful to reduce the ratio of incorrect semantic dependencies. Specifically the Reference method outperforms other methods.

7.3.6.4 Impact of Error Types

In the following experiments, we examine the impact of different error types. To remove the impact due to interactivity between multiple errors, we study a subset of sentences that have only one error. Our objective is to see whether some error types are more challenging for SRL systems than others.

[9]We chose the sentences with exactly two errors in order to have more sentences in each group. While in the experiments of Section 3.5.3, we chose sentences with three errors since the test datasets were larger in that experiment.

(a) ESL dataset

Method	0+ errors Near	0+ errors Far	1+ errors Near	1+ errors Far
min	7.76 (Reference)		9.25 (Reference)	
Basic	██████████	█████████░	██████████	████████░░
Reference	░░░░░░░░░░	░░░░░░░░░░	▌░░░░░░░░░	░░░░░░░░░░
Classification	████░░░░░░	███░░░░░░░	████░░░░░░	███░░░░░░░
Parser	█████░░░░░	████░░░░░░	█████░░░░░	███░░░░░░░
seq2seq	███░░░░░░░	████░░░░░░	████░░░░░░	███░░░░░░░
max	21.44 (Basic)		23.58 (Basic)	

(b) MT dataset

Method	HTER 0+ errors Near	HTER 0+ errors Far	HTER 1+ errors Near	HTER 1+ errors Far
min	7.45 (Reference)		9.39 (Reference)	
Basic	██████████	███████░░░	██████████	█████░░░░░
Reference	░░░░░░░░░░	██░░░░░░░░	▌░░░░░░░░░	░░░░░░░░░░
Classification	█████████░	████░░░░░░	███████░░░	█░░░░░░░░░
Parser	████████░░	████░░░░░░	███████░░░	██░░░░░░░░
seq2seq (trained on ESL)	████████░░	████░░░░░░	███████░░░	█░░░░░░░░░
Classification (trained on ESL)	████████░░	████░░░░░░	████████░░	▌░░░░░░░░░
max	16.17 (Basic)		18.43 (Basic)	

Table 17: False Discovery Rates on test sentences with two near and two far errors. Each bar indicates the level of FDR scaled to the lowest score (empty bar) and highest score (filled bar) of a group.

Impact of error semantic role

An error can be either in a verb role, an argument role, or no semantic role. We extract semantic role of the error on the ungrammatical sentence by running an automatic SRL system on the corrected version of the sentences. We then obtain the role of the errors using alignments between ungrammatical sentence and its corrected counterpart. Table 18 presents the performance of the methods in detecting incorrect semantic roles over sentences that have one error. Sentences with argument errors are more challenging for all the methods even the Reference method; the ratio of false positives are higher when there is an argument error in the sentence. These results are opposite of the parser robustness results in which we observed that handling errors in argument words is somewhat easier for parsers. The reason may be because the errors in arguments might not impact the syntactic structure of the sentence, but these errors may change the semantic of the sentence and so make difficulties to detect incorrect semantic dependencies.

To further study the impact of argument errors and to see which semantic role is more challenging, we breakdown the sentences with one argument error with the semantic role label of the argument error. Table 19 shows the results for the top seven argument roles in our test data. A brief description of the semantic roles is given in Table 22. In the the ESL dataset, the A2 semantic role seems to be the most challenging role for all the methods. In the MT dataset, the AM-LOC is the most difficult semantic role to detect; even the Reference method has the highest ratio of false positives for this role. In general, the variation of the semantic roles does not seem to impact the performance of the methods in detecting incorrect semantic roles; each method performs equally well or poorly across most of the roles. But there are some exceptions, for instance in the ESL dataset, fragmentation methods perform differently for the AM-MNR semantic role; the Reference method has the best performance by removing all the false positives, while the seq2seq has the worst performance. One reason of this huge variation is that there are only 25 sentences with one error where the error occurs on a word taking on an AM-MNR semantic role. Thus, considering a larger test sample, the average results might be different.

Impact of grammatical error types

In this experiment, we explore the impact of the three grammar error types: replacement (a word need replacing), missing (a word missing), and unnecessary (a word is redundant). Our goal is to

(a) ESL dataset

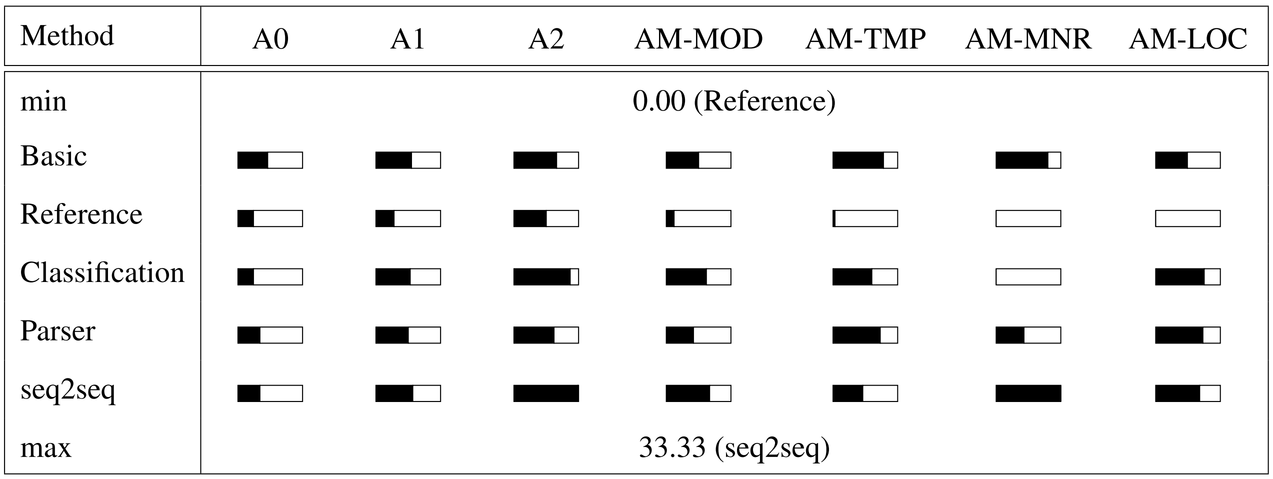
Method	Verb	Argument	No role
min		3.05 (Reference)	
Basic	■■■■■■□□□□	■■■■■■■■■□	■■■■■□□□□□
Reference	□□□□□□□□□□	■■■■□□□□□□	■□□□□□□□□□
Classification	■■■□□□□□□□	■■■■■■■■□□	■■■□□□□□□□
Parser	■■■■□□□□□□	■■■■■■■■■■	■■□□□□□□□□
seq2seq	■■■□□□□□□□	■■■■■■■■■□	■■□□□□□□□□
max		18.09 (Parser)	

(b) MT dataset

Method	Verb	Argument	No role
min		7.71 (Reference)	
Basic	■■■□□□□□□□	■■■■■■■■□□	■■■□□□□□□□
Reference	□□□□□□□□□□	■■■■■■■□□□	■□□□□□□□□□
Classification	■■□□□□□□□□	■■■■■■■■■■	■■■□□□□□□□
Parser	■■■□□□□□□□	■■■■■■■■■□	■■■□□□□□□□
seq2seq (trained on ESL)	■■■□□□□□□□	■■■■■■■□□□	■■■■□□□□□□
Classification (trained on ESL)	■■□□□□□□□□	■■■■■■■■■□	■■■□□□□□□□
max		20.1 (Classification)	

Table 18: False Discovery Rates on test sentences with one error where the error occurs on a word taking on a verb role, an argument role, or a word with no semantic role.

(a) ESL dataset



(b) MT dataset

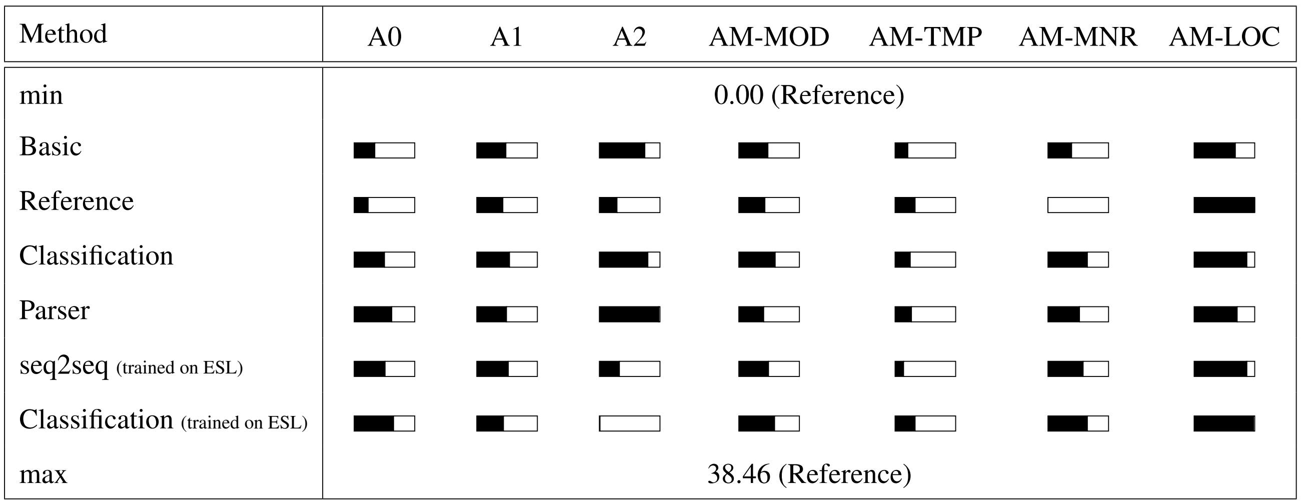


Table 19: False Discovery Rates on sentences with one error, where the error occurs on a word taking an argument role that has one of the seven frequent role labels.

see what types of errors are more problematic for detecting incorrect semantic dependencies. As shown in Table 20, in the ESL dataset, the missing word error is somewhat the less challenging error type, and the replacement word error is the most challenging one. While in the MT dataset, the missing word error is the most challenging error. In the MT dataset, except the Reference method, almost all the methods have difficulties with detecting incorrect semantic dependencies. This shows that the MT domain is more challenging than the ESL domain even when there is only one word change between the ungrammatical sentence and its corrected counterpart.

Impact of error word category

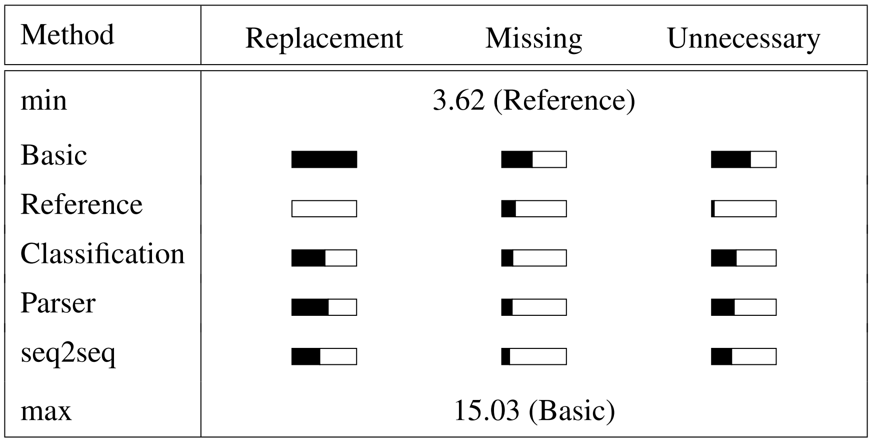
Another factor that might affect performance of the fragmentation methods in detecting incorrect semantic dependencies is the class of the errors. We separate the sentences into two groups: errors occurring on an open-class word (e.g. verbs and nouns) and errors occurring on closed-class word (e.g. prepositions and pronouns). As the Table 21 shows the open-class errors are generally more difficult. This might be because the impact of the open-class words is higher in the semantic of the sentence than the closed-class errors which are functional words. While in the parser robustness experiments (Section 3.5.4.2) the closed-class errors were more difficult for parsers, since they have higher impact on the structure of sentences.

7.3.6.5 Discussion

The results of the semantic role labeling experiments highlight the helpfulness of parse tree fragmentation in detecting incorrect semantic dependencies of ungrammatical sentences. We observe that the off-the-shelf semantic role labeler (Basic method) identifies high ratio of semantic dependencies that are not correct; using fragmentation features we are able to detect some of these incorrect semantic dependencies. Specifically, the Reference method significantly helps this task as the upper bound approach. Although there is a performance gap between the automatic fragmentation methods and the Reference method, the automatic methods are still useful in detecting incorrect semantic dependencies.

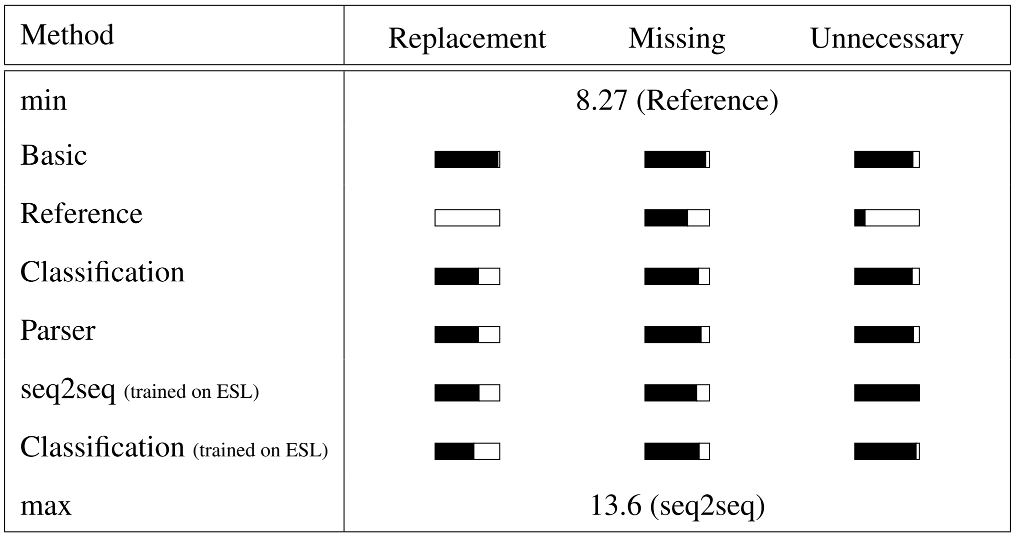
We also performed a set of error analysis experiments to examine the impact of various error types in this task. We observe that the performance of different methods varies with different error types; some error types are more problematic than others. The results of the error analysis would

(a) ESL dataset



Method	Replacement	Missing	Unnecessary
min	3.62 (Reference)		
Basic			
Reference			
Classification			
Parser			
seq2seq			
max	15.03 (Basic)		

(b) MT dataset



Method	Replacement	Missing	Unnecessary
min	8.27 (Reference)		
Basic			
Reference			
Classification			
Parser			
seq2seq (trained on ESL)			
Classification (trained on ESL)			
max	13.6 (seq2seq)		

Table 20: False Discovery Rates on sentences with one grammatical error, each can be categorized as a replacement word error, a missing word error or an unnecessary word error.

(a) ESL dataset

Method	Open class	Closed class
min	4.07 (Reference)	
Basic	██████████	████░░░░░░
Reference	░░░░░░░░░░	░░░░░░░░░░
Classification	█████░░░░░	█░░░░░░░░░
Parser	██████░░░░	█░░░░░░░░░
seq2seq	████░░░░░░	█░░░░░░░░░
max	16.45 (Basic)	

(b) MT dataset

Method	Open class	Closed class
min	7.6 (Reference)	
Basic	██████████	███████░░░
Reference	████░░░░░░	░░░░░░░░░░
Classification	█████░░░░░	██████░░░░
Parser	██████░░░░	████░░░░░░
seq2seq (trained on ESL)	██████░░░░	████░░░░░░
Classification (trained on ESL)	█████░░░░░	███░░░░░░░
max	14.55 (Basic)	

Table 21: False Discovery Rates on sentences with one error, where the error either occurs on an open-class (lexical) word or a closed-class (functional) word.

help researchers to adapt semantic role labelers to deal with ungrammatical text; they would also help to analyze the strength and weaknesses of different fragmentation methods on various error types to further improve them.

7.4 CHAPTER SUMMARY

We have applied the parse tree fragmentation framework in two downstream NLP applications. We have verified that the automatically extracted tree fragments are competitive with existing methods for making fluency judgments. Moreover, we evaluated parse tree fragmentation in the downstream NLP application of semantic role labeling and showed that fragmenting parse trees of ungrammatical sentences is helpful to detect their wrong semantic dependencies.

8.0 CONCLUSION AND FUTURE WORK

In this dissertation, we have examined the problems of parsing ungrammatical sentences. We have analyzed the negative impact that ungrammatical sentences have on the state-of-the-art statistical parsers and downstream applications that depend on accurate parse trees. We have introduced a new framework called parse tree fragmentation to address the challenges faced by these standard statistical parsers. The goal of parse tree fragmentation is to prune implausible dependency arcs of the parse trees. We have shown through empirical studies that fragmenting trees is helpful for natural language processing applications such as sentence-level grammaticality judgment and semantic role labeling. In the remaining of the chapter, we provide a summary of the contributions in this dissertation work and discuss how they address the thesis statements. Next, we propose some future research directions to further tackle this challenging NLP problem.

8.1 SUMMARY OF CONTRIBUTIONS AND RESULTS

The primary goal of this research was to investigate the impact of ungrammatical sentences on parsers. To accomplish this goal, we formulated three research questions and proposed methodologies to address them. In this section, we summarize the approaches that we took to deal with this problem, but there could be other directions even with better performances that we leave as the future work.

Question 1. In what ways does a parser's performance degrade when dealing with ungrammatical sentences?

To study the impact of ungrammatical sentences on statistical parsers, we have devised a robustness evaluation procedure and reported a set of empirical analysis on the performance of several leading parsers on these sentences. We have found that parsers indeed degrade and perform differently when dealing with ungrammatical sentences of various error types. The results of our error analysis would also help researchers to improve robustness of parsers in terms of various error types; they would also help practitioners to select an appropriate parser for their applications. Moreover, our results show that parsers do reasonably well when the dependency arcs that are related to the erroneous parts are ignored. This finding led us to approach parsing ungrammatical sentences by pruning their implausible dependency arcs.

Question 2. Is it feasible to automatically identify parse tree fragments that are plausible interpretations for the phrases they cover?

We have approached the problem of parsing ungrammatical sentences by proposing a new framework to re-interpret the parse trees by pruning the implausible dependency arcs. This results a set of tree fragments that are linguistically appropriate for the phrases they cover. We have proposed gold standard methods to automatically identify parse tree fragments using parallel corpora available for other NLP tasks; and we have used these methods to collect gold standard data. We have then proposed three automatic fragmentation methods that learn to fragment trees by training with the gold standard data: classification-based, parser-based, and sequence-to-sequence based methods. While these methods learn to fragment in a similar manner as the gold standard method, our studies suggest that the sequence-to-sequence mapping approach provides more accurate fragments. The sequence-to-sequence has an additional advantage in that it learns both to parse and fragment ungrammatical sentences. On the other hand, a drawback of this approach is that it needs a huge amount of parallel data that might not be available for some ungrammatical domains. While the Classification approach is applicable for domains that have a small but high quality error annotated ungrammatical sentences.

Question 3. Do the resulting parse tree fragments provide some useful information for downstream NLP applications?

We have investigated the utility of tree fragments for two NLP applications: sentence-level fluency judgment and semantic role labeling. Through experiments, we have found that parse tree fragmentation is helpful for these applications when dealing with ungrammatical sentences.

Especially applying the extracted features from the pseudo gold fragments significantly boosts the performance of two tasks. Although the pseudo gold fragments are considered as the upper bound, there is a performance gap between the automatic fragmentation methods and the pseudo gold fragments. One reason of this gap is that our trained fragmentation models are not optimal, for instance we did not search for the optimal training size for the Parser or the optimal size of the network for the sequence-to-sequence model, since our focus in this thesis was on validating the helpfulness of the fragmentation methods. In spite of the lower performance of automatic methods, our experiments show that they are still making reasonable decisions on fragmenting the trees; additionally, they are useful in judging fluency of sentences and detecting incorrect semantic dependencies. But it is apparent that there is still further scope for future improvements.

8.2 FUTURE WORK

Our study suggests that robustness evaluation of parsers and parse tree fragmentation framework are promising directions for further exploration. Although the approaches we proposed and the experiments we have conducted have shed some lights on parsing ungrammatical sentences, there are undoubtedly other directions and more sophisticated approaches that would lead to even more accomplishments. In this section, we discuss a number of areas for the future research on parser robustness and parser adaptation for ungrammatical sentences.

Parser Robustness

Our robustness evaluation study indicates that dependency parsers have different responses to ungrammatical sentences. This line of research can be further studied in several directions. First, since there are specialized parsers on different syntactic representation (dependency or constituency), it would be interesting to analyze the robustness of different parsers across the syntactic representation. In this case, our proposed robustness evaluation metric needs to be adapted to the constituency formalism.

Second, we have trained the parsers on two domains of the news texts and tweets data, and tested them on two domains of the learners' writings and machine translation outputs. One future direction is to expand these training and testing domains with the available treebanks on different domains of ungrammatical sentences. There is a newly released treebank for ESL writings that is manually annotated for erroneous sentences (Berzak et al., 2016). Although this is relatively a small corpus (containing around 5000 sentences), it can still be helpful to evaluate the robustness of parsers. Another treebank of noisy sentences is the Switchboard corpus (Godfrey et al., 1992), that contains Automatic Speech Recognition (ASR) transcripts of conversations and their manually annotated constituency parse trees. The error types of the annotated ASR transcripts is limited, but still it would be interesting to explore parser robustness on this corpus.

Parse Tree Fragmentation

Our proposed fragmentation framework consists of various parts that each could be optimized with more sophisticated methods; especially there is a performance gap between the proposed practical methods and the oracle which can be reduced by training a more powerful model for this task. Furthermore, since the focus of this thesis was on introducing the parse tree fragmentation and validating it, we have left finding the optimal models such as finding the optimal training size for the Parser and the optimal size of the network for the seq2seq method for the future work.

Furthermore, we have illustrated two possible use of tree fragments (for fluency assessment and semantic role labeling) to demonstrate how having tree fragments improves downstream applications when encountering ungrammatical sentences, but it would be interesting to apply fragmentation on a wider set of applications as well. A starting point could be based on the findings of the recent shared task of the Extrinsic Parser Evaluation (EPE)[1]; but still there is a need to collect annotated trees for the ungrammatical sentences to be able to evaluate them in the specific extrinsic applications of this shared task.

Parser Adaptation

We approached parsing ungrammatical sentences by introducing parse tree fragmentation, a framework to prune the incorrect dependency arcs of parse trees; another direction could be to build specialized parsers to handle these sentences. One approach is to adapt transition-based dependency parsers by adding new actions to handle grammatical mistakes in the sentences. This is more challenging than the previous work on jointly parsing and detecting disfluency in spoken

[1]http://epe.nlpl.eu/

utterances (Honnibal and Johnson, 2014; Yoshikawa et al., 2016), since there is a wider range of errors in written text. Another challenge is on collecting enough annotated data for training the adapted parser. An alternative to collect ungrammatical treebanks is to build one artificially; this could be done by adding simulated real world mistakes to grammatical sentences and alter their trees accordingly (Foster, 2007), but it still needs careful adaptation to filter out unrealistic grammatical mistakes.

APPENDIX

SEMANTIC ROLE LABELS

In this thesis, we use the PropBank style semantic role labels. A brief description of its semantic role labels are shown in Table 22. More details about PropBank semantic role labels are discussed in Bonial et al. (2010).

Label	**Description**
A0	Agent
A1	Patient, theme
A2	Instrument, benefactive, attribute
A3	Staring point
A4	Ending point
AM-MOD	Modals
AM-TMP	Temporal
AM-MNR	Manner
AM-LOC	Location
AM-DIR	Direction
AM-EXT	Extent
AM-REC	Reciprocals
AM-PRD	Secondary Predication
AM-PNC	Purpose
AM-CAU	Cause
AM-DIS	Discourse
AM-ADV	Adverbials
AM-NEG	Negation

Table 22: A list of semantic role labels.

BIBLIOGRAPHY

Abney, S. P. (1991). Parsing by chunks. In *Principle-Based Parsing*.

Akbik, A., Chiticariu, L., Danilevsky, M., Li, Y., Vaithyanathan, S., and Zhu, H. (2015). Generating high quality proposition banks for multilingual semantic role labeling. In *ACL*.

Akbik, A. and Li, Y. (2016). Polyglot: Multilingual semantic role labeling with unified labels. *ACL*.

Andor, D., Alberti, C., Weiss, D., Severyn, A., Presta, A., Ganchev, K., Petrov, S., and Collins, M. (2016). Globally normalized transition-based neural networks. *arXiv preprint arXiv:1603.06042*.

Baccianella, S., Esuli, A., and Sebastiani, F. (2009). Evaluation Measures for Ordinal Regression. *Intelligent Systems Design and Applications*, pages 283–287.

Bahdanau, D., Cho, K., and Bengio, Y. (2014). Neural machine translation by jointly learning to align and translate. *arXiv preprint arXiv:1409.0473*.

Baucom, E., King, L., and Kübler, S. (2013). Domain adaptation for parsing. In *Proceedings of the International Conference Recent Advances in Natural Language Processing*, pages 56–64.

Berka, J., Bojar, O., Fishel, M., Popovic, M., and Zeman, D. (2012). Automatic MT Error Analysis: Hjerson Helping Addicter. *LREC*, pages 2158–2163.

Berzak, Y., Kenney, J., Spadine, C., Wang, J. X., Lam, L., Mori, K. S., Garza, S., and Katz, B. (2016). Universal dependencies for learner English. In *ACL*, pages 737–746.

Bigert, J., Sjöbergh, J., Knutsson, O., and Sahlgren, M. (2005). Unsupervised evaluation of parser robustness. In *Computational Linguistics and Intelligent Text Processing*, pages 142–154.

Birch, A., Haddow, B., Germann, U., Nadejde, M., Buck, C., and Koehn, P. (2013). The feasibility of HMEANT as a human MT evaluation metric. In *Proceedings of the Eighth Workshop on Statistical Machine Translation*, pages 52–61.

Björkelund, A., Hafdell, L., and Nugues, P. (2009). Multilingual semantic role labeling. In *Proceedings of the Thirteenth Conference on Computational Natural Language Learning: Shared Task*, pages 43–48. Association for Computational Linguistics.

Black, E., Abney, S., Flickenger, S., Gdaniec, C., Grishman, C., Harrison, P., Hindle, D., Ingria, R., Jelinek, F., Klavans, J., Liberman, M., Marcus, M., Roukos, S., Santorini, B., and Strzalkowski, T. (1991). A procedure for quantitatively comparing the syntactic coverage of English grammars.

Bohnet, B. (2010). Very high accuracy and fast dependency parsing is not a contradiction. In *Proceedings of the 23rd International Conference on Computational Linguistics*, pages 89–97.

Bojar, O., Chatterjee, R., Federmann, C., Graham, Y., Haddow, B., Huck, M., Yepes, A. J., Koehn, P., Logacheva, V., Monz, C., et al. (2016). Findings of the 2016 conference on machine translation (WMT16). *Proceedings of WMT*.

Bonial, C., Babko-Malaya, O., Choi, J. D., Hwang, J., and Palmer, M. (2010). PropBank annotation guidelines. *Center for Computational Language and Education Research Institute of Cognitive Science University of Colorad at Boulder*.

Cahill, A. (2015). Parsing learner text: to shoehorn or not to shoehorn. In *The 9th Linguistic Annotation Workshop held in conjuncion with NAACL 2015*, page 144.

Charniak, E. and Johnson, M. (2005). Coarse-to-fine n-best parsing and maxent discriminative reranking. In *Proceedings of the 43rd Annual Meeting on Association for Computational Linguistics*, pages 173–180.

Chen, D. and Manning, C. D. (2014). A fast and accurate dependency parser using neural networks. In *Proceedings of the 2014 Conference on Empirical Methods in Natural Language Processing (EMNLP)*, volume 1, pages 740–750.

Cherry, C. and Quirk, C. (2008). Discriminative, syntactic language modeling through latent SVMs. *Proceeding of Association for Machine Translation in the America (AMTA)*, pages 21–25.

Cho, K., Van Merriënboer, B., Bahdanau, D., and Bengio, Y. (2014). On the properties of neural machine translation: Encoder-decoder approaches. *arXiv preprint arXiv:1409.1259*.

Choi, J. D., Tetreault, J., and Stent, A. (2015). It depends: Dependency parser comparison using a web-based evaluation tool. In *Proceedings of the 53rd Annual Meeting of the Association for Computational Linguistics*, pages 26–31.

Collobert, R., Weston, J., Bottou, L., Karlen, M., Kavukcuoglu, K., and Kuksa, P. (2011). Natural language processing (almost) from scratch. *The Journal of Machine Learning Research*, 12:2493–2537.

Dahlmeier, D., Ng, H. T., and Wu, S. M. (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*, pages 22–31.

Daiber, J. and van der Goot, R. (2016). The denoised web treebank: Evaluating dependency parsing under noisy input conditions. In *LREC*.

Dale, R., Anisimoff, I., and Narroway, G. (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*, pages 54–62.

Dale, R. and Kilgarriff, A. (2011). Helping our own: The HOO 2011 pilot shared task. In *Proceedings of the 13th European Workshop on Natural Language Generation*, pages 242–249.

Daudaravicius, V., Banchs, R. E., Volodina, E., and Napoles, C. (2016). A report on the automatic evaluation of scientific writing shared task. In *Workshop on Building Educational Applications Using NLP*, pages 53–62.

De Marneffe, M.-C., MacCartney, B., Manning, C. D., et al. (2006). Generating typed dependency parses from phrase structure parses. In *Proceedings of LREC*, volume 6, pages 449–454.

Denkowski, M. and Lavie, A. (2011). Meteor 1.3: Automatic Metric for Reliable Optimization and Evaluation of Machine Translation Systems. In *Proceedings of the EMNLP 2011 Workshop on Statistical Machine Translation*.

Dreyer, M., Smith, D. A., and Smith, N. A. (2006). Vine parsing and minimum risk reranking for speed and precision. In *Proceedings of the Tenth Conference on Computational Natural Language Learning*, pages 201–205.

Dridan, R. and Oepen, S. (2011). Parser evaluation using elementary dependency matching. In *Proceedings of the 12th International Conference on Parsing Technologies*, pages 225–230.

Eisenstein, J. (2013). What to do about bad language on the internet. *NAACL*, pages 359–369.

Eisner, J. and Smith, N. A. (2005). Parsing with soft and hard constraints on dependency length. In *Proceedings of the Ninth International Workshop on Parsing Technology*, pages 30–41.

Elming, J., Johannsen, A., Klerke, S., Lapponi, E., Alonso, H. M., and Søgaard, A. (2013). Downstream effects of tree-to-dependency conversions. In *HLT-NAACL*, pages 617–626.

Fan, R.-E., Chang, K.-W., Hsieh, C.-J., Wang, X.-R., and Lin, C.-J. (2008). Liblinear: A library for large linear classification. *The Journal of Machine Learning Research*, 9:1871–1874.

Ferguson, J., Durrett, G., and Klein, D. (2015). Disfluency detection with a semi-markov model and prosodic features. In *NAACL*.

Filippova, K., Alfonseca, E., Colmenares, C. A., Kaiser, L., and Vinyals, O. (2015). Sentence compression by deletion with LSTMs. In *EMNLP*, pages 360–368.

Fishel, M., Bojar, O., Zeman, D., and Berka, J. (2011). Automatic translation error analysis. *Text, Speech and Dialogue*.

FitzGerald, N., Täckström, O., Ganchev, K., and Das, D. (2015). Semantic role labeling with neural network factors. In *EMNLP*, pages 960–970.

Foland, W. and Martin, J. H. (2015). Dependency-based semantic role labeling using convolutional neural networks. In ** SEM, NAACL-HLT*, pages 279–288.

Foster, J. (2004). Parsing ungrammatical input: an evaluation procedure. In *LREC*.

Foster, J. (2007). Treebanks gone bad. *International Journal of Document Analysis and Recognition*, 10(3-4):129–145.

Foster, J. (2010). "cba to check the spelling" investigating parser performance on discussion forum posts. In *The Annual Conference of the North American Chapter of the Association for Computational Linguistics*, pages 381–384.

Foster, J., Çetinoglu, Ö., Wagner, J., Le Roux, J., Hogan, S., Nivre, J., Hogan, D., Van Genabith, J., et al. (2011a). # hardtoparse: POS tagging and parsing the twitterverse. In *proceedings of the Workshop On Analyzing Microtext (AAAI 2011)*, pages 20–25.

Foster, J., Cetinoglu, O., Wagner, J., and Roux, J. L. (2011b). From news to comment: Resources and benchmarks for parsing the language of web 2.0. *IJCNLP*.

Foster, J., Wagner, J., and Van Genabith, J. (2008). Adapting a WSJ-trained parser to grammatically noisy text. In *Proceedings of the Annual Meeting of the Association for Computational Linguistics*, pages 221–224.

Friedman, J. H. (2001). Greedy function approximation: a gradient boosting machine. *Annals of statistics*, pages 1189–1232.

Gamon, M. and Leacock, C. (2010). Search right and thou shalt find...: using web queries for learner error detection. In *Proceedings of the NAACL HLT 2010 Fifth Workshop on Innovative Use of NLP for Building Educational Applications*, pages 37–44.

Geertzen, J., Alexopoulou, T., and Korhonen, A. (2013). Automatic linguistic annotation of large scale l2 databases: the ef-cambridge open language database (EFCAMDAT). In *Proceedings of the 31st Second Language Research Forum*.

Georgila, K. (2009). Using integer linear programming for detecting speech disfluencies. In *Proceedings of Human Language Technologies: The 2009 Annual Conference of the North American Chapter of the Association for Computational Linguistics*, pages 109–112.

Gildea, D. (2001). Corpus variation and parser performance. In *Proceedings of the Conference on Empirical Methods in Natural Language Processing*, pages 167–202.

Gildea, D. and Jurafsky, D. (2002). Automatic labeling of semantic roles. *Computational linguistics*, 28(3):245–288.

Gimpel, K., Schneider, N., O'Connor, B., Das, D., Mills, D., Eisenstein, J., Heilman, M., Yogatama, D., Flanigan, J., and Smith, N. A. (2011). Part-of-speech tagging for Twitter: Annotation, features, and experiments. In *ACL-HLT*, pages 42–47.

Godfrey, J. J., Holliman, E. C., and McDaniel, J. (1992). Switchboard: Telephone speech corpus for research and development. In *International Conference on Acoustics, Speech, and Signal Processing (ICASSP)*, pages 517–520.

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

Hajič, J., Ciaramita, M., Johansson, R., Kawahara, D., Martí, M. A., Màrquez, L., Meyers, A., Nivre, J., Padó, S., Štěpánek, J., et al. (2009). The CoNLL-2009 shared task: Syntactic and semantic dependencies in multiple languages. In *Proceedings of the Thirteenth Conference on Computational Natural Language Learning: Shared Task*, pages 1–18.

Han, B. and Baldwin, T. (2011). Lexical normalisation of short text messages: Makn sens a# twitter. In *Proceedings of the 49th Annual Meeting of the Association for Computational Linguistics: Human Language Technologies*, pages 368–378.

Han, B., Cook, P., and Baldwin, T. (2012). Automatically constructing a normalisation dictionary for microblogs. In *Proceedings of the 2012 joint conference on empirical methods in natural language processing and computational natural language learning*, pages 421–432.

Hanley, J. A. and McNeil, B. J. (1982). The meaning and use of the area under a receiver operating characteristic (ROC) curve. *Radiology*, 143(1):29–36.

Hashemi, H. B. and Hwa, R. (2014). A comparison of MT errors and ESL errors. In *LREC*, pages 2696–2700.

Hashemi, H. B. and Hwa, R. (2016). Parse tree fragmentation of ungrammatical sentences. In *Proceedings of the Twenty-Fifth International Joint Conference on Artificial Intelligence (IJCAI).*

Heilman, M., Cahill, A., Madnani, N., and Tetreault, J. (2014). Predicting Grammaticality on an Ordinal Scale. *ACL*, pages 174–180.

Hochreiter, S. and Schmidhuber, J. (1997). Long short-term memory. *Neural computation*, 9(8):1735–1780.

Honnibal, M. and Johnson, M. (2014). Joint incremental disfluency detection and dependency parsing. *Transactions of the Association for Computational Linguistics*, 2:131–142.

Joshi, A. K. and Schabes, Y. (1997). Tree-adjoining grammars. *Handbook of formal languages*, 3:69–124.

Jurafsky, D. and Martin, J. H. (2009). *Speech and Language Processing An Introduction to Natural Language Processing, Computational Linguistics, and Speech*. Pearson Education.

Kakkonen, T. (2007). Robustness evaluation of two ccg, a pcfg and a link grammar parsers. *Proceedings of the 3rd Language & Technology Conference: Human Language Technologies as a Challenge for Computer Science and Linguistics.*

Kingsbury, P. and Palmer, M. (2002). From TreeBank to PropBank. In *LREC*.

Klein, D. and Manning, C. D. (2003). Accurate unlexicalized parsing. In *Proceedings of the Annual Meeting on Association for Computational Linguistics*, pages 423–430.

Klein, G., Kim, Y., Deng, Y., Senellart, J., and Rush, A. M. (2017). OpenNMT: Open-source toolkit for neural machine translation. *arXiv preprint arXiv:1701.02810*.

Kong, L., Schneider, N., Swayamdipta, S., Bhatia, A., Dyer, C., and Smith, N. A. (2014). A dependency parser for tweets. In *Proceedings of the Conference on Empirical Methods in Natural Language Processing*.

Kong, L. and Smith, N. A. (2014). An empirical comparison of parsing methods for stanford dependencies. *arXiv preprint arXiv:1404.4314*.

Kummerfeld, J. K., Hall, D., Curran, J. R., and Klein, D. (2012). Parser showdown at the wall street corral: An empirical investigation of error types in parser output. In *Proceedings of the 2012 Joint Conference on Empirical Methods in Natural Language Processing and Computational Natural Language Learning*, pages 1048–1059.

Lo, C.-k. and Wu, D. (2011). MEANT: an inexpensive, high-accuracy, semi-automatic metric for evaluating translation utility via semantic frames. In *Proceedings of the 49th Annual Meeting of the Association for Computational Linguistics: Human Language Technologies-Volume 1*, pages 220–229.

Maqsud, U., Arnold, S., Hülfenhaus, M., and Akbik, A. (2014). Nerdle: Topic-specific question answering using wikia seeds. In *25th International Conference on Computational Linguistics, Proceedings of the Conference System Demonstrations*, pages 81–85.

Marcus, M. P., Marcinkiewicz, M. A., and Santorini, B. (1993). Building a large annotated corpus of English: The penn treebank. *Computational linguistics*, 19(2):313–330.

Martins, A. F., Almeida, M., and Smith, N. A. (2013). Turning on the turbo: Fast third-order non-projective turbo parsers. In *Proceedings of the 51st Annual Meeting of the Association for Computational Linguistics*, pages 617–622. Citeseer.

McClosky, D., Charniak, E., and Johnson, M. (2006). Reranking and self-training for parser adaptation. In *Proceedings of the annual meeting of the Association for Computational Linguistics*, pages 337–344.

McClosky, D., Charniak, E., and Johnson, M. (2010). Automatic domain adaptation for parsing. In *The Annual Conference of the North American Chapter of the Association for Computational Linguistics*, pages 28–36.

McDonald, R. and Nivre, J. (2011). Analyzing and integrating dependency parsers. *Computational Linguistics*, 37(1):197–230.

McDonald, R. T. and Pereira, F. C. (2006). Online learning of approximate dependency parsing algorithms. In *EACL*.

Miyao, Y., Sætre, R., Sagae, K., Matsuzaki, T., and Tsujii, J. (2008). Task-oriented evaluation of syntactic parsers and their representations. In *ACL*, volume 8, pages 46–54.

Murphy, K. P. (2012). *Machine learning: a probabilistic perspective*. MIT press.

Mutton, A., Dras, M., Wan, S., and Dale, R. (2007). GLEU: Automatic evaluation of sentence-level fluency. In *ACL*.

Ng, H. T., Wu, S. M., Briscoe, T., Hadiwinoto, C., Susanto, R. H., and Bryant, C. (2014). The CoNLL-2014 shared task on grammatical error correction. In *CoNLL Shared Task*, pages 1–14.

Ng, H. T., Wu, S. M., Hadiwinoto, C., and Tetreault, J. (2013). The CoNLL-2013 shared task on grammatical error correction. In *Conference on Computational Natural Language Learning: Shared Task*, pages 1–12.

Nivre, J. (2004). Incrementality in deterministic dependency parsing. In *Proceedings of the Workshop on Incremental Parsing: Bringing Engineering and Cognition Together*, pages 50–57. Association for Computational Linguistics.

Nivre, J., Hall, J., Nilsson, J., Chanev, A., Eryigit, G., Kübler, S., Marinov, S., and Marsi, E. (2007). MaltParser: A language-independent system for data-driven dependency parsing. *Natural Language Engineering*, 13(02):95–135.

Oepen, S., Øvrelid, L., Björne, J., Johansson, R., Lapponi, E., Ginter, F., and Velldal, E. (2017). The 2017 Shared Task on Extrinsic Parser Evaluation. Towards a reusable community infrastructure. In *The 2017 Shared Task on Extrinsic Parser Evaluation (EPE)*, pages 1–16.

Ott, N. and Ziai, R. (2010). Evaluating dependency parsing performance on german learner language. *Proceedings of TLT-9*, 9:175–186.

Palmer, M., Gildea, D., and Kingsbury, P. (2005). The proposition bank: An annotated corpus of semantic roles. *Computational linguistics*, 31(1):71–106.

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

Pedregosa, F., Varoquaux, G., Gramfort, A., Michel, V., Thirion, B., Grisel, O., Blondel, M., Prettenhofer, P., Weiss, R., Dubourg, V., et al. (2011). Scikit-learn: Machine learning in Python. *The Journal of Machine Learning Research*, 12:2825–2830.

Petrov, S., Chang, P.-C., Ringgaard, M., and Alshawi, H. (2010). Uptraining for accurate deterministic question parsing. In *Proceedings of the Conference on Empirical Methods in Natural Language Processing*, pages 705–713.

Pinker, S. (2015). *The Sense of Style: The Thinking Person's Guide to Writing in the 21st Century!* Penguin Books.

Popović, M. and Ney, H. (2011). Towards automatic error analysis of machine translation output. *Computational Linguistics*.

Post, M. (2011). Judging grammaticality with tree substitution grammar derivations. In *Proceedings of the 49th Annual Meeting of the Association for Computational Linguistics*, pages 217–222.

Post, M. and Bergsma, S. (2013). Explicit and implicit syntactic features for text classification. In *ACL*, volume 2, pages 866–872.

Potet, M., Esperança-Rodier, E., Besacier, L., and Blanchon, H. (2012). Collection of a large database of French-English SMT output corrections. In *LREC*, pages 4043–4048.

Pradhan, S., Hacioglu, K., Ward, W., Martin, J. H., and Jurafsky, D. (2005). Semantic role chunking combining complementary syntactic views. In *Proceedings of the Ninth Conference on Computational Natural Language Learning*, pages 217–220. Association for Computational Linguistics.

Punyakanok, V., Roth, D., and Yih, W.-t. (2008). The importance of syntactic parsing and inference in semantic role labeling. *Computational Linguistics*, 34(2):257–287.

Qian, X. and Liu, Y. (2013). Disfluency detection using multi-step stacked learning. In *HLT-NAACL*, pages 820–825.

Quirk, C. and Corston-Oliver, S. (2006). The impact of parse quality on syntactically-informed statistical machine translation. In *Proceedings of the 2006 Conference on Empirical Methods in Natural Language Processing*, pages 62–69. Association for Computational Linguistics.

Ragheb, M. and Dickinson, M. (2012). Defining syntax for learner language annotation. In *COLING (Posters)*, pages 965–974.

Rasooli, M. S. and Tetreault, J. (2015). Yara parser: A fast and accurate dependency parser. *arXiv preprint arXiv:1503.06733*.

Rasooli, M. S. and Tetreault, J. R. (2013). Joint parsing and disfluency detection in linear time. In *EMNLP*, pages 124–129.

Resnik, P. and Lin, J. (2010). Evaluation of NLP Systems. *Handb. Comput. Linguist. Nat. Lang. Process.*, 57:271.

Ritter, A., Clark, S., and Etzioni, O. (2011). Named Entity Recognition in Tweets : An Experimental Study. *EMNLP*, pages 1524–1534.

Roark, B., Harper, M., Charniak, E., Dorr, B., Johnson, M., Kahn, J. G., Liu, Y., Ostendorf, M., Hale, J., Krasnyanskaya, A., et al. (2006). SParseval: Evaluation metrics for parsing speech. In *Proc. LREC*.

Roth, M. and Woodsend, K. (2014). Composition of word representations improves semantic role labelling. In *EMNLP*, pages 407–413.

Rozovskaya, A. and Roth, D. (2014). Building a state-of-the-art grammatical error correction system. *Transactions of the Association for Computational Linguistics*, 2:419–434.

Rozovskaya, A. and Roth, D. (2016). Grammatical error correction: Machine translation and classifiers. *ACL*.

Sagae, K. and Tsujii, J. (2007). Dependency parsing and domain adaptation with LR models and parser ensembles. In *EMNLP-CoNLL*, volume 2007, pages 1044–1050.

Sakaguchi, K., Post, M., and Van Durme, B. (2017). Error-repair dependency parsing for ungrammatical texts. In *ACL*.

Schmaltz, A., Kim, Y., Rush, A. M., and Shieber, S. M. (2016). Sentence-level grammatical error identification as sequence-to-sequence correction. *arXiv preprint arXiv:1604.04677*.

Schmaltz, A., Kim, Y., Rush, A. M., and Shieber, S. M. (2017). Adapting sequence models for sentence correction. *EMNLP*.

Serban, I. V., Sordoni, A., Bengio, Y., Courville, A., and Pineau, J. (2015). Building end-to-end dialogue systems using generative hierarchical neural network models. *arXiv preprint arXiv:1507.04808*.

Sha, F. and Pereira, F. (2003). Shallow parsing with conditional random fields. In *Proceedings of the 2003 Conference of the North American Chapter of the Association for Computational Linguistics*, pages 134–141.

Shen, D. and Lapata, M. (2007). Using semantic roles to improve question answering. In *Emnlp-conll*, pages 12–21.

Snover, M., Dorr, B., Schwartz, R., Micciulla, L., and Makhoul, J. (2006). A study of translation edit rate with targeted human annotation. In *Proceedings of association for machine translation in the Americas*, pages 223–231.

Sultan, M. A., Bethard, S., and Sumner, T. (2014). Back to basics for monolingual alignment: Exploiting word similarity and contextual evidence. *Transactions of the Association for Computational Linguistics*, 2:219–230.

Sun, X., Morency, L.-P., Okanohara, D., and Tsujii, J. (2008). Modeling latent-dynamic in shallow parsing: a latent conditional model with improved inference. In *Proceedings of the 22nd International Conference on Computational Linguistics*, pages 841–848.

Sutskever, I., Vinyals, O., and Le, Q. V. (2014). Sequence to sequence learning with neural networks. In *Advances in neural information processing systems*, pages 3104–3112.

Toutanova, K., Klein, D., Manning, C. D., and Singer, Y. (2003). Feature-rich part-of-speech tagging with a cyclic dependency network. In *NAACL*, pages 173–180.

Venugopalan, S., Rohrbach, M., Donahue, J., Mooney, R., Darrell, T., and Saenko, K. (2015). Sequence to sequence-video to text. In *Proceedings of the IEEE International Conference on Computer Vision*, pages 4534–4542.

Vilar, D., Xu, J., D'Haro, L., and Ney, H. (2006). Error analysis of statistical machine translation output. *Proc. Lr.*, pages 697–702.

Vinyals, O., Bengio, S., and Kudlur, M. (2015a). Order matters: Sequence to sequence for sets. *arXiv preprint arXiv:1511.06391*.

Vinyals, O., Kaiser, Ł., Koo, T., Petrov, S., Sutskever, I., and Hinton, G. (2015b). Grammar as a foreign language. In *Advances in Neural Information Processing Systems*, pages 2773–2781.

Wagner, J., Foster, J., and van Genabith, J. (2009). Judging grammaticality: Experiments in sentence classification. *CALICO Journal*, 26(3):474–490.

Wiebe, J., Wilson, T., and Cardie, C. (2005). Annotating expressions of opinions and emotions in language. *Language resources and evaluation*, 39(2):165–210.

Wiseman, S. and Rush, A. M. (2016). Sequence-to-sequence learning as beam-search optimization.

Wisniewski, G., Singh, A. K., Segal, N., and Yvon, F. (2013). Design and analysis of a large corpus of post-edited translations: quality estimation, failure analysis and the variability of post-edition. In *Machine Translation Summit*, volume 14, pages 117–124.

Wong, S.-M. J. and Dras, M. (2010). Parser features for sentence grammaticality classification. In *Proceedings of the Australasian Language Technology Association Workshop*, pages 67–75.

Xu, K., Ba, J., Kiros, R., Cho, K., Courville, A., Salakhudinov, R., Zemel, R., and Bengio, Y. (2015). Show, attend and tell: Neural image caption generation with visual attention. In *International Conference on Machine Learning*, pages 2048–2057.

Yannakoudakis, H., Briscoe, T., and Medlock, B. (2011). A new dataset and method for automatically grading ESOL texts. In *Proceedings of the 49th Annual Meeting of the Association for Computational Linguistics*, pages 180–189.

Yarmohammadi, M., Dunlop, A., and Roark, B. (2014). Transforming trees into hedges and parsing with "hedgebank" grammars. In *ACL (2)*, pages 797–802.

Yoshikawa, M., Shindo, H., and Matsumoto, Y. (2016). Joint transition-based dependency parsing and disfluency detection for automatic speech recognition texts. In *EMNLP*, pages 1036–1041.

Yuan, Z. and Briscoe, T. (2016). Grammatical error correction using neural machine translation. In *Proceedings of NAACL-HLT*, pages 380–386.

Zhou, J. and Xu, W. (2015). End-to-end learning of semantic role labeling using recurrent neural networks. In *Proceedings of the Annual Meeting of the Association for Computational Linguistics*.