An Evaluation of Parser Robustness for Ungrammatical Sentences

Homa B. Hashemi
Intelligent Systems Program
University of Pittsburgh
hashemi@cs.pitt.edu

Rebecca Hwa
Computer Science Department
University of Pittsburgh
hwa@cs.pitt.edu

Abstract

For many NLP applications that require a parser, the sentences of interest may not be well-formed. If the parser can overlook problems such as grammar mistakes and produce a parse tree that closely resembles the correct analysis for the intended sentence, we say that the parser is *robust*. This paper compares the performances of eight state-of-the-art dependency parsers on two domains of ungrammatical sentences: learner English and machine translation outputs. We have developed an evaluation metric and conducted a suite of experiments. Our analyses may help practitioners to choose an appropriate parser for their tasks, and help developers to improve parser robustness against ungrammatical sentences.

1 Introduction

Previous works have shown that, in general, parser performances degrade when applied to out-of-domain sentences (Gildea, 2001; McClosky et al., 2010; Foster, 2010; Petrov et al., 2010; Foster et al., 2011). If a parser performs reasonably well for a wide range of out-of-domain sentences, it is said to be *robust* (Bigert et al., 2005; Kakkonen, 2007; Foster, 2007).

Sentences that are ungrammatical, awkward, or too casual/colloquial can all be seen as special kinds of out-of-domain sentences. These types of sentences are commonplace for NLP applications, from product reviews and social media analysis to intelligent language tutors and multilingual processing. Since parsing is an essential component for many applications, it is natural to ask: Are some parsers more robust than others against sentences that are not well-formed? Previous works on parser evaluation that focused on accuracy and speed (Choi et al., 2015; Kummerfeld et al., 2012; McDonald and Nivre, 2011; Kong and Smith, 2014) have not taken ungrammatical sentences into consideration.

In this paper, we report a set of empirical analyses of eight leading dependency parsers on two domains of ungrammatical text: English-as-a-Second Language (ESL) learner text and machine translation (MT) outputs. We also vary the types of training sources; the parsers are trained with the Penn Treebank (to be comparable with other studies) and Tweebank, a treebank on tweets (to be a bit more like the test domain) (Kong et al., 2014).

The main contributions of the paper are:

- a metric and methodology for evaluating ungrammatical sentences without referring to a gold standard corpus;
- 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; and
- 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.

2 Evaluation of Parser Robustness

Parser evaluation for ungrammatical texts presents some domain-specific challenges. The typical approach to evaluate parsers is to compare parser out-

Proceedings of the 2016 Conference on Empirical Methods in Natural Language Processing, pages 1765–1774,
Austin, Texas, November 1-5, 2016. ©2016 Association for Computational Linguistics

puts against manually annotated gold standards. Although there are a few small semi-manually constructed treebanks on learner texts (Geertzen et al., 2013; Ott and Ziai, 2010) or tweets (Daiber and van der Goot, 2016), their sizes make them unsuitable for the evaluation of parser robustness. Moreover, some researchers have raised valid questions about the merit of creating a treebank for ungrammatical sentences or adapting the annotation schema (Cahill, 2015; Ragheb and Dickinson, 2012).

A "gold-standard free" alternative is to compare the parser output for each problematic 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 problematic 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 the robustness of four parsers over sentences with misspelled words.

Our proposed evaluation methodology is 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 Proposed Evaluation Methodology

For the purpose of robustness evaluation, we take the automatically produced parse tree of a well-formed sentence as "gold-standard" and compare the parser output for the corresponding problematic sentence against it. 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.

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 1), 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 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 -

[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.

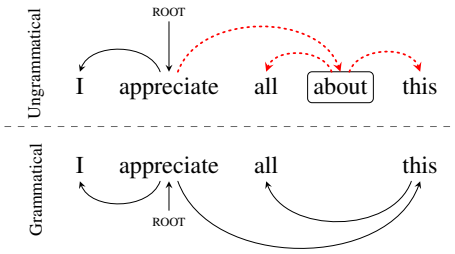

Figure 1: Example of evaluating robustness of an ungrammatical sentence (top) dependency parse tree with its corresponding grammatical sentence (bottom).

# 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); and

- *Robustness* F_1 is the harmonic mean of precision and recall.

Figure 1 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. The two shared dependencies between the trees result in a precision of $2/(5-3) = 1$, recall of $2/(4-0) = 0.5$, and Robustness F_1 of 66%.

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 the robustness of parsers over two datasets that contain ungrammatical sentences: writings of English-as-a-Second language learners and machine translation outputs. We choose datasets for which the corresponding correct sentences are available (or easily reconstructed).

4.1 Parsers

Our evaluation is over eight state-of-the-art dependency parsers representing a wide range of approaches. We use the publicly available versions of each parser 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.

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 the Turbo Parser specialized to parse tweets. Tweebo Parser adds a new constraint to the Turbo Parser's integer linear programming to ignore some Twitter tokens from parsing, but also simultaneously uses these tokens as parsing features.

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

4.2 Data

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

4.2.1 Parser Training Data

Penn Treebank (PTB) We follow the standard splits of Penn Treebank, using section 2-21 for training, section 22 for development, and section 23 for

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

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 pretrained 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)).

4.2.2 Robustness Test Data

To test the robustness of parsers, we choose two datasets of ungrammatical sentences for which their corresponding correct sentences are available. For a fair comparison, we automatically assign POS tags to the test data. When parsers are trained on PTB, we use the 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).

English-as-a-Second Language corpus (ESL) For the ungrammatical sentences, we use the First Certificate in English (FCE) dataset (Yannakoudakis et al., 2011) that contains the 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. 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) Machine translation outputs are another domain of ungrammatical sentences. We use the LIG (Potet et al., 2012) which contains 10,881 and LISMI's TRACE corpus[11] which contains 6,693 French-to-English machine translation outputs and their human post-editions. From these corpora, we randomly select 10,000 sentences with at least edit distance one (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.

4.3 Evaluation Metric

In the robustness evaluation metric (Section 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 FCE and MT data in a slightly different way. In the FCE 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)[12] 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. However, we ignore punctuations when parsers are trained with the Tweebank data, because punctuations are not annotated in the tweets with their dependencies.

5 Experiments

The experiments aim to address the following questions given separate training and test data:

1. How do parsers perform on erroneous sentences? (Section 5.1)

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

[11] anrtrace.limsi.fr/trace_postedit.tar.bz2

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

Parser	Train on PTB §1-21			Train on Tweebank$_{train}$		
	UAS	Robustness F_1		UAF_1	Robustness F_1	
	PTB §23	ESL	MT	Tweebank$_{test}$	ESL	MT
Malt	*89.58*	93.05	76.26	77.48	**94.36**	80.66
Mate	**93.16**	93.24	77.07	76.26	91.83	75.74
MST	91.17	*92.80*	76.51	73.99	92.37	77.71
SNN	90.70	93.15	74.18	*53.4*	88.90	*71.54*
SyntaxNet	93.04	93.24	76.39	75.75	*88.78*	**81.87**
Turbo	92.84	**93.72**	**77.79**	79.42	93.28	78.26
Tweebo	-	-	-	**80.91**	93.39	79.47
Yara	93.09	93.52	*73.15*	78.06	93.04	75.83

Table 1: 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.

2. To what extent is each parser negatively impacted by the increase in the number of errors in sentences? (Section 5.2)

3. To what extent is each parser negatively impacted by the interactions between multiple errors? (Section 5.3)

4. What types of errors are more problematic for parsers? (Section 5.4)

5.1 Overall Accuracy and Robustness

The overall performances of all parsers are shown in Table 1. 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 1 shows that, for both training conditions, the parser that has the best robustness score in the 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.

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 2 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 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,

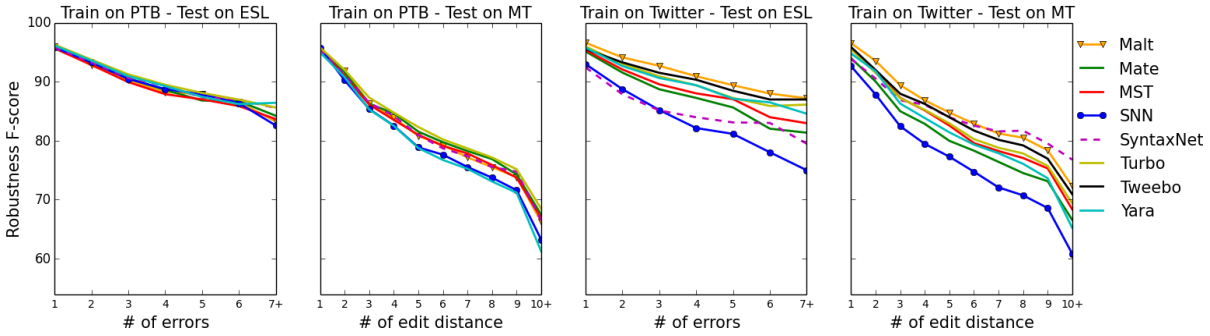

Figure 2: Variation in parser robustness as the number of errors in the test sentences increases.

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.

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. 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 each other with the robustness when the errors are far apart.

Table 2 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 indicates 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[13], 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 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).

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.

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

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

Parser	Train on PTB §1-21				Train on Tweebank$_{train}$			
	ESL		MT		ESL		MT	
	Near	Far	Near	Far	Near	Far	Near	Far
min	87.3 (Yara)		79.1 (Yara)		82.4 (SyntaxNet)		80.6 (SNN)	
Malt								
Mate								
MST								
SNN								
SyntaxNet								
Turbo								
Tweebo								
Yara								
max	93.4 (SNN)		91.5 (Yara)		94.5 (Malt)		94.4 (Malt)	

Table 2: Parser performance on test sentences with three near and three 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.

Parser	Train on PTB §1-21						Train on Tweebank$_{train}$					
	ESL			MT			ESL			MT		
	Repl.	Miss.	Unnec.	Repl.	Miss.	Unnec.	Repl.	Miss.	Unnec.	Repl.	Miss.	Unnec.
min	93.7 (MST)			92.8 (Yara)			89.4 (SyntaxNet)			87.8 (SNN)		
Malt												
Mate												
MST												
SNN												
SyntaxNet												
Turbo												
Tweebo												
Yara												
max	96.9 (Turbo)			97.2 (SNN)			97.8 (Malt)			97.6 (Malt)		

Table 3: Parser robustness on sentences with one grammatical error, each can be categorized as a replacement error, a missing word error or an unnecessary word error.

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 3, replacement errors are the least problematic error type for all the parsers; on the other hand, missing 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.

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.

Table 4 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.

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

Parser	Train on PTB §1-21				Train on Tweebank$_{train}$			
	ESL		MT		ESL		MT	
	Open class	Closed class	Open class	Closed class	Open class	Closed class	Open class	Closed class
min	95.1 (SNN)		94.5 (Yara)		89.6 (SyntaxNet)		91.5 (SNN)	
Malt								
Mate								
MST								
SNN								
SyntaxNet								
Turbo								
Tweebo								
Yara								
max	96.8 (Malt)		96.1 (SNN)		97.6 (Malt)		97.0 (Malt)	

Table 4: Parser robustness on sentences with one error, where the error either occurs on an open-class (lexical) word or a closed-class (functional) word.

Parser	Train on PTB §1-21						Train on Tweebank$_{train}$					
	ESL			MT			ESL			MT		
	Verb	Argument	No role	Verb	Argument	No role	Verb	Argument	No role	Verb	Argument	No role
min	94.1 (SyntaxNet)			91.8 (Malt)			91.8 (SNN)			92.2 (SNN)		
Malt												
Mate												
MST												
SNN												
SyntaxNet												
Turbo												
Tweebo												
Yara												
max	96.7 (Turbo)			96.7 (SyntaxNet)			96.9 (Malt)			96.9 (Malt)		

Table 5: 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.

sentences. We then obtain the role of the errors using alignments between ungrammatical sentence and its corrected counterpart.

Table 5 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.

6 Conclusions and Recommendations

In this paper, 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 find that parsers indeed have different responses to ungrammatical sentences of various types. 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 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 are good choices.

Our results also suggest that some preprocessing steps may be necessary for ungrammatical sentences, such as handling redundant and missing word errors. While there are some previous works on fixing the unnecessary words in the literature (Xue and Hwa, 2014), it is worthy to develop better NLP methods for catching and mitigating the missing word errors prior to parsing. Finally, this work corroborate the importance of building grammar error correction systems for handling closed-class er-

rors such as preposition errors.

Acknowledgments

This work was supported in part by the National Science Foundation Award #1550635. We would like to thank the anonymous reviewers and the Pitt NLP group for their helpful comments.

References

Daniel Andor, Chris Alberti, David Weiss, Aliaksei Severyn, Alessandro Presta, Kuzman Ganchev, Slav Petrov, and Michael Collins. 2016. Globally normalized transition-based neural networks. *arXiv preprint arXiv:1603.06042*.

Johnny Bigert, Jonas Sjöbergh, Ola Knutsson, and Magnus Sahlgren. 2005. Unsupervised evaluation of parser robustness. In *Computational Linguistics and Intelligent Text Processing*, pages 142–154.

E. Black, S. Abney, S. Flickenger, C. Gdaniec, C. Grishman, P. Harrison, D. Hindle, R. Ingria, F. Jelinek, J. Klavans, M. Liberman, M. Marcus, S. Roukos, B. Santorini, and T. Strzalkowski. 1991. A procedure for quantitatively comparing the syntactic coverage of English grammars. In *Proceedings of the DARPA Speech and Natural Language Workshop*, pages 306–311.

Bernd Bohnet. 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.

Aoife Cahill. 2015. Parsing learner text: to shoehorn or not to shoehorn. In *Proceedings of LAW IX - The 9th Linguistic Annotation Workshop*, page 144.

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

Jinho D Choi, Joel Tetreault, and Amanda Stent. 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.

Ronan Collobert, Jason Weston, Léon Bottou, Michael Karlen, Koray Kavukcuoglu, and Pavel Kuksa. 2011. Natural language processing (almost) from scratch. *The Journal of Machine Learning Research*, 12:2493–2537.

Joachim Daiber and Rob van der Goot. 2016. The denoised web treebank: Evaluating dependency parsing under noisy input conditions. In *LREC*.

Marie-Catherine De Marneffe, Bill MacCartney, Christopher D Manning, et al. 2006. Generating typed dependency parses from phrase structure parses. In *LREC*, number 2006, pages 449–454.

Jennifer Foster, Özlem Çetinoglu, Joachim Wagner, Joseph Le Roux, Stephen Hogan, Joakim Nivre, Deirdre Hogan, Josef Van Genabith, et al. 2011. # hardtoparse: POS tagging and parsing the twitterverse. In *proceedings of the Workshop On Analyzing Microtext (AAAI 2011)*, pages 20–25.

Jennifer Foster. 2004. Parsing ungrammatical input: an evaluation procedure. In *LREC*.

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

Jennifer Foster. 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.

Jeroen Geertzen, Theodora Alexopoulou, and Anna Korhonen. 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*.

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

Kevin Gimpel, Nathan Schneider, Brendan O'Connor, Dipanjan Das, Daniel Mills, Jacob Eisenstein, Michael Heilman, Dani Yogatama, Jeffrey Flanigan, and Noah A Smith. 2011. Part-of-speech tagging for Twitter: Annotation, features, and experiments. In *ACL-HLT*, pages 42–47.

Tuomo Kakkonen. 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*.

Lingpeng Kong and Noah A Smith. 2014. An empirical comparison of parsing methods for stanford dependencies. *arXiv preprint arXiv:1404.4314*.

Lingpeng Kong, Nathan Schneider, Swabha Swayamdipta, Archna Bhatia, Chris Dyer, and Noah A Smith. 2014. A dependency parser for tweets. In *Proceedings of the Conference on Empirical Methods in Natural Language Processing*.

Jonathan K Kummerfeld, David Hall, James R Curran, and Dan Klein. 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.

André FT Martins, Miguel Almeida, and Noah A Smith. 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.

David McClosky, Eugene Charniak, and Mark Johnson. 2010. Automatic domain adaptation for parsing. In *The Annual Conference of the North American Chapter of the Association for Computational Linguistics*, pages 28–36.

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

Ryan T McDonald and Fernando CN Pereira. 2006. Online learning of approximate dependency parsing algorithms. In *EACL*.

Joakim Nivre, Johan Hall, Jens Nilsson, Atanas Chanev, Gülsen Eryigit, Sandra Kübler, Svetoslav Marinov, and Erwin Marsi. 2007. Maltparser: A language-independent system for data-driven dependency parsing. *Natural Language Engineering*, 13(02):95–135.

Niels Ott and Ramon Ziai. 2010. Evaluating dependency parsing performance on german learner language. *Proceedings of the Ninth Workshop on Treebanks and Linguistic Theories (TLT-9)*, 9:175–186.

Slav Petrov, Pi-Chuan Chang, Michael Ringgaard, and Hiyan Alshawi. 2010. Uptraining for accurate deterministic question parsing. In *Proceedings of the Conference on Empirical Methods in Natural Language Processing*, pages 705–713.

Marion Potet, Emmanuelle Esperança-Rodier, Laurent Besacier, and Hervé Blanchon. 2012. Collection of a large database of French-English SMT output corrections. In *LREC*, pages 4043–4048.

Vasin Punyakanok, Dan Roth, and Wen-tau Yih. 2008. The importance of syntactic parsing and inference in semantic role labeling. *Computational Linguistics*, 34(2):257–287.

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

Mohammad Sadegh Rasooli and Joel Tetreault. 2015. Yara parser: A fast and accurate dependency parser. *arXiv preprint arXiv:1503.06733*.

Brian Roark, Mary Harper, Eugene Charniak, Bonnie Dorr, Mark Johnson, Jeremy G Kahn, Yang Liu, Mari Ostendorf, John Hale, Anna Krasnyanskaya, et al. 2006. Sparseval: Evaluation metrics for parsing speech. In *LREC*.

Kristina Toutanova, Dan Klein, Christopher D Manning, and Yoram Singer. 2003. Feature-rich part-of-speech tagging with a cyclic dependency network. In *NAACL*, pages 173–180.

Huichao Xue and Rebecca Hwa. 2014. Redundancy detection in esl writings. In *EACL*, pages 683–691.

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