# Does Size Matter? Text and Grammar Revision for Parsing Social Media Data

**Mohammad Khan**
Indiana University
Bloomington, IN USA
khanms@indiana.edu

**Markus Dickinson**
Indiana University
Bloomington, IN USA
md7@indiana.edu

**Sandra Kübler**
Indiana University
Bloomington, IN USA
skuebler@indiana.edu

## Abstract

We explore improving parsing social media and other web data by altering the input data, namely by normalizing web text, and by revising output parses. We find that text normalization improves performance, though spell checking has more of a mixed impact. We also find that a very simple tree reviser based on grammar comparisons performs slightly but significantly better than the baseline and well outperforms a machine learning model. The results also demonstrate that, more than the size of the training data, the goodness of fit of the data has a great impact on the parser.

## 1 Introduction and Motivation

Parsing data from social media data, as well as other data from the web, is notoriously difficult, as parsers are generally trained on news data (Petrov and McDonald, 2012), which is not a good fit for social media data. The language used in social media does not follow standard conventions (e.g., containing many sentence fragments), is largely unedited, and tends to be on different topics than standard NLP technology is trained for. At the same time, there is a clear need to develop even basic NLP technology for a variety of types of social media and contexts (e.g., Twitter, Facebook, YouTube comments, discussion forums, blogs, etc.). To perform tasks such as sentiment analysis (Nakagawa et al., 2010) or information extraction (McClosky et al., 2011), it helps to perform tagging and parsing, with an eye towards providing a shallow semantic analysis.

We advance this line of research by investigating adapting parsing to social media and other web data. Specifically, we focus on two areas: 1) We compare the impact of various text normalization techniques on parsing web data; and 2) we explore parse revision techniques for dependency parsing web data to improve the fit of the grammar learned by the parser.

One of the major problems in processing social media data is the common usage of non-standard terms (e.g., *kawaii*, a Japanese-borrowed net term for 'cute'), ungrammatical and (intentionally) misspelled text (e.g., *cuttie*), emoticons, and short posts with little contextual information, as exemplified in (1).[1]

(1) Awww cuttie little kitten, so Kawaii <3

To process such data, with its non-standard words, we first develop techniques for normalizing the text, so as to be able to accommodate the wide range of realizations of a given token, e.g., all the different spellings and intentional misspellings of *cute*. While previous research has shown the benefit of text normalization (Foster et al., 2011; Gadde et al., 2011; Foster, 2010), it has not teased apart which parts of the normalization are beneficial under which circumstances.

A second problem with parsing social media data is the data situation: parsers can be trained on the standard training set, the Penn Treebank (Marcus et al., 1993), which has a sufficient size for training a statistical parser, but has the distinct downside of modeling language that is very dissimilar

[1]Taken from: `http://www.youtube.com/watch?v=eHSpHCprXLA`

from the target. Or one can train parsers on the English Web Treebank (Bies et al., 2012), which covers web language, including social media data, but is rather small. Our focus on improving parsing for such data is on exploring parse revision techniques for dependency parsers. As far as we know, despite being efficient and trainable on a small amount of data, parse revision (Henestroza Anguiano and Candito, 2011; Cetinoglu et al., 2011; Attardi and Dell'Orletta, 2009; Attardi and Ciaramita, 2007) has not been used for web data, or more generally for adapting a parser to out-of-domain data; an investigation of its strengths and weaknesses is thus needed.

We describe the data sets used in our experiments in section 2 and the process of normalization in section 3 before turning to the main task of parsing in section 4. Within this section, we discuss our main parser as well as two different parse revision methods (sections 4.2 and 4.3). In the evaluation in section 5, we will find that normalization has a positive impact, although spell checking has mixed results, and that a simple tree anomaly detection method (Dickinson and Smith, 2011) outperforms a machine learning reviser (Attardi and Ciaramita, 2007), especially when integrated with confidence scores from the parser itself. In addition to the machine learner requiring a weak baseline parser, some of the main differences include the higher recall of the simple method at positing revisions and the fact that it detects odd structures, which parser confidence can then sort out as incorrect or not.

## 2 Data

For our experiments, we use two main resources, the Wall Street Journal (WSJ) portion of the Penn Treebank (PTB) (Marcus et al., 1993) and the English Web Treebank (EWT) (Bies et al., 2012). The two corpora were converted from PTB constituency trees into dependency trees using the Stanford dependency converter (de Marneffe and Manning, 2008).[2]

The EWT is comprised of approximately 16,000 sentences from weblogs, newsgroups, emails, reviews, and question-answers. Instead of examining each group individually, we chose to treat all web data equally, pulling from each type of data in the training/testing split.

```
1 <<_ -LRB--LRB-_ 2 punct _ _
2 File _ NN NN _ 0 root _ _
3 : _ : : _ 2 punct _ _
4 220b _ GW GW _ 11 dep _ _
5 -_ GW GW _ 11 dep _ _
6 dg _ GW GW _ 11 dep _ _
7 -_ GW GW _ 11 dep _ _
8 Agreement _ GW GW _ 11 dep _ _
9 for _ GW GW _ 11 dep _ _
10 Recruiting _ GW GW _ 11 dep _ _
11 Services.doc _ NN NN _ 2 dep _ _
12 >>_ -RRB--RRB-_ 2 punct _ _
13 <<_ -LRB--LRB-_ 14 punct _ _
14 File _ NN NN _ 2 dep _ _
15 : _ : : _ 14 punct _ _
16 220a _ GW GW _ 22 dep _ _
17 DG _ GW GW _ 22 dep _ _
18 -_ GW GW _ 22 dep _ _
19 Agreement _ GW GW _ 22 dep _ _
20 for _ GW GW _ 22 dep _ _
21 Contract _ GW GW _ 22 dep _ _
22 Services.DOC _ NN NN _ 14 dep _ _
23 >>_ -RRB--RRB-_ 14 punct _ _
```

Figure 1: A sentence with GW POS tags.

Additionally, for our experiments, we deleted the 212 sentences from EWT that contain the POS tags AFX and GW tags. EWT uses the POS tag AFX for cases where a prefix is written as a separate word from its root, e.g., *semi*/AFX *automatic*/JJ. Such segmentation and tagging would interfere with our normalization process. The POS tag GW is used for other non-standard words, such as document names. Such "sentences" are often difficult to analyze and do not correspond to phenomena found in the PTB (cf., figure 1).

To create training and test sets, we broke the data into the following sets:

- WSJ training: sections 02-22 (42,009 sentences)
- WSJ testing: section 23 (2,416 sentences)
- EWT training: 80% of the data, taking the first four out of every five sentences (13,130 sentences)
- EWT testing: 20% of the data, taking every fifth sentence (3,282 sentences)

[2]http://nlp.stanford.edu/software/stanford-dependencies.shtml

## 3 Text normalization

Previous work has shown that accounting for variability in form (e.g., misspellings) on the web, e.g., by mapping each form to a normalized form (Foster, 2010; Gadde et al., 2011) or by delexicalizing the parser to reduce the impact of unknown words (Øvrelid and Skjærholt, 2012), leads to some parser or tagger improvement. Foster (2010), for example, lists adapting the parser's unknown word model to handle capitalization and misspellings of function words as a possibility for improvement. Gadde et al. (2011) find that a model which posits a corrected sentence and then is POS-tagged—their tagging after correction (TAC) model—outperforms one which cleans POS tags in a postprocessing step. We follow this line of inquiry by developing text normalization techniques prior to parsing.

### 3.1 Basic text normalization

Machine learning algorithms and parsers are sensitive to the surface form of words, and different forms of a word can mislead the learner/parser. Our basic text normalization is centered around the idea that reducing unnecessary variation will lead to improved parsing performance.

For basic text normalization, we reduce all web URLs to a single token, i.e., each web URL is replaced with a uniform place-holder in the entire EWT, marking it as a URL. Similarly, all emoticons are replaced by a single marker indicating an emoticon. Repeated use of punctuation, e.g., *!!!*, is reduced to a single punctuation token.

We also have a module to shorten words with consecutive sequences of the same character: Any character that occurs more than twice in sequence will be shortened to one character, unless they appear in a dictionary, including the internet and slang dictionaries discussed below, in which case they map to the dictionary form. Thus, the word *Awww* in example (1) is shortened to *Aw*, and *cooool* maps to the dictionary form *cool*. However, since we use gold POS tags for our experiments, this module is not used in the experiments reported here.

### 3.2 Spell checking

Next, we run a spell checker to normalize misspellings, as online data often contains spelling errors (e.g. *cuttie* in example (1)). Various systems for parsing web data (e.g., from the SANCL shared task) have thus also explored spelling correction; McClosky et al. (2012), for example, used 1,057 autocorrect rules, though—since these did not make many changes—the system was not explored after that. Spell checking web data, such as YouTube comments or blog data, is a challenge because it contains non-standard orthography, as well as acronyms and other short-hand forms unknown to a standard spelling dictionary. Therefore, before mapping to a corrected spelling, it is vital to differentiate between a misspelled word and a non-standard one.

We use Aspell[3] as our spell checker to recognize and correct misspelled words. If asked to correct non-standard words, the spell checker would choose the closest standard English word, inappropriate to the context. For example, Aspell suggests *Lil* for *lol*. Thus, before correcting, we first check whether a word is an instance of *internet speech*, i.e., an abbreviation or a slang term.

We use a list of more than 3,000 acronyms to identify acronyms and other abbreviations not used commonly in formal registers of language. The list was obtained from NetLingo, restricted to the entries listed as chat acronyms and text message shorthand.[4] To identify slang terminology, we use the Urban Dictionary[5]. In a last step, we combine both lists with the list of words extracted from the WSJ.

If a word is not found in these lists, Aspell is used to suggest a correct spelling. In order to restrict Aspell from suggesting spellings that are too different from the word in question, we use Levenshtein distance (Levenshtein, 1966) to measure the degree of similarity between the original form and the suggested spelling; only words with small distances are accepted as spelling corrections. Since we have words of varying length, the Levenshtein distance is normalized by the length of the suggested spelling (i.e., number of characters). In non-exhaustive tests on a subset of the test set, we found that a normalized score of 0.301, i.e., a relatively low score accepting only conservative changes, achieves the best results when used as a threshold for accepting a sug-

[3]`www.aspell.net`
[4]`http://www.netlingo.com/acronyms.php`
[5]`www.urbandictionary.com`

gested spelling. The utilization of the threshold restricts Aspell from suggesting wrong spellings for a majority of the cases. For example, for the word *mujahidin*, Aspell suggested *Mukden*, which has a score of 1.0 and is thus rejected. Since we do not consider context or any other information besides edit distance, spell checking is not perfect and is subject to making errors, but the number of errors is considerably smaller than the number of correct revisions. For example, *lol* would be changed into *Lil* if it were not listed in the extended lexicon. Additionally, since the errors are consistent throughout the data, they result in normalization even when the spelling is wrong.

## 4 Parser revision

We use a state of the art dependency parser, MSTParser (McDonald and Pereira, 2006), as our main parser; and we use two parse revision methods: a machine learning model and a simple tree anomaly model. The goal is to be able to learn where the parser errs and to adjust the parses to be more appropriate given the target domain of social media texts.

### 4.1 Basic parser

MSTParser (McDonald and Pereira, 2006)[6] is a freely available parser which reaches state-of-the-art accuracy in dependency parsing for English. MST is a graph-based parser which optimizes its parse tree globally (McDonald et al., 2005), using a variety of feature sets, i.e., edge, sibling, context, and non-local features, employing information from words and POS tags. We use its default settings for all experiments.

We use MST as our base parser, training it in different conditions on the WSJ and the EWT. Also, MST offers the possibility to retrieve confidence scores for each dependency edge: We use the KD-Fix edge confidence scores discussed by Mejer and Crammer (2012) to assist in parse revision. As described in section 4.4, the scores are used to limit which dependencies are candidates for revision: if a dependency has a low confidence score, it may be revised, while high confidence dependencies are not considered for revision.

[6]`http://sourceforge.net/projects/mstparser/`

### 4.2 Reviser #1: machine learning model

We use DeSR (Attardi and Ciaramita, 2007) as a machine learning model of parse revision. DeSR uses a tree revision method based on decomposing revision actions into basic graph movements and learning sequences of such movements, referred to as a *revision rule*. For example, the rule `-1u` indicates that the reviser should change a dependent's head one word to the left (`-1`) and then up one element in the tree (`u`). Note that DeSR only changes the heads of dependencies, but not their labels. Such revision rules are learned for a base parser by comparing the base parser output and the gold-standard of some unseen data, based on a maximum entropy model.

In experiments, DeSR generally only considers the most frequent rules (e.g., 20), as these cover most of the errors. For best results, the reviser should: a) be trained on extra data other than the data the base parser is trained on, and b) begin with a relatively poor base parsing model. As we will see, using a fairly strong base parser presents difficulties for DeSR.

### 4.3 Reviser #2: simple tree anomaly model

Another method we use for building parse revisions is based on a method to detect anomalies in parse structures (APS) using $n$-gram sequences of dependency structures (Dickinson and Smith, 2011; Dickinson, 2010). The method checks whether the same head category (e.g., verb) has a set of dependents similar to others of the same category (Dickinson, 2010).

To see this, consider the partial tree in figure 2, from the dependency-converted EWT.[7] This tree is converted to a rule as in (2), where all dependents of a head are realized.

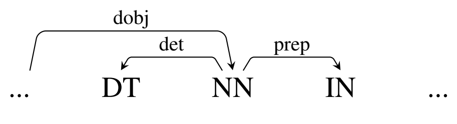

Figure 2: A sketch of a basic dependency tree

(2) dobj → det:DT NN **prep:IN**

[7]DT/det=determiner, NN=noun, IN/prep=preposition, dobj=direct object

This rule is then broken down into its component $n$-grams and compared to other rules, using the formula for scoring an element ($e_i$) in (3). $N$-gram counts ($C(ngrm)$) come from a training corpus; an instantiation for this rule is in (4).

(3) $s(e_i) = \sum_{ngrm: e_i \in ngrm \wedge n \geq 3} C(ngrm)$

(4) $s$(prep:IN) = $C$(det:DT NN **prep:IN**)
+ $C$(NN **prep:IN** END)
+ $C$(START det:DT NN **prep:IN**)
+ $C$(det:DT NN **prep:IN** END)
+ $C$(START det:DT NN **prep:IN** END)

We modify the scoring slightly, incorporating bigrams ($n \geq 2$), but weighing them as 0.01 of a count ($C(ngrm)$); this handles the issue that bigrams are not very informative, yet having some bigrams is better than none (Dickinson and Smith, 2011).

The method detects non-standard parses which may result from parser error or because the text is unusual in some other way, e.g., ungrammatical (Dickinson, 2011). The structures deemed atypical depend upon the corpus used for obtaining the grammar that parser output is compared to.

With a method of scoring the quality of individual dependents in a tree, one can compare the score of a dependent to the score obtaining by hypothesizing a revision. For error detection, this ameliorates the effect of odd structures for which no better parse is available. The revision checking algorithm in Dickinson and Smith (2011) posits new labelings and attachments—maintaining projectivity and acyclicity, to consider only reasonable candidates[8]—and checks whether any have a higher score.[9] If so, the token is flagged as having a better revision and is more likely to be an error.

In other words, the method checks revisions for error detection. With a simple modification of the code,[10] one can also keep track of the best revision for each token and actually change the tree structure. This is precisely what we do. Because the method relies upon very coarse scores, it can suggest too many revisions; in tandem with parser confidence, though, this can filter the set of revisions to a reasonable amount, as discussed next.

## 4.4 Pinpointing erroneous parses

The parse revision methods rely both on being able to detect errors and on being able to correct them. We can assist the methods by using MST confidence scores (Mejer and Crammer, 2012) to pinpoint candidates for revision, and only pass these candidates on to the parse revisers. For example, since APS (anomaly detection) detects atypical structures (section 4.3), some of which may not be errors, it will find many strange parses and revise many positions on its own, though some be questionable revisions. By using a confidence filter, though, we only consider ones flagged below a certain MST confidence score. We follow Mejer and Crammer (2012) and use confidence$\leq$0.5 as our threshold for identifying errors. Non-exhaustive tests on a subset of the test set show good performance with this threshold.

In the experiments reported in section 5, if we use the revision methods to revise everything, we refer to this as the *DeSR* and the *APS* models; if we filter out high confidence cases and restrict revisions to low confidence scoring cases, we refer to this as *DeSR restricted* and *APS restricted*.

Before using the MST confidence scores as part of the revision process, then, we first report on using the scores for error detection at the $\leq$0.5 threshold, as shown in table 1. As we can see, using confidence scores allows us to pinpoint errors with high precision. With a recall around 40–50%, we find errors with upwards of 90% precision, meaning that these cases are in need of revision. Interestingly, the highest error detection precision comes with WSJ as part of the training data and EWT as the testing. This could be related to the great difference between the WSJ and EWT grammatical models and the greater number of unknown words in this experiment, though more investigation is needed. Although data sets are hard to compare, the precision seems to outperform that of more generic (i.e., non-parser-specific) error detection methods (Dickinson and Smith, 2011).

[8] We remove the cyclicity check, in order to be able to detect errors where the head and dependent are flipped.

[9] We actually check whether a new score is greater than or equal to twice the original score, to account for meaningless differences for large values, e.g., 1001 vs. 1000. We do not expect our minor modifications to have a huge impact, though more robust testing is surely required.

[10] `http://cl.indiana.edu/~md7/papers/dickinson-smith11.html`

| Train | Test | Normalization (on test) | Tokens | Attach. Errors | Label. Errors | Total Errors | Precision | Recall |
|---|---|---|---|---|---|---|---|---|
| WSJ | WSJ | none | 4,621 | 2,452 | 1,297 | 3,749 | 0.81 | 0.40 |
| WSJ | EWT | none | 5,855 | 3,621 | 2,169 | 5,790 | 0.99 | 0.38 |
| WSJ | EWT | full | 5,617 | 3,484 | 1,959 | 5,443 | 0.97 | 0.37 |
| EWT | EWT | none | 7,268 | 4,083 | 2,202 | 6,285 | 0.86 | 0.51 |
| EWT | EWT | full | 7,131 | 3,905 | 2,147 | 6,052 | 0.85 | 0.50 |
| WSJ+EWT | EWT | none | 5,622 | 3,338 | 1,849 | 5,187 | 0.92 | 0.40 |
| WSJ+EWT | EWT | full | 5,640 | 3,379 | 1,862 | 5,241 | 0.93 | 0.41 |

Table 1: Error detection results for MST confidence scores ($\leq 0.5$) for different conditions and normalization settings. Number of tokens and errors below the threshold are reported.

## 5 Experiments

We report three major sets of experiments: the first set compares the two parse revision strategies; the second looks into text normalization strategies; and the third set investigates whether the size of the training set or its similarity to the target domain is more important. Since we are interested in parsing in these experiments, we use gold POS tags as input for the parser, in order to exclude any unwanted interaction between POS tagging and parsing.

### 5.1 Parser revision

In this experiment, we are interested in comparing a machine learning method to a simple $n$-gram revision model. For all experiments, we use the original version of the EWT data, without any normalization.

The results of this set of experiments are shown in table 2. The first row reports MST's performance on the standard WSJ data split, giving an idea of an upper bound for these experiments. The second part shows MST's performance on the EWT data, when trained on WSJ or the combination of the WSJ and EWT training sets. Note that there is considerable decrease for both settings in terms of *unlabeled accuracy* (UAS) and *labeled accuracy* (LAS), of approximately 8% when trained on WSJ and 5.5% on WSJ+EWT. This drop in score is consistent with previous work on non-canonical data, e.g., web data (Foster et al., 2011) and learner language (Krivanek and Meurers, 2011). It is difficult to compare these results, due to different training and testing conditions, but MST (without any modifications) reaches results that are in the mid-high range of results reported by Petrov and McDonald (2012, table 4) in their overview of the SANCL shared task using the EWT data: 80.10–87.62% UAS; 71.04%–83.46% LAS.

Next, we look at the performance of the two revisers on the same data sets. Note that since DeSR requires training data for the revision part that is different from the training set of the base parser, we conduct parsing and revision in DeSR with two different data sets. Thus, for the WSJ experiment, we split the WSJ training set into two parts, WSJ02-11 and WSJ12-2, instead of training on the whole WSJ. For the EWT training set, we split this set into two parts and use 25% of it for training the parser (EWTs) and the rest for training the reviser (EWTr). In contrast, APS does not need extra data for training and thus was trained on the same data as the base parser. While this means that the base parser for DeSR has a smaller training set, note that DeSR works best with a weak base parser (Attardi, p.c.).

The results show that DeSR's performance is below MST's on the same data. In other words, adding DeSRs revisions decreases accuracy. APS also shows a deterioration in the results, but the difference is much smaller. Also, training on a combination of WSJ and EWT data increases the performance of both revisers by 2-3% over training solely on WSJ.

Since these results show that the revisions are harmful, we decided to restrict the revisions further by using MST's KD-Fix edge confidence scores, as described in section 4.4. We apply the revisions only if MST's confidence in this dependency is low (i.e., below or equal to 0.5). The results of this experiment are shown in the last section of table 2. We can see

| Method | Parser Train | Reviser Train | Test | UAS | LAS |
|---|---|---|---|---|---|
| MST | WSJ | n/a | WSJ | 89.94 | 87.24 |
| MST | WSJ | n/a | EWT | 81.98 | 78.65 |
| MST | WSJ+EWT | n/a | EWT | 84.50 | 81.61 |
| DeSR | WSJ02-11 | WSJ12-22 | EWT | 80.63 | 77.33 |
| DeSR | WSJ+EWTs | EWTr | EWT | 82.68 | 79.77 |
| APS | WSJ | WSJ | EWT | 81.96 | 78.40 |
| APS | WSJ+EWT | WSJ+EWT | EWT | 84.45 | 81.29 |
| DeSR restricted | WSJ+EWTs | EWTr | EWT | 84.40 | 81.50 |
| APS restricted | WSJ+EWT | WSJ+EWT | EWT | ***84.53*** | *****81.66*** |

Table 2: Results of comparing a machine learning reviser (DeSR) with a tree anomaly model (APS), with base parser MST (* = sig. at the 0.05 level, as compared to row 2).

that both revisers improve over their non-restricted versions. However, while DeSR's results are still below MST's baseline results, APS shows slight improvements over the MST baseline, significant in the LAS. Significance was tested using the CoNLL-X evaluation script in combination with Dan Bikel's Randomized Parsing Evaluation Comparator, which is based on sampling.[11]

For the original experiment, APS changes 1,402 labels and 272 attachments of the MST output. In the restricted version, label changes are reduced to 610, and attachment to 167. In contrast, DeSR changes 1,509 attachments but only 303 in the restricted version. The small numbers, given that we have more than 3,000 sentences in the test set, show that finding reliable revisions is a difficult task. Since both revisers are used more or less off the shelf, there is much room to improve.

Based on these results and other results based on different settings, which, for DeSR, resulted in low accuracy, we decided to concentrate on APS in the following experiments, and more specifically focus on the restricted version of APS to see whether there are significant improvements under different data conditions.

## 5.2 Text normalization

In this set of experiments, we investigate the influence of the text normalization strategies presented in section 3 on parsing and more specifically on our parse revision strategy. Thus, we first apply a *partial normalization*, using only the basic text normalization. For the *full normalization*, we combine the basic text normalization with the spell checker. For these experiments, we use the restricted APS reviser and the EWT treebank for training and testing.

The results are shown in table 3. Note that since we also normalize the training set, MST will also profit from the normalizations. For this reason, we present MST and APS (restricted) results for each type of normalization. The first part of the table shows the results for MST and APS without any normalization; the numbers here are higher than in table 2 because we now train only on EWT—an issue we take up in section 5.3. The second part shows the results for partial normalization. These results show that both approaches profit from the normalization to the same degree: both UAS and LAS increase by approximately 0.25 percent points. When we look at the full normalization, including spell checking, we can see that it does not have a positive effect on MST but that APS's results increase, especially unlabeled accuracy. Note that all APS versions significantly outperform the MST versions but also that both normalized MST versions significantly outperform the non-normalized MST.

## 5.3 WSJ versus domain data

In these experiments, we are interested in which type of training data allows us to reach the highest accuracy in parsing. Is it more useful to use a large, out-of-domain training set (WSJ in our case), a small, in-domain training set, or a combination of both? Our assumption was that the largest data set, consisting of the WSJ and the EWT training sets, would

[11] `http://ilk.uvt.nl/conll/software.html`

| Norm. | Method | UAS | LAS |
|---|---|---|---|
| Train:no; Test:no | MST | 84.87 | 82.21 |
| Train:no; Test:no | APS restr. | ***84.90* | **82.23* |
| Train:part; Test:part | MST | *85.12 | *82.45 |
| Train:part; Test:part | APS restr. | ***85.18* | **82.50* |
| Train:full; Test:full | MST | **85.20 | *82.45 |
| Train:full; Test:full | APS restr. | *****85.24*** | *****82.52*** |

Table 3: Results of comparing different types of text normalization, training and testing on EWT sets. (Significance tested for APS versions as compared to the corresponding MST version and for each MST with the non-normalized MST: * = sig. at the 0.05 level, ** = significance at the 0.01 level).

give the best results. For these experiments, we use the EWT test set and different combinations of text normalization, and the results are shown in table 4.

The first three sections in the table show the results of training on the WSJ and testing on the EWT. The results show that both MST and APS profit from text normalization. Surprisingly, the best results are gained by using the partial normalization; adding the spell checker (for full normalization) is detrimental, because the spell checker introduces additional errors that result in extra, non-standard words in EWT. Such additional variation in words is not present in the original training model of the base parser.

For the experiments with the EWT and the combined WSJ+EWT training sets, spell checking does help, and we report only the results with full normalization since this setting gave us the best results. To our surprise, results with only the EWT as training set surpass those of using the full WSJ+EWT training sets (a UAS of 85.24% and a LAS of 82.52% for EWT vs. a UAS of 82.34% and a LAS of 79.31%). Note, however, that when we reduce the size of the WSJ data such that it matches the size of the EWT data, performance increases to the highest results, a UAS of 86.41% and a LAS of 83.67%. Taken together, these results seem to indicate that quality (i.e., in-domain data) is more important than mere (out-of-domain) quantity, but also that more out-of-domain data can help if it does not overwhelm the in-domain data. It is also obvious that MST per se profits the most from normalization, but that the APS consistently provides small but significant improvements over the MST baseline.

## 6 Summary and Outlook

We examined ways to improve parsing social media and other web data by altering the input data, namely by normalizing such texts, and by revising output parses. We found that normalization improves performance, though spell checking has more of a mixed impact. We also found that a very simple tree reviser based on grammar comparisons performs slightly but significantly better than the baseline, across different experimental conditions, and well outperforms a machine learning model. The results also demonstrated that, more than the size of the training data, the goodness of fit of the data has a great impact on the parser. Perhaps surprisingly, adding the entire WSJ training data to web training data leads to a deteriment in performance, whereas balancing it with web data has the best performance.

There are many ways to take this work in the future. The small, significant improvements from the APS restricted reviser indicate that there is potential for improvement in pursuing such grammar-corrective models for parse revision. The model we use relies on a simplistic notion of revisions, neither checking the resulting well-formedness of the tree nor how one correction influences other corrections. One could also, for example, treat grammars from different domains in different ways to improve scoring and revision. Another possibility would be to apply the parse revisions also to the out-of-domain training data, to make it more similar to the in-domain data.

For text normalization, the module could benefit from a few different improvements. For example, non-contracted words such as *well* to mean *we'll* require a more complicated normalization step, in-

| Train | Test | Normalization | Method | UAS | LAS |
|---|---|---|---|---|---|
| WSJ | EWT | train:no; test:no | MST | 81.98 | 78.65 |
| WSJ | EWT | train:no; test:no | APS | 81.96 | 78.40 |
| WSJ | EWT | train:no; test:no | APS restr | *82.02* | ***78.71* |
| WSJ | EWT | train:no; test:part | MST | 82.31 | 79.27 |
| WSJ | EWT | train:no; test:part | APS restr. | **82.36* | **79.32* |
| WSJ | EWT | train:no; test:full | MST | 82.30 | 79.26 |
| WSJ | EWT | train:no; test:full | APS restr. | *82.34* | **79.31* |
| EWT | EWT | train:full; test:full | MST | 85.20 | 82.45 |
| EWT | EWT | train:full; test:full | APS restr. | ***85.24* | ***82.52* |
| WSJ+EWT | EWT | train:full; test:full | MST | 84.59 | 81.68 |
| WSJ+EWT | EWT | train:full; test:full | APS restr. | ***84.63* | **81.73* |
| Balanced WSJ+EWT | EWT | train:full; test:full | MST | 86.38 | 83.62 |
| Balanced WSJ+EWT | EWT | train:full; test:full | APS restr. | ****86.41*** | *****83.67*** |

Table 4: Results of different training data sets and normalization patterns on parsing the EWT test data. (Significance tested for APS versions as compared to the corresponding MST: * = sig. at the 0.05 level, ** = sig. at the 0.01 level)

volving machine learning or $n$-gram language models. In general, language models could be used for more context-sensitive spelling correction. Given the preponderance of terms on the web, using a named entity recognizer (e.g., Finkel et al., 2005) for preprocessing may also provide benefits.

## Acknowledgments

We would like to thank Giuseppe Attardi for his help in using DeSR; Can Liu, Shoshana Berleant, and the IU CL discussion group for discussion; and the three anonymous reviewers for their helpful comments.

## References

Giuseppe Attardi and Massimiliano Ciaramita. 2007. Tree revision learning for dependency parsing. In *Proceedings of HLT-NAACL-07*, pages 388–395. Rochester, NY.

Giuseppe Attardi and Felice Dell'Orletta. 2009. Reverse revision and linear tree combination for dependency parsing. In *Proceedings of HLT-NAACL-09, Short Papers*, pages 261–264. Boulder, CO.

Ann Bies, Justin Mott, Colin Warner, and Seth Kulick. 2012. English Web Treebank. Linguistic Data Consortium, Philadelphia, PA.

Ozlem Cetinoglu, Anton Bryl, Jennifer Foster, and Josef Van Genabith. 2011. Improving dependency label accuracy using statistical post-editing: A cross-framework study. In *Proceedings of the International Conference on Dependency Linguistics*, pages 300–309. Barcelona, Spain.

Marie-Catherine de Marneffe and Christopher D. Manning. 2008. The Stanford typed dependencies representation. In *COLING 2008 Workshop on Cross-framework and Cross-domain Parser Evaluation*. Manchester, England.

Markus Dickinson. 2010. Detecting errors in automatically-parsed dependency relations. In *Proceedings of ACL-10*. Uppsala, Sweden.

Markus Dickinson. 2011. Detecting ad hoc rules for treebank development. *Linguistic Issues in Language Technology*, 4(3).

Markus Dickinson and Amber Smith. 2011. Detecting dependency parse errors with minimal resources. In *Proceedings of IWPT-11*, pages 241–252. Dublin, Ireland.

Jenny Rose Finkel, Trond Grenager, and Christopher Manning. 2005. Incorporating non-local information into information extraction systems by gibbs sampling. In *Proceedings of ACL'05*, pages 363–370. Ann Arbor, MI.

Jennifer Foster. 2010. "cba to check the spelling": Investigating parser performance on discussion forum posts. In *Proceedings of NAACL-HLT 2010*, pages 381–384. Los Angeles, CA.

Jennifer Foster, Ozlem Cetinoglu, Joachim Wagner, Joseph Le Roux, Joakim Nivre, Deirdre Hogan, and Josef van Genabith. 2011. From news to comment: Resources and benchmarks for parsing the language of web 2.0. In *Proceedings of IJCNLP-11*, pages 893–901. Chiang Mai, Thailand.

Phani Gadde, L. V. Subramaniam, and Tanveer A. Faruquie. 2011. Adapting a WSJ trained part-of-speech tagger to noisy text: Preliminary results. In *Proceedings of Joint Workshop on Multilingual OCR and Analytics for Noisy Unstructured Text Data*. Beijing, China.

Enrique Henestroza Anguiano and Marie Candito. 2011. Parse correction with specialized models for difficult attachment types. In *Proceedings of EMNLP-11*, pages 1222–1233. Edinburgh, UK.

Julia Krivanek and Detmar Meurers. 2011. Comparing rule-based and data-driven dependency parsing of learner language. In *Proceedings of the Int. Conference on Dependency Linguistics (Depling 2011)*, pages 310–317. Barcelona.

Vladimir I. Levenshtein. 1966. Binary codes capable of correcting deletions, insertions, and reversals. *Cybernetics and Control Theory*, 10(8):707–710.

Mitchell Marcus, Beatrice Santorini, and Mary Ann Marcinkiewicz. 1993. Building a large annotated corpus of English: The Penn Treebank. *Computational Linguistics*, 19(2):313–330.

David McClosky, Wanxiang Che, Marta Recasens, Mengqiu Wang, Richard Socher, and Christopher Manning. 2012. Stanford's system for parsing the English web. In *Workshop on the Syntactic Analysis of Non-Canonical Language (SANCL 2012)*. Montreal, Canada.

David McClosky, Mihai Surdeanu, and Christopher Manning. 2011. Event extraction as dependency parsing. In *Proceedings of ACL-HLT-11*, pages 1626–1635. Portland, OR.

Ryan McDonald, Koby Crammer, and Fernando Pereira. 2005. Online large-margin training of dependency parsers. In *Proceedings of ACL-05*, pages 91–98. Ann Arbor, MI.

Ryan McDonald and Fernando Pereira. 2006. Online learning of approximate dependency parsing algorithms. In *Proceedings of EACL-06*. Trento, Italy.

Avihai Mejer and Koby Crammer. 2012. Are you sure? Confidence in prediction of dependency tree edges. In *Proceedings of the NAACL-HTL 2012*, pages 573–576. Montréal, Canada.

Tetsuji Nakagawa, Kentaro Inui, and Sadao Kurohashi. 2010. Dependency tree-based sentiment classification using CRFs with hidden variables. In *Proceedings of NAACL-HLT 2010*, pages 786–794. Los Angeles, CA.

Lilja Øvrelid and Arne Skjærholt. 2012. Lexical categories for improved parsing of web data. In *Proceedings of the 24th International Conference on Computational Linguistics (COLING 2012)*, pages 903–912. Mumbai, India.

Slav Petrov and Ryan McDonald. 2012. Overview of the 2012 shared task on parsing the web. In *Workshop on the Syntactic Analysis of Non-Canonical Language (SANCL 2012)*. Montreal, Canada.