

University of Groningen

## Normalization and parsing algorithms for uncertain input

van der Goot, Rob Matthijs

**IMPORTANT NOTE: You are advised to consult the publisher's version (publisher's PDF) if you wish to cite from it. Please check the document version below.**

*Document Version*

Publisher's PDF, also known as Version of record

*Publication date:*  
2019

[Link to publication in University of Groningen/UMCG research database](#)

*Citation for published version (APA):*

van der Goot, R. M. (2019). *Normalization and parsing algorithms for uncertain input*. [Thesis fully internal (DIV), University of Groningen]. University of Groningen.

**Copyright**

Other than for strictly personal use, it is not permitted to download or to forward/distribute the text or part of it without the consent of the author(s) and/or copyright holder(s), unless the work is under an open content license (like Creative Commons).

The publication may also be distributed here under the terms of Article 25fa of the Dutch Copyright Act, indicated by the "Taverne" license. More information can be found on the University of Groningen website: <https://www.rug.nl/library/open-access/self-archiving-pure/taverne-amendment>.

**Take-down policy**

If you believe that this document breaches copyright please contact us providing details, and we will remove access to the work immediately and investigate your claim.

*Downloaded from the University of Groningen/UMCG research database (Pure): <http://www.rug.nl/research/portal>. For technical reasons the number of authors shown on this cover page is limited to 10 maximum.*

# Normalization and Parsing Algorithms for Uncertain Input

Rob van der Goot

The work presented in this thesis was carried out under the auspices of the Center for Language and Cognition Groningen (CLCG) at the Faculty of Arts of the University of Groningen.



Groningen Dissertations in Linguistics 177

ISSN: 0928-0030

ISBN: 978-94-034-1458-4 (printed version)

ISBN: 978-94-034-1457-7 (electronic version)

© 2019, Rob van der Goot

Cover image: taken by Yiping Duan, in Vancouver, Canada. Edited by Rob van der Goot, with a little help from Lasha Abzianidze.



university of  
 groningen

# Normalization and Parsing Algorithms for Uncertain Input

**PhD thesis**

to obtain the degree of PhD at the  
 University of Groningen  
 on the authority of the  
 Rector Magnificus Prof. dr. E. Sterken  
 and in accordance with  
 the decision by the College of Deans.

This thesis will be defended in public on

Thursday 04 April 2019 at 14:30 hours

by

**Rob Matthijs van der Goot**

born on 11 juli 1991  
 in Heerenveen

**Supervisors**

Prof. G.J.M. van Noord

Dr. M. Nissim

**Assessment committee**

Prof. J. Nivre

Prof. A. van den Bosch

Prof. M.J. Broersma

# Acknowledgements

Thanks Gertjan! It was a fascinating learning experience to work with you. You have provided me with a perfect balance between freedom and professional guidance during the whole process. I enjoyed our weekly meetings, in which you gave constructive criticism, but also ideas towards solutions.

Special thanks go to the Nuance Foundation for funding the project. Furthermore, special thanks to the reading committee, Joakim Nivre, Antal van den Bosch and Marcel Broersma as well as my paranymphs Hessel and Rik.

Thanks to Jennifer Foster and Orphée De Clercq for sharing their datasets, Kevin Humphreys for helping with the Aspell API, and Frank Brokken and Jurjen Bokma for teaching me c++.

During my time in groningen, I enjoyed sharing the office with Valerio, Anna, Dieke, Johannes, Duy, Kilian, Ahmet, Hessel and Rik (not alphabetically ordered, also not random). Thanks all, for enduring my sense of humour. To my other colleagues, Barbara, Malvina, Lasha, Johan, Antonio, Leonie, Gosse, Gregory, Masha, Andreas, Tommaso, Martijn, Simon, John and George. (also not alphabetically ordered). It was a pleasure to work, lunch and have an occasional drink with you.

Furthermore, I enjoyed collaborating and spending time with Nikola Ljubešić and Joachim Daiber.

Finally, many thanks to my family: papa, mama, Lassie and Mawk Mawk, but also Frank, Akke and Bert. I enjoyed spending many weekends with you, bedankt voor de gezelligheid!

And last but not least, thanks to Yiping. The last four years would have been an infinite amount more boring without you!



# Contents

<b>1</b>	<b>Introduction</b>	<b>1</b>
1.1	Contributions . . . . .	3
1.2	Chapter Guide . . . . .	4
1.3	Publications . . . . .	6
1.4	Reproducibility . . . . .	8
<b>I</b>	<b>Background</b>	<b>11</b>
<b>2</b>	<b>Input Uncertainty</b>	<b>13</b>
2.1	Lexical Normalization . . . . .	14
2.1.1	Other Normalization Tasks . . . . .	16
2.2	Data Sets . . . . .	17
2.2.1	Normalization Corpora . . . . .	18
2.2.2	Raw Data . . . . .	21
2.3	A Taxonomy for Normalization Replacements . . . . .	23
2.3.1	Motivation . . . . .	23
2.3.2	Proposed Taxonomy . . . . .	24
2.3.3	Annotation . . . . .	28
2.4	Domain Adaptation . . . . .	30
2.5	Summary . . . . .	32
<b>3</b>	<b>Parsing</b>	<b>33</b>
3.1	Constituency Parsing . . . . .	34
3.1.1	Constituency Trees . . . . .	34
3.1.2	Context-Free Grammar . . . . .	35

---

3.1.3	Probabilistic Context-Free Grammar . . . . .	38
3.1.4	The CYK algorithm . . . . .	38
3.1.5	Latent Annotation . . . . .	41
3.1.6	Error Analysis of an Out-of-the-box Parser on Social Media Data . . . . .	43
3.2	Dependency Parsing . . . . .	45
3.2.1	Dependency Trees . . . . .	45
3.2.2	Transition-based Parsing . . . . .	47
3.2.3	Neural Network Transition-based Parsers . . . . .	50
3.3	Treebanks . . . . .	50
3.3.1	Wall Street Journal (WSJ) . . . . .	50
3.3.2	English Web Treebank (EWT) . . . . .	51
3.3.3	Web2.0 treebank . . . . .	52
3.3.4	MoNoise treebank . . . . .	52
3.4	Summary . . . . .	53
<b>II</b>	<b>Lexical Normalization</b>	<b>55</b>
<b>4</b>	<b>MoNoise: A Modular Approach to Normalization</b>	<b>57</b>
4.1	Automatic Spelling Correction . . . . .	58
4.2	MoNoise: Overview . . . . .	60
4.3	Candidate Generation . . . . .	61
4.3.1	Previous Work . . . . .	62
4.3.2	Modules . . . . .	63
4.4	Candidate Ranking . . . . .	66
4.4.1	Previous Work . . . . .	66
4.4.2	Features . . . . .	66
4.4.3	Classifier . . . . .	69
4.5	Summary . . . . .	70
<b>5</b>	<b>Evaluation Of MoNoise</b>	<b>73</b>
5.1	Evaluation Metrics for Normalization . . . . .	74
5.1.1	Evaluation beyond the word level . . . . .	75
5.1.2	F1 score . . . . .	75
5.1.3	Accuracy . . . . .	77
5.1.4	Error Reduction Rate (ERR) . . . . .	78

---

5.1.5	Area Under the ROC Curve . . . . .	79
5.2	Test Data . . . . .	79
5.2.1	Error Reduction Rate per Corpus . . . . .	80
5.2.2	Comparison with Previous work . . . . .	81
5.3	Type of Errors . . . . .	83
5.3.1	Performance per Category . . . . .	83
5.3.2	Performance per Module . . . . .	85
5.4	Evaluation of Sub-tasks . . . . .	86
5.4.1	Candidate Generation . . . . .	86
5.4.2	Candidate Ranking . . . . .	90
5.4.3	Amount of Training Data . . . . .	93
5.4.4	Separate Error Detection . . . . .	94
5.5	Robustness . . . . .	98
5.6	Conclusion . . . . .	99
<b>6</b>	<b>The Impact of Normalization on POS Tagging</b>	<b>101</b>
6.1	Experimental Setup . . . . .	103
6.1.1	Data . . . . .	104
6.1.2	Bilty . . . . .	104
6.2	The Effect of Normalization on POS Tagging . . . . .	105
6.3	Semi Supervised Settings . . . . .	107
6.4	Combining Normalization and Semi Supervised Learning . .	108
6.4.1	Effect on Known Words Versus Unknown Words . .	109
6.4.2	Performance per POS . . . . .	109
6.5	Evaluation . . . . .	111
6.6	Conclusion . . . . .	112
<b>III</b>	<b>Constituency Parsing</b>	<b>113</b>
<b>7</b>	<b>Integration of Normalization in a PCFG-LA Parser</b>	<b>115</b>
7.1	Related Work . . . . .	117
7.2	Data . . . . .	118
7.3	Method . . . . .	120
7.3.1	Uncertainty in CYK . . . . .	120
7.3.2	Concrete Setup . . . . .	121
7.4	Results . . . . .	122

---

7.5	Analysis . . . . .	125
7.5.1	Effect of Lower Pruning Thresholds . . . . .	125
7.5.2	Performance on Canonical Data . . . . .	126
7.5.3	When is Integrating Normalization Beneficial? . . .	126
7.6	Efficiency . . . . .	130
7.7	Conclusion . . . . .	131
<b>IV</b>	<b>Dependency Parsing</b>	<b>133</b>
<b>8</b>	<b>Integration of Normalization in a Neural Network Parser</b>	<b>135</b>
8.1	Normalization Strategies . . . . .	137
8.1.1	Normalization . . . . .	137
8.1.2	Neural Network Parser . . . . .	137
8.1.3	Integration Strategy . . . . .	139
8.2	Data . . . . .	140
8.3	Evaluation . . . . .	141
8.3.1	Normalization Strategies . . . . .	143
8.3.2	Test Data . . . . .	143
8.3.3	Robustness . . . . .	144
8.3.4	Analysis . . . . .	144
8.4	Conclusion . . . . .	145
<b>V</b>	<b>Conclusion</b>	<b>147</b>
<b>9</b>	<b>Summary and Conclusions</b>	<b>149</b>
	<b>Appendices</b>	<b>152</b>
<b>A</b>	<b>Proof of Equivalence for Error Reduction Rate Formulas</b>	<b>153</b>
<b>B</b>	<b>Relation Between Error Reduction Rate and Distance in ROC Space</b>	<b>155</b>
<b>C</b>	<b>Results of MoNoise On Multiple Normalization Evaluation Metrics</b>	<b>157</b>



---

<b>D Overview of Twitter-specific POS tags</b>	<b>159</b>
<b>E Annotation Guidelines for Universal Dependencies Annotation for Tweets</b>	<b>161</b>
E.1 Tokenization . . . . .	161
E.2 POS tags . . . . .	162
E.3 Unknown Words . . . . .	163
E.4 Emoticons, Emojis, URL's and Phrasal Abbreviations . . .	163
E.5 Domain Specific Tokens . . . . .	163
<b>Bibliography</b>	<b>163</b>
<b>Nederlandse Samenvatting</b>	<b>185</b>
<b>Groningen Dissertations in Linguistics</b>	<b>187</b>



# Chapter 1

## Introduction

*lmao kause I kan  it ain't English klass, its twittr *  
— 2018, lia

With the introduction of the web2.0 and the rise of social media platforms, regular internet users transformed from content consumers to content producers. This led to an interesting new source of information, mainly due to the size, pace and diversity of content found on social media. The spontaneous, informal nature of this new data, led to many new linguistic phenomena, including missing words, shortened words, non-standard capitalization, slang and character repetitions.

These new linguistic phenomena also introduced new challenges for existing natural language processing systems. They face many difficulties processing this spontaneous and hastily produced texts. Consider for example the quote from lia on top of this page. Most English speaking people (especially those familiar with social media) will be able to understand this utterance, despite the high rate of non-standard tokens. However, current natural language processing tools are often designed with standard texts in mind; they break down when they stumble upon such irregularities.

Traditionally, these irregularities were handled by automatic spelling correction. However, these automatic spelling correction models only target unintentional anomalies, whereas in social media texts many intentional anomalies occur. These intentional anomalies include novel words, transformation of existing words, word lengthening and non-standard use of punctuation and capitalization. Our approach to tackle these problems is

---

to transform this non-standard text into its more canonical, or ‘normal’, equivalent. This task is also referred to as normalization. For the quote at the beginning of this chapter the normalization would be:

“lmao because I can 🧑 it ain’t English class, it’s Twitter 🧠”

## Syntactic Parsing of Social Media Texts

In this thesis, we will focus on a fundamental task for natural language processing; syntactic parsing. Syntactic parsing is the process of automatically deriving the syntactic structure of a sentence. Because a syntactic structure is an important step towards the interpretation of a sentence, it is successfully used for many natural language processing (NLP) applications.

Almost all modern parsers are supervised parsers, meaning that they require annotated datasets. They use these datasets to learn linguistic information, which can then be used to derive syntactic structures of new sentences. For decades, parsers have been benchmarked using the Wall Street Journal part of the Penn Treebank (Marcus et al., 1993). On this treebank, accuracies well above 90% have been achieved. However, this treebank consists of well-edited newswire texts. It is unlikely that this performance is transferable to data from non-standard domains, like social media.

An empirical experiment with the Berkeley parser (Petrov and Klein, 2007) on a small social media corpus reveals the severity of this problem. The performance of the Berkeley parser drops from 90% to 68% for the social media domain. Because the parser is trained on news texts, it does not know how to handle the substantially different language occurring on social media.

The most straightforward solution to this problem is to create new treebanks for the social media domain. However, the annotation of high-quality treebanks is very expensive. For the social media domain, even more training data might be necessary compared to the WSJ treebank, because social media texts naturally contain more variety. Furthermore, language on social media is constantly changing, making annotation efforts less valuable over time (Jaidka et al., 2018).

In this thesis, we will explore another solution; normalization. Normalization is the task of converting non-standard language to standard language.

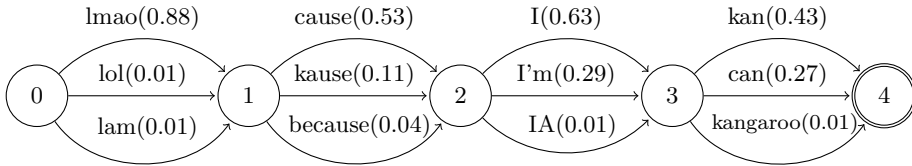


Figure 1.1: Normalization output of “lmao kause I kan”

In this thesis, we will only focus on lexical normalization, which means that we normalize only on the word level. In the remainder of this thesis, we will use the term ‘normalization’ to refer to lexical normalization. Traditionally, normalization is used as pre-processor for natural language processing systems. In this setting, the input is first normalized, and then the output of the normalization is processed instead of the original input. However, this has some disadvantages. Errors made by the normalization model are propagated directly, even when the correct candidate is found, but not ranked as the best candidate. Additionally, the original word is not taken into account. In this thesis, we attempt to overcome these disadvantages by exploiting the top-N candidates of the normalization model.

For a concrete example, see Figure 1.1. In this example, the utterance “lmao kause I kan” would have been normalized to “lmao cause I kan”. By using the top-N candidates, the aforementioned problems can be avoided: errors are not directly propagated, and the correct replacements (‘because’ and ‘can’) are available. A similar approach was theoretically motivated by Levy (2008). In this work, we examine this integration in a more realistic setting.

## 1.1 Contributions

First, we propose MoNoise; a modular approach to normalization. This model is motivated by the idea that normalization consists of different types of replacements (see also Section 2.3). To model these different types of replacements, several modules are developed, each targeting a specific subset of the normalization problem. MoNoise improves upon the existing state-of-the-art for multiple benchmarks.

MoNoise consists of two parts; candidate generation and candidate ranking. For the candidate generation, the most important modules are a classical spelling correction algorithm, word embeddings and a translation dictionary. For the candidate ranking, features are extracted from the modules which were used for the generation, since they often offer some sort of scoring or ranking. On top of these, additional features are added, from which n-grams features are the best predictors. All features are combined in a random forest classifier, which predicts the probability that a candidate belongs to the ‘correct’ class. Accordingly, it is easy to output a list of top-N normalization candidates.

Part II

We experiment with two methods of exploiting the top-N candidates. In Chapter 7, the word graph is used as input for the parsing algorithm. The parser then searches the optimal path through the word-graph with respect to the grammar. As a result, we obtain both a syntactic tree and a syntactically motivated normalization sequence. This approach has similarities to the early work on parsing the output of a speech recognition system (Bates, 1975; Lang, 1989). The output of traditional speech recognition systems was often modeled as a word-graph, the parser then finds the best way through this word graph with respect to the grammar.

Part III

Another way of exploiting the top-N candidates is explored in Chapter 8, where we use a neural network dependency parser. In neural network parsers, words are converted to real-valued vectors, which represent the meaning of the word (this is explained in more detail in Section 3.2.3). In this chapter, vectors from the top-N normalization candidates are merged into one vector, which represents all candidates for a position. Compared to the method explored in Chapter 7, this method does not yield a specific path in the word graph. An advantage of this method is that it does not influence the search space directly since we still have an input of the same length as the original sentence.

Part IV

## 1.2 Chapter Guide

We begin this thesis with an overview of the task of normalization. In Chapter 2, we discuss the scope of the task and quantify which types of phenomena are annotated. Furthermore, we discuss the difference between domain adaptation and normalization.

In Chapter 3, we will give an overview of syntactic parsing. In this thesis, we will focus on two types of parsing: constituency parsing and dependency parsing. For each of these types of parsing, we first introduce the syntactic formalisms, followed by an explanation of how a basic parsing algorithm works. Finally, we describe extensions to these basic algorithms, which are used as a starting point in respectively Chapter 7 and chapter 8.

The normalization model used in the remainder of this thesis (MoNoise) is described in Chapter 4. In this chapter, we start by describing the traditional framework for automatic spelling correction, which is used by MoNoise. Then we give an overview of the model, followed by more detailed descriptions of the two parts of the MoNoise: candidate generation and candidate ranking.

MoNoise is evaluated in detail in Chapter 5. We evaluate on several datasets, containing a variety of languages: English, Dutch, Spanish, Slovenian, Serbian and Croatia. We start the chapter by explaining previously used evaluation metrics and their shortcomings, and introduce a new evaluation metric: error reduction rate. Next, we test the performance of MoNoise using the error reduction rate and compare it to previous work on multiple benchmarks. Furthermore, we examine performance on different types of normalization replacements, examine the performance on candidate generation and ranking separately and test the robustness of the model on data which is more canonical.

Chapter 6, is the first extrinsic evaluation of MoNoise. In this chapter, we test the effect of normalization on a neural network POS tagger for the Twitter domain. We compare the use of normalization to exploiting large amounts of raw texts in a semi-supervised approach.

In Chapter 7, we show that using normalization as pre-processing is also beneficial for constituency parsing. Furthermore, we use the parsing as intersection algorithm (Bar-Hillel et al., 1961) to integrate the top-N candidates from the normalization into the parser.

Recently introduced neural network parsers can exploit character level information, and leverage large amounts of raw text by using pre-trained embeddings. In Chapter 8, we test whether normalization is still useful beyond these novel methods, or if they target the same phenomena. More specifically, we test if normalization is useful for a neural network dependency parser which already makes use character level embeddings and pre-trained

word embeddings. Additionally, we introduce an efficient way to integrate the top-N normalization candidates into neural network parsers.

## 1.3 Publications

Almost all chapters in this thesis are adapted versions of peer-reviewed publications:

Chapter 2:

Rob van der Goot, Rik van Noord, and Gertjan van Noord. A taxonomy for in-depth evaluation of normalization for user generated content. In *Proceedings of the Eleventh International Conference on Language Resources and Evaluation (LREC 2018)*, Miyazaki, Japan, May 2018b. European Language Resources Association (ELRA)

Chapter 4 and 5:

Rob van der Goot and Gertjan van Noord. MoNoise: Modeling noise using a modular normalization system. *Computational Linguistics in the Netherlands Journal*, 7:129–144, December 2017a

Rob van der Goot. Normalizing social media texts by combining word embeddings and edit distances in a random forest regressor. In *Normalisation and Analysis of Social Media Texts (NormSoMe)*, 2016

Chapter 6:

Rob van der Goot, Barbara Plank, and Malvina Nissim. To normalize, or not to normalize: The impact of normalization on part-of-speech tagging. In *Proceedings of the 3rd Workshop on Noisy User-generated Text*, pages 31–39, Copenhagen, Denmark, September 2017. Association for Computational Linguistics

Chapter 7:

Rob van der Goot and Gertjan van Noord. Parser adaptation for social media by integrating normalization. In *Proceedings of the 55th Annual Meeting of the Association for Computational Linguistics (Volume 2: Short Papers)*, pages 491–497, Vancouver, Canada, July 2017b. Association for Computational Linguistics

Chapter 8:

Rob van der Goot and Gertjan van Noord. Modeling input uncertainty in neural network dependency parsing. In *Proceedings of the 2018 Conference on Empirical Methods in Natural Language Processing*, pages 4984–4991, Brussels, Belgium, October 2018. Association for Computational Linguistics

### Other publications

At the beginning of the project, I collaborated with Joachim Daiber to investigate the effect of normalization for dependency parsing:

Joachim Daiber and Rob van der Goot. The Denoised Web Treebank: Evaluating dependency parsing under noisy input conditions. In *Proceedings of the Tenth International Conference on Language Resources and Evaluation (LREC 2016)*, Portoro, Slovenia, May 2016. European Language Resources Association (ELRA)

Another small contribution which did not make it into this thesis was the evaluation of normalization for estimation of semantic relatedness:

Rob van der Goot and Gertjan van Noord. ROB: Using semantic meaning to recognize paraphrases. In *Proceedings of the 9th International Workshop on Semantic Evaluation (SemEval 2015)*, pages 40–44, Denver, Colorado, June 2015. Association for Computational Linguistics

Besides these publications, two other papers on unrelated topics were published during the project:

Malvina Nissim, Lasha Abzianidze, Kilian Evang, Rob van der Goot, Hessel Haagsma, Barbara Plank, and Martijn Wieling. Sharing is caring: The future of shared tasks. *Computational Linguistics*, 43(4):897–904, 2017

Rob van der Goot, Nikola Ljubešić, Ian Matroos, Malvina Nissim, and Barbara Plank. Bleaching text: Abstract features for cross-lingual gender

prediction. In *Proceedings of the 56th Annual Meeting of the Association for Computational Linguistics (Volume 2: Short Papers)*, pages 383–389, Melbourne, Australia, 2018a. Association for Computational Linguistics

## 1.4 Reproducibility

All experiments reported in this thesis can be reproduced by cloning the repository for a specific chapter, and simply execute `./scripts/runAll.sh` from the root of the repository. All results from `runAll.sh` are included in the repositories (in the `preds/` folder). All tables and graphs that are used in the respective chapter can be generated by executing the bash script `./scripts/genAll.sh`. Small differences in the results when re-running can be experienced due to different versions of underlying software or different handling of special characters.

The repositories can be found on the following links:

Chapter 2:

<https://bitbucket.org/robvanderger/normtax>

Chapter 5<sup>1</sup>:

<https://bitbucket.org/robvanderger/chapter5>

Chapter 6:

<https://bitbucket.org/robvanderger/chapter6>

Chapter 7<sup>2</sup>:

<https://bitbucket.org/robvanderger/berkeleygraph/>

Chapter 8:

<https://bitbucket.org/robvanderger/normpar>

---

<sup>1</sup>To rerun the experiments from this chapter on the Dutch data, a copy of the normalization dataset from De Clercq et al. (2014b) is required.

<sup>2</sup>To rerun the experiments from this chapter, a copy of the development and test treebank from Foster et al. (2011a) is required.

So to reproduce all the results reported in this thesis:

```
for REPO in normtax chapter5 chapter6 berkeleygraph normpar;
do
    git clone https://bitbucket.org/robvanderg/$REPO
    cd $REPO
    ./scripts/runAll.sh
    ./scripts/genAll.sh
    cd ..
done
```

However, due to the number of experiments, I strongly suggest to use a parallel setting to run the commands in the `runAll.sh` scripts. I implemented this for the SLURM workload manager, to activate this, simply call `runAll.sh` with the `--slurm` argument: `./scripts/runall.sh --slurm`.



Part I

Background



## Chapter 2

# Input Uncertainty

The rise of social media has led to a valuable new source of information. In contrast to traditional domains used for natural language processing, texts on social media can be written by virtually everyone. This led to a variety of new linguistic phenomena and conventions. Furthermore, the language use on social media is ever-changing, making it much harder to design robust natural language processing models.

To investigate this relatively new type of language, Jones (2010) held a survey among 214 English people aged 18-24, and asked them for the main reasons for unconventional spellings on the internet. The three most commonly chosen reasons were: “it’s become the norm”, “it’s faster” and “people are unsure of the correct spellings”. However, among the seven options, five options were picked by more than half of the respondents. This variety of reasons for alternating from the traditional spelling, also result in a variety of types of alternations.

In this chapter we discuss the problems which are introduced by this new, constantly changing language occurring on social media. In Section 2.1 we discuss the task of lexical normalization in more detail. In Section 2.2, we overview existing normalization datasets. Section 2.3 evaluates which type of replacements are included in the normalization task. Finally, we reflect upon the relation between normalization and domain adaptation.

The taxonomy containing the types of normalization replacements (Section 2.3) is based on:

Rob van der Goot, Rik van Noord, and Gertjan van Noord. A taxonomy for in-depth evaluation of normalization for user generated content. In *Proceedings of the Eleventh International Conference on Language Resources and Evaluation (LREC 2018)*, Miyazaki, Japan, May 2018b. European Language Resources Association (ELRA)

This taxonomy was joint work with Rik van Noord, who helped with refining the category descriptions as well as the annotation.

The annotated data can be found at:

<https://bitbucket.org/robvander/normtax>

## 2.1 Lexical Normalization

There is a variety of methods to tackle the problem sketched in the introduction of this chapter. The most straightforward strategy is to include data from the target domain in the training data. However, manual data annotation is expensive and will not be sufficient over time, since language is constantly changing and new social media platforms will be developed. Hence, automatically annotated data is often exploited: existing models are used to annotate raw data, which is then added to the training data. The difficulty lies in the fact that new information must be added, which at the same time must be annotated correctly. There is ample previous work in this direction in which different strategies of up-training are explored (Foster et al., 2011a; Petrov and McDonald, 2012).

In this thesis, we will explore an orthogonal approach: normalization. Normalization is the task of translating non-standard text to its more canonical, or “normal”, equivalent. In this setup, input from a non-canonical domain is normalized before further processing. This has some advantages over the use of self-training or annotating data for new domains. Firstly, a normalization model can easily be used for multiple tasks, whereas up-training needs to be done for every task. Secondly, if the normalization is robust, it can be applied to multiple domains and data from different timespans. Thirdly, normalization reduces the variance in the data, which has additional advantages. This makes learning models simpler and thus

faster. Besides this, it can also be used to standardize training data.

However, there are also some disadvantages to this approach. Some of the meaning of the original text might be lost after normalizing. For this reason, normalization might result in worse performance for tasks like author profiling or sentiment analysis. Think for example about lowercasing words which were originally typed in capitals; these are important clues for these tasks. However, for syntactically oriented tasks, like the ones explored in this thesis, this is less problematic. To circumvent these issues, a combination of the normalized sentence and the original sentence can be exploited.

Another problem with normalization is that the definition and scope of the task is subjective; not everyone agrees on what the ‘norm’ is. However, most corpora are created using annotation guidelines which describe which phenomena should be normalized. In Section 2.2.1, we look into the annotation process in more detail.

In this thesis, we will make use of published normalization datasets, so the scope of the normalization task is fixed. We further discuss the scope of the normalization task in Section 2.3. We will use the term ‘anomaly’ for words in need of normalization according to the annotators. An example of an annotated sentence is the following:

- (1) mostt social ppl r troublesome  
most social people are troublesome

This example shows that the normalization task includes a variety of types of transformations. The first replacement (‘mostt’ $\mapsto$ ‘most’) is probably a result of an unintentional typing error. The other two replacements are intentional anomalies, and are idiomatic for the domain of social media; for ‘ppl’, all the vowels are removed, and ‘r’ $\mapsto$ ‘are’ is based on pronunciation. We will discuss more examples and differences in annotation in Section 2.2.1.

## Twitter

The social media platform Twitter provides an ideal testbed for the task of normalization. On Twitter, users can share short messages (144 characters, recently extended to 288 characters) called tweets. All of a user’s followers will get a notification of his/her new tweets. Because of this setup, this plat-

form naturally encourages spontaneous, informal texts, which contains more anomalies than other platforms like newspapers, emails or blogs. Another advantage is that there is a huge amount of tweets publicly available, which can be exploited in semi-supervised settings. Because of these properties, it is not a coincidence that most published corpora annotated with normalization contain data from the Twitter domain. There are some conventions which are accommodated by the Twitter platform:

- **Hashtags:** Words starting with the ‘#’ character. Used to indicate the topic or sentiment of the tweet. Often located at the end of the tweet.
- **Mentions:** Words starting with ‘@’, indicating the user this tweet is directed at. Often used at the beginning of the tweet.
- **Retweets:** A repetition of another tweet, indicated by prefixing the original tweet with the token ‘RT’. Retweeting is Usually done because the user agrees with the original tweet or wants to give more attention to a specific tweet. More recently, Twitter started to indicate retweets with a symbol above the tweet, but in this thesis, we will still use the old representation of starting with ‘RT’.

Besides these, many different conventions have been developed by Twitter users. These conventions differ per social demographic group; some of these are discussed in more detail in Section 2.3. Tweets are visualized on the Twitter platform as shown in Figure 2.1. Each tweet is accompanied by the profile picture of the sender. The hashtags and mentions are displayed as hyperlinks, through which respectively more tweets using the same hashtag and the page of the mentioned user can be found. Some statistics about the tweet are shown on the bottom, in this case: 45 replies, 97 retweets, and 161 likes.

### 2.1.1 Other Normalization Tasks

Even within the natural language processing community, the term ‘normalization’ is used for a variety of concepts. There is of course, the normalization of values, but also a variety of tasks which are referred to as normalization. In this section we will shortly review these.

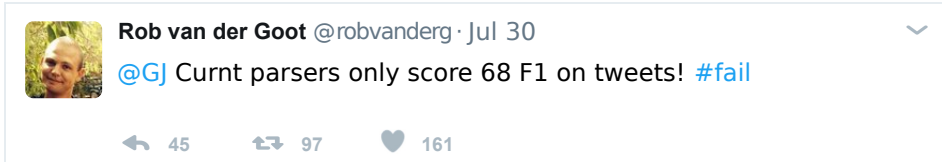


Figure 2.1: An example tweet

The translation of historical texts to modern texts is also often referred to as normalization. For this task a variety of approaches have been evaluated on a variety of datasets, including rule-based, character edit distances (Bollmann, 2012), statistical machine translation (Ljubešić et al., 2016; Pettersson et al., 2013) and neural network methods (Korchagina, 2017; Bollmann et al., 2017). Because of the fragmented nature of the evaluation benchmarks, it is hard to compare these methods.

Even though this task is related to lexical normalization of tweets (it is also often only done on the word-level), initial experiments with our proposed system showed low performance. This is because for the lexical normalization of tweets, the correct replacement of the anomaly occurs in the texts ('ppl' and 'people' are both used on Twitter), which is crucial for some of our features.

Besides this, there is also some previous work which attempts to normalize mentions of time to the same format (Bethard, 2013). Beyond this, there is research to verbalize numbers and other special characters to improve text-to-speech systems (Flint et al., 2017; Gorman and Sproat, 2016).

One other definition of the normalization task, is normalization beyond the word-level. The effects of this normalization for dependency parsing of tweets is theoretically tested by using manually annotated normalization by Baldwin and Li (2015) and Daiber and van der Goot (2016). Furthermore, Aw et al. (2006) test the effect of this normalization for machine translation of social media data.

## 2.2 Data Sets

Because of data availability, we focus on data from the microblog service Twitter. As explained in Section 2.1, this platform contains a lot of sponta-

neous, hastily produced language, which results in a lot of intentional as well as unintentional anomalies. This makes social media data very suitable for testing the performance of a normalization model. Another advantage is that Twitter data is publicly available in large quantities. In addition to annotated corpora, we also exploit raw data in our normalization model. We will first describe the corpora annotated with normalization, and discuss the annotator agreement for these. Secondly, we will give an overview of how the raw data is collected.

### 2.2.1 Normalization Corpora

The annotated normalization corpora used in this thesis are shown in Table 2.1, ordered by size. This order is used throughout this thesis. If the corpus does not include a standard development or test split, we use the first 60% of the sentences for training, the following 20% for development and the last 20% for testing. For English, there are multiple annotated corpora. LiLiu is used as training data for LexNorm1.2 because of their similar annotation style and the small size of LexNorm1.2. LexNorm2015 has a different annotation style, and is thus used as a separate benchmark.

There is some difference in the percentage of tokens that are normalized, which can be attributed to differences in the collection of the tweets, or annotation. Because the goal of the datasets is to test normalization models, the tweets are usually collected using a selection procedure. This is often done by only selecting tweets which containing a minimum amount of out of vocabulary words. This ensures a certain level of non-standardness. While this results in a biased dataset, it makes annotation ‘denser’ and thus speeds up the annotation process.

The ‘1-N’ column in Table 2.1 indicates whether normalization beyond the word-level is considered. None of these corpora include annotation for word insertion, deletion or re-ordering; annotation beyond the word-level is restricted to splitting (1-N) and merging (N-1). Capitalization is kept in most corpora, but it should be noted that it is not corrected in any of these datasets. It is transferred from the original word (‘NICEE’ $\mapsto$ ‘NICE’) or annotated inconsistently. Therefore, we convert all data to lowercase in all our normalization experiments. However, for the experiments where normalization is used to improve POS tagging (Chapter 6) or parsing (Chapter 7 and 8), we exploit a case-sensitive model because capitalization can be

Corpus Source	Words	Lang.	%normed	1-N	Caps
GhentNorm De Clercq et al. (2014b)	12,901	NL	4.8	+	+
TweetNorm Alegria et al. (2013)	13,542	ES	6.3	+	+
LexNorm1.2 Yang and Eisenstein (2013)	10,576	EN	11.6	-	-
LiLiu Li and Liu (2014)	40,560	EN	10.5	-	+
LexNorm2015 Baldwin et al. (2015a)	73,806	EN	9.1	+	-
Janes-Norm Erjavec et al. (2017)	75,276	SL	15.0	-	+
ReLDI-hr Ljubešić et al. (2017a)	89,052	HR	9.0	-	+
ReLDI-sr Ljubešić et al. (2017b)	91,738	SR	8.0	-	+

Table 2.1: Comparison of the normalization corpora used in this thesis. %normed indicates the percentage of words which are normalized. The ‘1-N’ column indicates whether words are split/merged in the annotation, the ‘caps’ column indicates whether capitalization was transferred to the normalization (it is not corrected).

informative for syntax. For these experiments, we will use a normalization model trained on the LiLiu corpus, in which the capitalization from the original word is kept.

To give a better idea of the nature of the data and annotation, we will discuss some example sentences below.

- (2) lol or it could b sumthn else ...  
 lol or it could be something else ...

Example 2 is taken from the LiLiu corpus, this example contains two

replacements. The replacements are subsequent words, which might lead to problems for using context directly. The replacement ‘b’ $\mapsto$ ‘be’ can be solved by adding only one character, whereas the replacement of ‘sumthn’ $\mapsto$ ‘something’ is more distant. These more distant replacements are problematic for traditional spelling correction algorithms since these are focused on smaller repairs.

- (3) i aint messin with no1s wifey yo lol  
 i ain’t messing with no one’s wifey you laughing out loud

Example 3 originates from the LexNorm2015 corpus. This annotation also includes 1-N replacements; ‘no1s’ and ‘lol’ are expanded. the word ‘no1s’ is not only split, but also contains a substitution of a number to its written form. In contrast to the previous example, here the token ‘lol’ is expanded; this is a matter of differences in annotation guidelines. The annotator decided to leave the word ‘wifey’ as is, whereas it could have been normalized to wife, this reflects the fact that the annotation guidelines prefer conservativity (Baldwin et al., 2015b). In other words, if the annotator is unsure about an annotation, the original word should be kept. On the other hand, the annotator decided to normalize ‘yo’ to ‘you’, even though this could also be considered as an interjection.

- (4) nee ! :-D kzal no es vriendelijk doen lol  
 nee ! :-D ik zal nog eens vriendelijk doen laughing out loud  
 no ! :-D I shall more once friendly do laughing out loud

Example 4 is taken from the GhentNorm corpus. The word ‘ik’ (EN: I) is often abbreviated and merged with a verb in Dutch tweets, leading to ‘kzal’ which is split in the annotation to ‘ik zal’ (EN: I shall). ‘no’ is probably a typographical mistake, whereas ‘es’ is a shortening based on pronunciation. Similar to the LexNorm2015 annotation, the phrasal abbreviation ‘lol’ is expanded.

### **Annotator Agreement**

The normalization task can be seen as a rather subjective task; the annotators are asked to convert noisy texts to ‘normal’ language. The annotation guidelines are usually quite short (De Clercq et al., 2014a; Baldwin et al.,

2015b), leaving space for interpretation; which might lead to inconsistent annotation. In this section we will compare agreement among annotators for corpora that were annotated by multiple annotators. This will reveal how subjective the task is, and give us an idea of a theoretical upper bound performance.

Annotator agreement is usually evaluated using Cohen’s kappa (Cohen, 1968) or Fleiss’ kappa (Fleiss, 1971). These measures do not simply calculate the agreement as a percentage, but also take chance into account. Cohen’s kappa is used for agreements between two annotators, whereas Fleiss’ kappa gives scores in the same range for more than two annotators. In general, kappa scores above 0.60 are considered to indicate substantially high agreement, and scores higher than 0.80 indicate near perfect agreement.

To the best of our knowledge, kappa scores have only been published for two datasets. Pennell and Liu (2014) report a Fleiss’ kappa of 0.891 on the detection of words in need of normalization, whereas Baldwin et al. (2015a) report a Cohen’s kappa of 0.5854. The first kappa indicates a near perfect agreement, whereas the second indicates a high agreement. Differences in the kappa score can be due to multiple reasons, e.g. differences in annotators, guidelines or data.

Pennell and Liu (2014) also shared the annotation efforts of each annotator for the candidate selection; we used this data to calculate the pairwise human performance on the choice of the correct normalization candidate. This revealed that the annotators agree on the choice of the normalized word in 98.73% of the cases. Note that this percentage is calculated assuming gold error detection. In conclusion, we can say that despite differences in datasets, the inter-annotator agreements indicate a high till near-perfect agreement for the decision whether to normalize. Furthermore, on the choice of the correct normalization, annotators usually agree.

### 2.2.2 Raw Data

Raw texts can be exploited as an extra source of information in a semi-supervised setup. This data can usually be obtained quite easily in huge amounts. We collected two separate datasets for each language: one containing canonical texts and one containing user-generated content. As a source

Language	ISO	Words
Dutch	NL	226,278,545
Spanish	ES	492,538,560
English	EN	1,950,682,547
Slovenian	SL	29,322,403
Croatian	HR	37,212,539
Serbian	SR	56,999,554

Table 2.2: The number of words in our Wikipedia datasets.

for canonical data, we used Wikipedia dumps from 01-01-2018<sup>1</sup>. These dumps are cleaned using the WikiExtractor<sup>2</sup>. The sizes for the different languages in number of words are shown in Figure 2.2.

For the user-generated texts, we used existing datasets and in-house collections based on availability, these are summarized in Table 2.3. The tweets are collected through the Twitter API. To get the tweets for our specific languages, word-lists with words common in that language are used. For Dutch, we used the method from Tjong Kim Sang and van den Bosch (2013), these tweets are collected between 2010 and 2016. For English, we used the 100 most frequent words of the Oxford English Corpus<sup>3</sup> and collected tweets during 2016. Note that this list is a lot less tuned, and contains a lot of stop words, but since most tweets are written in English this should be sufficient. For Spanish, we use a selection of frequent words from Stopwords ISO<sup>4</sup> and collected tweets during a part of 2010 and 2017. For the South Slavic languages, we used the existing Web-as-Corpus (WaC) datasets (Ljubešić and Klubička, 2014), because there are fewer tweets for these languages and it is more difficult to filter tweets for languages we are not familiar with.

We did not perform any tokenization (ie. we use whitespace as delimiter) on these corpora nor on the Wikipedia corpora because the variety in the data sets and languages, tokenization is a non-trivial problem and

<sup>1</sup><https://dumps.wikimedia.org/backup-index.html>

<sup>2</sup><https://github.com/attardi/wikiextractor>

<sup>3</sup>[https://en.wikipedia.org/wiki/Most\\_common\\_words\\_in\\_English](https://en.wikipedia.org/wiki/Most_common_words_in_English)

<sup>4</sup><https://github.com/stopwords-iso/stopwords-es>

Language	ISO	Source	Words	Sentences/tweets
Dutch	NL	Twitter	17,068,906,534	1,545,871,819
Spanish	ES	Twitter	1,567,148,804	108,319,387
English	EN	Twitter	9,956,184,920	760,744,676
Slovenian	SL	slWaC	1,259,553,862	73,661,424
Croatian	HR	hrWaC	2,028,739,765	98,431,007
Serbian	SR	srWaC	891,060,438	7,080,671

Table 2.3: Some basic statistics for the non-standard datasets.

consistently wrong tokenization might harm the coverage. The only pre-processing is the replacement of URLs by the token ‘<URL>’ and Twitter usernames by ‘<USERNAME>’. This is done to keep vocabulary sizes smaller and speed up the processing of this data. After replacing these tokens, we delete all duplicate sentences. For the Twitter data, this resulted in removing approximately half of the data, mainly because of retweets. Table 2.3 shows the size of the resulting corpora in number of words. Since we did not segment the tweets, we give the number of tweets instead of sentences for the Twitter corpora.

## 2.3 A Taxonomy for Normalization Replacements

In this section we will investigate the different types of anomalies that are normalized in normalization corpora. To this end, we introduce a novel taxonomy of categories of replacements used in normalization corpora. This taxonomy can be used to clarify which problems are most prevailing in the normalization task, and can also be used to evaluate a normalization model in more detail.

### 2.3.1 Motivation

For other natural language processing tasks concerning the conversion of text to another format, such as grammatical error correction and machine translation, there already exist detailed error taxonomies, which help in evaluating the strengths and weaknesses of systems (Mariana, 2014; Ng

et al., 2014). For lexical normalization, such an evaluation does not exist yet. Most previous work uses accuracy or F1 score for evaluation. To gain more insights, Reynaert (2008) proposed an evaluation framework which evaluates the different sub-tasks in more detail; enabling the evaluation of error detection, candidate generation, and candidate ranking. Orthogonal to this approach, we propose a more in-depth evaluation of normalization, focusing on categories of different normalization replacements.

Existing error taxonomies are unfortunately not suitable for the task of normalization since the categories are substantially different. For machine translation, taxonomies as the Multidimensional Quality Metrics (Mariana, 2014) are proposed, which contains 3 main categories: accuracy, verity and fluency. Because in machine translation, meaning can more easily be lost, there are many categories focusing on the semantics (accuracy and verity), for the normalization task these are less relevant. For grammatical error correction, often a very detailed taxonomy for errors is used; the default benchmark has 28 categories (Ng et al., 2014). However, many of the errors in this taxonomy are not annotated in the normalization benchmarks, while at the same time the normalization corpora also have replacements which are not included in this benchmark.

Different benchmarks for normalization specify the task slightly differently; a striking example is the inclusion of the expansions of phrasal abbreviations like” ‘lol’ $\mapsto$ ‘laughing out loud’. From a syntactic perspective, this is not the desired output; ‘lol’ is often used as interjection. This reveals another potential use for a taxonomy of normalization actions: it enables us to filter the categories before training, and thus learn a model which only handles the desired categories.

### 2.3.2 Proposed Taxonomy

Our proposed taxonomy is loosely based on the categories used by the Forebank (Kaljahi et al., 2015) and Baldwin and Li (2015). On top of these, we took categories from the annotation guidelines of LexNorm2015 (Baldwin et al., 2015b) since they include which kind of anomalies should be annotated. The categories of our taxonomy are a combination of the categories used in previous work, and are empirically refined during the early stages of annotation. We make a main distinction between intentional and unintentional anomalies since they have a different origin; our hypothesis

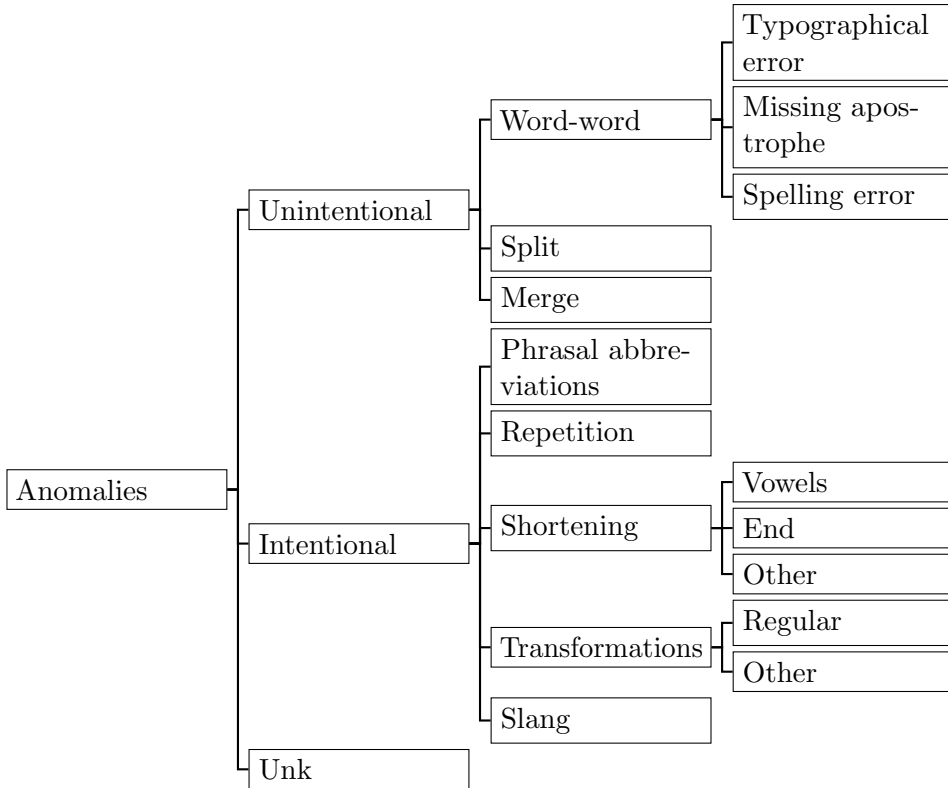


Figure 2.2: Our proposed taxonomy of anomalies in user-generated text.

is that they also might require different handling in NLP systems. Our proposed taxonomy is shown in Figure 2.2; accompanying examples can be found in Table 2.4. We will now describe each final category in more detail.

**1. Typographical error** This includes small errors, which are a result of mistyping keys on keyboards. In case of doubt with another category, we put words with a character edit distance of one in this category (e.g. ‘bidge’ $\mapsto$ ‘bridge’, ‘feela’ $\mapsto$ ‘feels’).

**2. Missing apostrophe** In social media text, the apostrophe is often skipped. Even though this category is relatively trivial to solve, it might

Cat.	Examples
1.	spirite $\mapsto$ spirit, complaining $\mapsto$ complaining, throwg $\mapsto$ throw
2.	im $\mapsto$ i'm, yall $\mapsto$ y'all, microsofts $\mapsto$ microsoft's
3.	dieing $\mapsto$ dying, themselves $\mapsto$ themselves, favourite $\mapsto$ favorite
4.	pre order $\mapsto$ preorder, screen shot $\mapsto$ screenshot
5.	alot $\mapsto$ a lot, nomore $\mapsto$ no more, appstore $\mapsto$ app store
6.	lol $\mapsto$ laughing out loud, pmsl $\mapsto$ pissing myself laughing
7.	soooo $\mapsto$ so, weiiiiird $\mapsto$ weird
8.	pls $\mapsto$ please, wrked $\mapsto$ worked, rmx $\mapsto$ remix
9.	gon $\mapsto$ gonna, congrats $\mapsto$ congratulations, g $\mapsto$ girl
10.	cause $\mapsto$ because, smth $\mapsto$ something, tl $\mapsto$ timeline,
11.	foolin $\mapsto$ fooling, wateva $\mapsto$ whatever, droppin $\mapsto$ dropping
12.	hackd $\mapsto$ hacked, gentille $\mapsto$ gentle, rizky $\mapsto$ risky
13.	cuzi $\mapsto$ because, fina $\mapsto$ going to, plz $\mapsto$ please
14.	skept $\mapsto$ sunglasses, putos $\mapsto$ photos

Table 2.4: Examples of normalization pairs for each category.

have large effects in a pipeline approach, since it can resolve tokenization issues.

**3. Spelling error** This category includes all cases in which a word is unintentionally used in the wrong form, including spelling and grammatical errors. We also include mismatches between American English and British English here. When in doubt between this category and the first category, annotators should answer the following question: if the sender were to send the message again, would he/she make the same mistake?

**4. Split** When a word is split into multiple words. There is one case in our corpus where this happens intentionally ('l o v e' $\mapsto$ 'love'), this is still annotated in this category.

**5. Merge** There is no space between two subsequent words, this is a special case of a typographical error.

**6. Phrasal abbreviation** In some datasets, phrasal abbreviations, such as ‘lol’, ‘idk’ and ‘brb’ are expanded to respectively “laughing out loud”, “I don’t know” and “be right back”. These abbreviations consist of all first characters of the words they represent.

**7. Repetition** On social media, extra focus is put on words by character repetition. Repetition can also occur on sequences of characters, e.g. ‘haha-hahahaha’. Even when adding only one extra character, we categorize the replacement here.

**8. Shortening vowels** A common way to shorten words is to leave out vowels. In this category, we also place words in which most, but not all, of the vowels are removed (‘pple’ $\mapsto$ ‘people’).

**9. Shortening end** Another way to shorten words is too leave out the last character(s) or syllable(s). Based on context, it is often trivial for humans to understand which word is intended. If the anomaly includes a suffix to indicate plurality, we still classify it in this category (‘favs’ $\mapsto$ ‘favorites’).

**10. Shortening other** There are other variations to shorten words. For example, using only the first letter of each part of a compound, skipping another syllable then the last or using standard abbreviations (‘pdx’ $\mapsto$ ‘portland’). This category also contains combinations of the previous two categories (‘talkn’ $\mapsto$ ‘talking’, ‘smth’ $\mapsto$ ‘something’).

**11. Regular transformation** For this category, we consider common transformations of endings of words. On Twitter, it is common to end participles and gerunds with ‘in’ instead of ‘ing’. Another common transformation is to replace the last syllable with ‘a’. Transformations like ‘cuz’ $\mapsto$ ‘because’ do not fit in this category, because this transformation is not transferable to other words.

**12. Other transformation** Other transformations include replacements with similar sounding characters or syllables. Similar sounding characters include for example ‘u’ $\mapsto$ ‘you’, ‘s’ $\mapsto$ ‘z’, ‘d’ $\mapsto$ ‘t’. Sometimes even similar looking characters are used (‘3Volution’ $\mapsto$ ‘evolution’).

**13. Slang** This category includes all novel words specific to this domain. These can be derived from a combination of the previous categories, but are now considered standard vocabulary for this domain.

**14. Unk** Annotator is not sure in which category a word belongs. This can be because the annotator does not agree with the normalization annotation, or because the tweet is not understandable for the annotator.

It should be noted that these categories only include phenomena which are annotated in most of the datasets used in this thesis. For example, capitalization corrections are not included, since they are usually not consistently annotated. Furthermore, this taxonomy does not include word insertion, deletion, and reordering. The only categories which go beyond word to word replacements are SPLITTING, MERGING and PHRASAL ABBREVIATIONS.

### 2.3.3 Annotation

To test our proposed taxonomy, we annotated the training part of the LexNorm2015 corpus (Baldwin et al., 2015a) with an extra layer, which indicates for each normalization replacement to which category in our taxonomy it belongs. We choose to annotate this dataset because of its size, it is publicly available, the most recent and annotation is verified by shared task participants. Furthermore, a variety of approaches has already been tested on this benchmark. It should be noted, however, that as long as alignment is available, the taxonomy can easily be adapted for other corpora.

To ease the annotation effort, we annotate unique normalization replacement pairs. Since ambiguity problems should be resolved by the normalization layer, this is a safe generalization. In case of doubt, annotators still have access to the contexts. Sometimes a replacement fits in two categories, for example: ‘difffff’ $\mapsto$ ‘different’ fits in category 7 and 9. In these cases, it is up to the annotator to decide which category defines the replacement most.

One annotator annotated all the 1,204 replacement pairs present in the training part of the LexNorm2015 dataset. Additionally, a second annotator annotated a random shuffle of 150 replacements to test the inter-annotator agreement. Both annotators are guided by the descriptions in Section 2.3.2

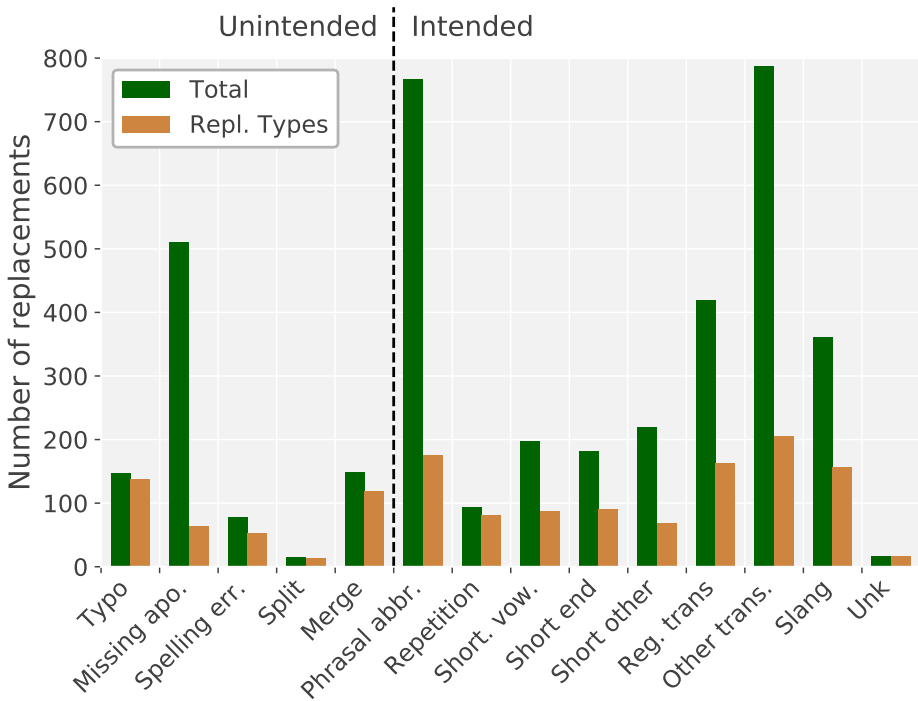


Figure 2.3: The distribution of the different categories. We show the total number of replacements as well as the unique replacements.

The annotators reached a Cohen’s Kappa (Cohen, 1960) of 0.807 on the replacement types, which indicates a near perfect agreement. There was no clear trend in the disagreements; the most common disagreement was between category `SHORTENING VOWELS` and `SLANG`, but this only occurred three times. After annotation, both annotators discussed and resolved the differently annotated pairs and refined the description of the categories.

Figure 2.3 shows the distribution of the categories. We distinguish between the total number of replacements pairs in each category and the unique replacement pairs in each category, which we refer to as ‘replacement types’. The number of replacement types is rather evenly distributed. On the other hand, some categories have a much higher total frequency, this is mostly due to a couple of very frequent replacements, like ‘u’→‘you’ (`OTHER`

TRANSFORMATION) and ‘lol’ $\rightarrow$ ‘laughing out loud’ (PHRASAL ABBREVIATION). Most of the replacements are intentional word-word replacements; about half of these are OTHER TRANSFORMATIONS. Other large categories are PHRASAL ABBREVIATIONS and MISSING APOSTROPHE. Categories with a high number of total occurrences and a relatively low number of replacements types should be relatively easy to solve since they can be learned directly from the training data.

## 2.4 Domain Adaptation

Most natural language processing systems are designed with standard data in mind. If these systems are used on data from another domain, they often suffer a performance drop, because they simply lack the knowledge about the structures and phenomena for the other domain. This is also known as the problem of domain shift. The task of adapting a natural language system to a domain different than what it is trained on is called domain adaptation. The severity of the domain shift depends on how distant the different domains are. One could easily imagine that when training on newswire data, performance on Wikipedia articles will be higher compared to spoken conversations.

To properly define the task of domain adaptation, it is crucial to answer the question: What is a domain? Unfortunately, there is no clear agreement on the notion of a domain. For an overview of the different terminology and definitions of different types and styles of text, we refer to Lee (2002). In natural language processing, corpora are often collected from different platforms, which are then considered to be domains. However, this is a simplification. Consider for example data from telephone conversations or private messaging applications. The language use on these platforms can vary greatly on these platforms, depending on who is speaking to whom, with which goal. For domain adaptation of natural language processing, it might be more realistic to classify domains as a variety in a high-dimensional variety space (Plank, 2016). However, this complicates testing, since this results in a virtually unlimited number of domains. In this work, we comply with the original approach, and broadly divide our datasets in a binary manner: canonical data and non-canonical data, where newswire texts and Wikipedia texts fit in the former category, and social media data in the

latter category. Actually, social media data also contains some news articles, which might be very similar to the newswire texts. However, most of the published datasets are created using a filtering strategy to exclude these.

The task of domain adaptation is related to the normalization task described in Section 2.1, however, it is not the same. Normalization has the potential to solve a part of the domain adaptation problem, but it skips over many problems inherent to domain adaptation. At the same time, it does more than just domain adaptation: since the goal of normalization is to convert language to a standard, it reduces the number of phenomena that have to be handled. This can be beneficial for efficiency since vocabularies will be smaller. Additionally, normalization can even be useful when not switching domains. Consider a situation where training and test data is available only from a very non-standard domain; normalization can be used to standardize both train and test data.

The normal use of normalization is to convert the test data to be more similar to the training data. The opposite direction is also explored in previous work: converting the training data to be more like the test data. In early work, this was done to improve error detection (Foster and Andersen, 2009) and error correction (Felice and Yuan, 2014) of learner data. In this previous work, artificial training data is generated by rule-based methods which insert errors into canonical English.

More recently, similar approaches have been used for syntactic parsing. Sakaguchi et al. (2017) insert different amounts of errors in their training and test data, and show that performance is improved for both parsing and grammatical error correction when using the training data with a similar percentage of errors as the test data. Blodgett et al. (2018) generate synthetic training data, in which they insert social media specific structures as well as African American English structures. To test the effect of this new training data, they create two small test treebanks; one containing mainstream English tweets and one containing African American English tweets. Their results show consistent improvements when training on the synthetic data, even when exploiting a domain-specific POS tagger and external embeddings.

## 2.5 Summary

Existing natural language processing systems are often developed with canonical texts in mind. Hence, their performance drops dramatically when switching to a non-standard domain like social media. One way to narrow this performance gap is to normalize the data before processing it. Normalization is the task of translating non-standard texts to its more canonical equivalent.

The main advantage of using normalization to adapt natural language processing systems is that is broadly applicable: one normalization system can be used to adapt multiple systems, and to adapt to data from a different domain or time-span, only the normalization has to be updated. Furthermore, normalizing leads to less variety in the data and smaller vocabulary sizes, which might speed up the processing. The main disadvantage of normalizing, some of the meaning of a sentence is lost, however for syntactically oriented task this disadvantage is less relevant.

In this thesis, we will evaluate for the normalization task on 7 benchmarks, in a variety of languages: English, Dutch, Spanish, Slovenian, Croatian and Serbian. Previously published inter annotator agreements for normalization datasets have shown that annotators have a high till near-perfect agreement for the choice on whether or not to normalize a word. For the choice of the correct normalization replacement, annotators almost always agree.

To gain a deeper insight into the problem of normalization, we investigated the different types of replacements occurring in an English dataset. This revealed which categories are most frequent: MISSING APOSTROPHE, PHRASAL ABBREVIATIONS, and OTHER TRANSFORMATIONS. The distribution based on unique replacements has a much flatter distribution, showing that the main differences in size are due to a few very frequent replacements.

# Chapter 3

## Parsing

In this thesis we will examine the effect of normalization for two different types of parsers: constituency parsers (Chapter 7) and dependency parsers (Chapter 8). In this chapter we will explain the corresponding syntactic formalisms: context-free grammars and dependency representations. These two syntactic formalisms model the syntactic structure of natural language differently. In constituency trees, words are grouped into constituents, whereas in dependency structures, words are connected to each other directly. These different structures require different parsing algorithms.

This chapter is divided in a constituency parsing section and a dependency parsing section. For both of the syntactic formalisms, we will first explain what the syntactic structures look like. Then we will look into more detail in the parsing process; how can a parser learn about such structures from annotated data, and exploit this knowledge to parse new sentences. We start out with rudimentary algorithms, followed by extensions which reach a superior performance and will be the starting point in respectively Chapter 7 and Chapter 8. For constituency parsing, we will focus on latent annotation; a technique to learn more specific syntactic categories. For dependency parsing, we will shortly explain how the basic algorithm can be adapted to be used in a neural network. After discussing both parsing formalisms, we will describe the treebanks that are used in this thesis.

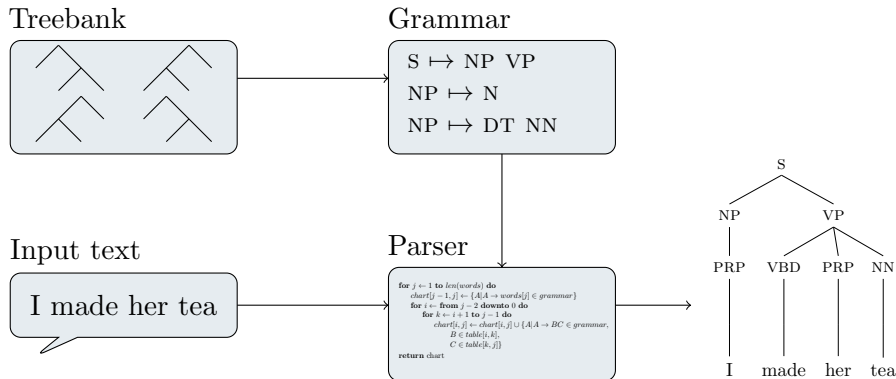


Figure 3.1: Schematic overview of training and using a parser

## 3.1 Constituency Parsing

Starting with the constituency format, we will first explain what a constituency tree looks like. Then, we will discuss how a grammar can be learned from a dataset of annotated trees, also called a treebank. Next, we show how such a grammar can be used in the CYK algorithm to derive parse trees for new input. The whole process of learning a grammar, and running a parser is schematically shown in Figure 3.1. After explaining the basics, we will shortly discuss an extension called latent annotation, which results in more powerful grammars. Finally, we will show some examples which demonstrate that current parsers are inadequate for the parsing of social media data.

### 3.1.1 Constituency Trees

In a constituency tree, a sentence is recursively decomposed into smaller segments, called constituents. These constituents are classified into categories. The terminal nodes are the words of the sentence, which are usually first assigned a word level label, also called a part-of-speech (POS) tag.

An example constituency tree for the sentence “I made her tea” is shown in Figure 3.2. This simple sentence is a declarative clause, indicated with an s. The left part of the tree is a noun phrase (NP), containing only the personal pronoun (PRP) ‘I’. The rest of the sentence is a verb phrase (VP).

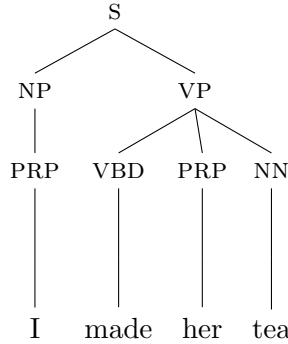


Figure 3.2: I made her tea

However, this sentence can be interpreted in at least two different ways, which leads to different constituency trees. Having multiple syntactic trees for one input is also called ambiguity. The tree in Figure 3.2 gives rise to the meaning that ‘I’ prepared a tea for ‘her’.

The tree for the alternative meaning is shown in Figure 3.3. The meaning of this derivation is that ‘I’ made tea which belongs to ‘her’. Syntactically, the difference is that ‘her’ and ‘tea’ form a separate noun phrase (NP). Furthermore, ‘her’ is tagged as possessive pronoun (PRP-S), since it now indicates the possession of the tea.

This example illustrates one of the main problems syntactic parsers face. Most difficulties do not arise in finding *a* syntactic tree, but in finding the *correct* syntactic tree. This is referred to as the problem of disambiguation. This is not only a problem for automatic parsers, even humans do not always agree on the choice of the correct tree. Human agreement on annotation of syntactic trees is around 90-95% (Berzak et al., 2016), indicating that for 5-10% of the constituents they disagree.

### 3.1.2 Context-Free Grammar

A grammar provides linguistic information for the parser. Grammars can be derived from a treebank automatically or can be manually constructed by humans. Almost all modern parsers use automatically derived grammars. In this section, we will discuss context-free grammars (CFG) (Chomsky, 1956), and in the next section we will discuss a probabilistic variant of a

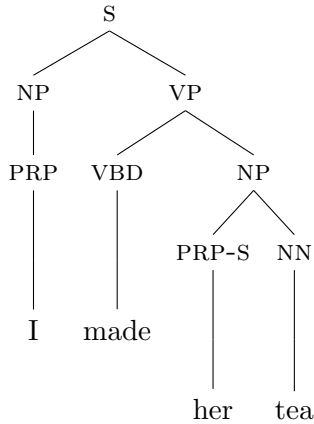


Figure 3.3: I made her tea (alternative interpretation)

CFG.

Formally, a context-free grammar ( $G$ ) is a quadruple:

- $N$ : finite set of non-terminals
- $T$ : finite set of terminals (disjoint from  $N$ :  $N \cap T = \emptyset$ )
- $R$ : finite set of production rules, which can be considered tuples containing  $(\alpha, \beta)$ , indicating that  $\alpha$  can be rewritten to  $\beta$ . Here,  $\alpha \in N$  and  $\beta$  is a sequence of terminals and non-terminals:  $(T \cup N)^*$ .
- $s$ : the start symbol, a special non-terminal:  $s \in N$

A context-free grammar  $G$  describes a language  $L(G)$  consisting of all sequences of terminals which can be generated by the production rules in  $R$ . Starting with the special non-terminal  $s$ , any rule from  $R$  where  $s$  is rewritten can be used:  $s \Rightarrow x$ , where  $(s, x) \in R$ . From here, any non-terminal in  $x$  is iteratively rewritten using rules from  $R$  until only terminals are left. This recursive rewriting process is also called the reflexive transitive closure. The reflexive transitive closure is denoted as:  $S \xRightarrow{*} y$  and yields all strings of language  $L(G)$ .

For the parsing of natural language, POS tags are commonly considered as terminal tags instead of words. In practice, both POS tags and words

1.  $S \rightarrow NP VP$
2.  $NP \rightarrow PRP$
3.  $VP \rightarrow VBD NP$
4.  $NP \rightarrow PRP-S NN$

Table 3.1: An example context-free grammar derived from the tree in Figure 3.3

can be used, since they have a 1-1 relation. If words are used as terminal nodes,  $R$  contains rules like  $NN \rightarrow \text{house}$ . Most existing parsers require POS tags as input or contain an internal POS tagger. This simplifies the parsing algorithm as POS tags are a closed set. In this chapter, we will comply with this common approach and consider POS tags as terminal nodes.

It should be noted that the right-hand side of a rule can also be empty. This is indicated by rewriting the left-hand side of a rule into an epsilon ( $\epsilon$ ). This also occurs in treebanks, for example to denote ellipsis. However, it is common practice to remove epsilons when training a parser and ignore them during evaluation. We will comply with this common practice in this thesis.

A basic CFG can simply be read from a treebank; for every expansion of a node, we can simply use the parent node as left-hand side of the rewrite rule, and its child-nodes as right-hand side. This is also called a treebank grammar. The rules that can be derived from the tree in Figure 3.3 are shown in Table 3.1. The first rule is the splitting of the main  $S$  node in an  $NP$  and a  $VP$ . The second rule rewrites an  $NP$  into a POS tag ( $PRP$ ). The last two rules are extracted from the right side of the tree. If we also derive rules from the other tree (Figure 3.2), the rule “ $VP \rightarrow VBD PRP NN$ ” would be added.

A context-free grammar can be used to generate a set of trees for a given sentence, however, it does not have a preference on one tree over another, whereas in many situations it is desirable to retrieve only the most probable tree. This is where probabilistic context-free grammars come into play.

### 3.1.3 Probabilistic Context-Free Grammar

Formally, a probabilistic context-free grammar (PCFG) (Booth and Thompson, 1973) is a quintuple: on top of a standard CFG, it includes:

- $P$ : containing a probability for each rule  $\alpha \rightarrow \beta$  in  $R$ , of the form:  $p(\beta|\alpha)$ . For a PCFG to be sound the probability of all rules of every  $\alpha$  should sum to 1.0:  $\forall \alpha : \sum_j P(\alpha \rightarrow \beta_j) = 1.0$

For a simple probabilistic context-free grammar, rule probabilities can be estimated by dividing the frequency of a specific production rule by the frequency of the constituent of the left-hand side based on a training treebank:

$$p(\beta|\alpha) = \frac{\text{count}(\alpha \rightarrow \beta)}{\text{count}(\alpha)}$$

For our example grammar, this means that rule 1 and 3 in Table 3.1 get a probability of 1 (1/1). Rules 2 and 4 get a probability of 0.5 (1/2), because both of these rules occur once, and the tree has two NP's.

The probability of parse tree  $y$  containing rules  $\alpha_1 \rightarrow \beta_1, \alpha_2 \rightarrow \beta_2, \dots, \alpha_n \rightarrow \beta_n$  is defined as the product of all the probabilities of the rules used to form the tree:

$$p(y) = \prod_{i=1}^n P(\alpha_i \rightarrow \beta_i)$$

All generated parse trees can be ranked using this probability, after which the best parse tree or even the top-N trees can be used. The probability of a sentence with respect to the grammar, is the sum of the probabilities of all its possible parse trees.

### 3.1.4 The CYK algorithm

The task of a parsing algorithm is to derive the most probable parse tree for a given input sequence with respect to a grammar. In other setups, like the parsing of programming code, there should be only one possible tree for the given input. But when parsing natural language, there are often

---

**Algorithm 1** CYK algorithm

---

```

1: function CYK(words, grammar)
2:   for  $j \leftarrow 1$  to  $\text{len}(\text{words})$  do
3:      $\text{chart}[j - 1, j] \leftarrow \{A \mid A \rightarrow \text{words}[j] \in \text{grammar}\}$ 
4:     for  $i \leftarrow$  from  $j - 2$  downto 0 do
5:       for  $k \leftarrow i + 1$  to  $j - 1$  do
6:          $\text{chart}[i, j] \leftarrow \text{chart}[i, j] \cup \{A \mid A \rightarrow BC \in \text{grammar},$ 
7:            $B \in \text{table}[i, k],$ 
8:            $C \in \text{table}[k, j]\}$ 
9:   return chart

```

---

multiple possible parse trees, the algorithm is then also tasked with finding the most probable syntactic tree with respect to the grammar.

In this thesis, we will use the Cocke-Younger-Kasami (CYK) algorithm (Cocke, 1969; Younger, 1967; Kasami, 1966). This is a bottom-up, chart-based parser. More concretely, it starts to parse from the words (bottom-up) and stores partial trees in a table (chart). This algorithm requires the grammar to be in Chomsky normal form (Chomsky, 1959). This means that only rules of the form  $x \rightarrow yz$  and  $x \rightarrow q$  are allowed, where  $x$ ,  $y$ , and  $z$  depict a non-terminal ( $x, y, z \in N$ ) and  $q$  a terminal ( $q \in T$ ). However, this is no problem for context-free grammars, since every context-free grammar can be converted to an equivalent Chomsky normal form ( $G_n$ ), which yields the exact same language ( $L(G) == L(G_n)$ ). After parsing, the resulting tree can be converted back, to match the labels of the original context-free grammar.

The CYK algorithm finds all possible parse trees by iteratively filling the chart. The pseudo code of the CYK algorithm is shown in Algorithm 1. In the third line, all POS tags for each word are inserted into the chart. After having the POS tags, the algorithm starts with finding constituents with a span size of 2 (line 4) for each split position (line 5), after which it increases the span size (line 4) until the complete length of the sentence is reached. In line 6 the algorithm checks if there exists a rule that fits in this position; the presence of the left and right constituent are checked in respectively line 7 and 8.

The chart for our example constituency trees in Figure 3.2 and 3.3 is

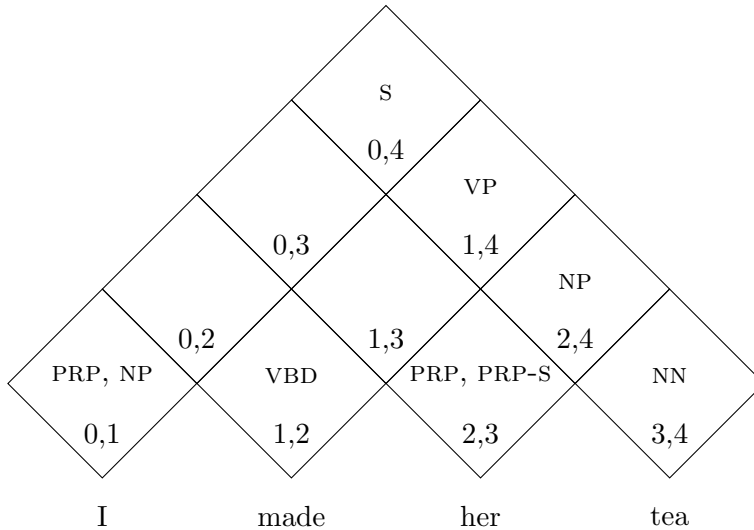


Figure 3.4: Example chart when parsing “I made her tea”

shown in Figure 3.4. At the bottom of each node, its span is shown. In the middle of each position, the possible constituents for this position are shown. The VP in position (1,4) could be created through two different paths, resulting in the two constituency trees discussed before.

This basic version of the algorithm does not take probabilities into account and returns the chart instead of a parse tree. To include probabilities, we have to multiply the probability of the left child, right child and the new rule in line 6 and store the probability in the chart. If a constituent can be composed from multiple paths, we save the highest probability, since that is the probability used for the most probable parse.

To make it easier to obtain the parse tree from the chart, backpointers can be saved every time a new constituent is added to the chart, saving the positions of its children. These backpointers can simply be followed to generate the corresponding constituency tree. If a position can be reached through multiple paths (indicating ambiguity), we store only the backpointer of the most probable sub-tree. However, if we want to retrieve the top-N trees, multiple backpointers have to be saved per position, including their corresponding probabilities.

### 3.1.5 Latent Annotation

Simple probabilistic context-free grammars learned directly from a treebank as described in Section 3.1.3 (also called a treebank grammar) have limitations. Their main weakness is the absence of context-awareness; every rule only takes the parent node and its direct children into account. This is obviously too short-sighted, since for many decisions more contextual information is necessary. A straightforward way to handle this within the PCFG framework is to define more specific syntactic categories. For example by using parent annotation (Johnson, 1998), markovization (Collins, 2003), lexicalization (Collins, 1997; Charniak and Johnson, 2005) or by splitting the syntactic categories (Klein and Manning, 2003; Matsuzaki et al., 2005; Petrov et al., 2006). Since the last strategy shows the most promising results, this approach is used later in this thesis (Chapter 7). This technique is also called latent annotation, hence the complete name becomes PCFG-LA. In the remainder of this section we will explain this extension in more detail.

Splitting the categories is motivated by the fact that the syntactic categories defined by the standard PCFG are too generic. Think for example about the differences between an active and a passive verb phrase (VP), which are used in very different syntactic contexts, however, during the parsing process they are considered to be the same. As a result of splitting the syntactic categories, a grammar can learn more specific rules, and thus recognise more specific constructions. In early work, the splitting of the constituent was done by manually designed rules (Klein and Manning, 2003). Later, algorithms were developed to automatically optimize the splits of the categories (Matsuzaki et al., 2005; Petrov et al., 2006; Petrov and Klein, 2007). We will focus on the approach of the Berkeley parser (Petrov and Klein, 2007).

This parser trains in multiple stages, in every stage the parser becomes more refined. At the beginning of each stage, all the constituents of the grammar are split into two sub-constituents, after which the performance difference for each split is estimated. If a certain split does not improve performance it is merged back again. In this way, the number of splits is tuned per constituent. The maximum number of splits is the number of parsing sieges to the power of two. The optimal amount of splits was empirically shown to be six (Petrov et al., 2006), the maximum number of sub-constituents for a constituent is then:  $2^6 = 64$ .

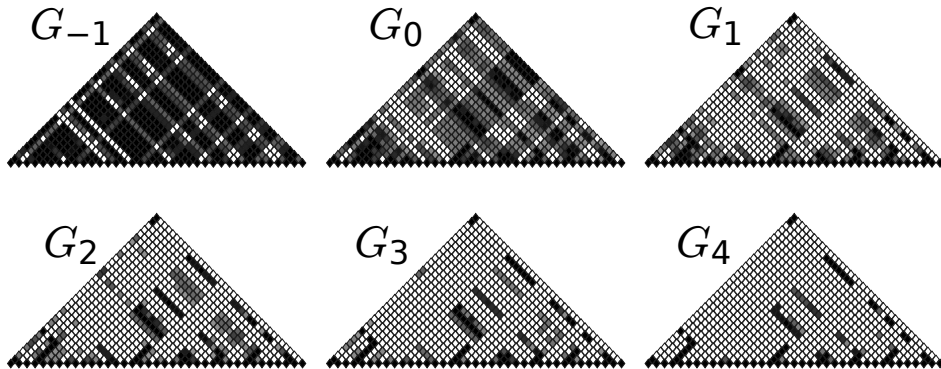


Figure 3.5: Example of the charts in the different parsing sieges of the Berkeley parser. The darker the position in the chart, the more probable it is (pruned positions are displayed in white).

At parsing time, this refined grammar is problematic due to the large number of terminals and non-terminals. This results in an impractically slow parser; the search space becomes too large. To maintain efficiency, the parser is, similar to the grammar learning process, also divided into multiple sieges. This is also called hierarchical coarse-to-fine parsing. In this setup, we start parsing with a very general grammar. After each parsing siege, unlikely constituents are pruned away and a finer grammar is used.

Figure 3.5<sup>1</sup> visualizes the hierarchical parsing process of the Berkeley parser. The color of the nodes in the chart indicate how probable it is that this node survives according to the parsing algorithm. In the actual algorithm there is also a third dimension containing the constituents which is not visualized here; the probabilities of all constituents are merged per position. The  $G_{-1}$  grammar is a grammar in which all non-terminals are merged to the same label, making it even coarser than a standard PCFG. It is clearly visible that the pruning helps to focus the attention of the parser in a certain direction, leading to a much smaller search space. Petrov and Klein (2007) show that this hierarchical parsing leads to a 100 times speedup, without significant loss in accuracy.

<sup>1</sup>taken from Petrov and Klein (2007)

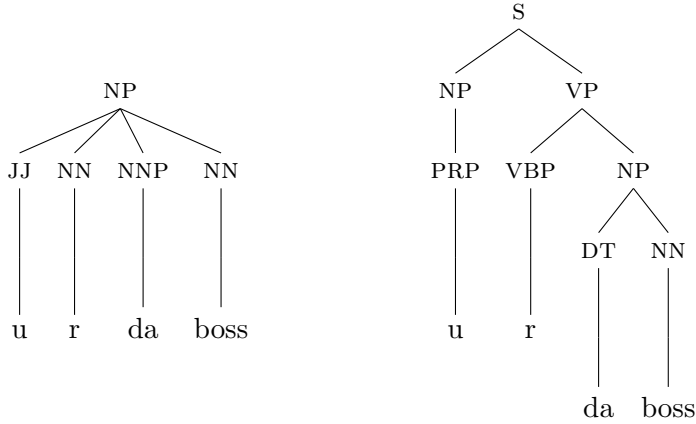


Figure 3.6: Output of Berkeley parser (left) and the correct constituency tree (right) for: “u r da boss”

### 3.1.6 Error Analysis of an Out-of-the-box Parser on Social Media Data

To illustrate the problems current parsers are suffering, we will use two example sentences, taken from the microblog service Twitter. These are parsed with the Berkeley parser trained on the standard training split of the Wall Street Journal part of the Penn Treebank (Marcus et al., 1993), which consists of news texts.

The output of the Berkeley parser for the first example sentence is shown in Figure 3.6. This Tweet contains only one correctly spelled token: ‘boss’, which is also the only correctly tagged word. ‘da’ and ‘r’ also occur in the training data in different contexts; ‘da’ is often part of a name, and is thus tagged as proper noun (NNP). ‘r’ is only used as a separate letter in the training data, where it is tagged as noun. Because ‘u’ is an unknown word, the parser used context to guess that it is probably an adjective, because the next word is a noun. Because of these mistakes on the word-level, the parser completely fails to recover the structure of the sentence.

The second example, shown in Figure 3.7<sup>2</sup>, has more standard tokens.

<sup>2</sup>It should be noted that the annotated tree is according to the interpretation of the author of this thesis

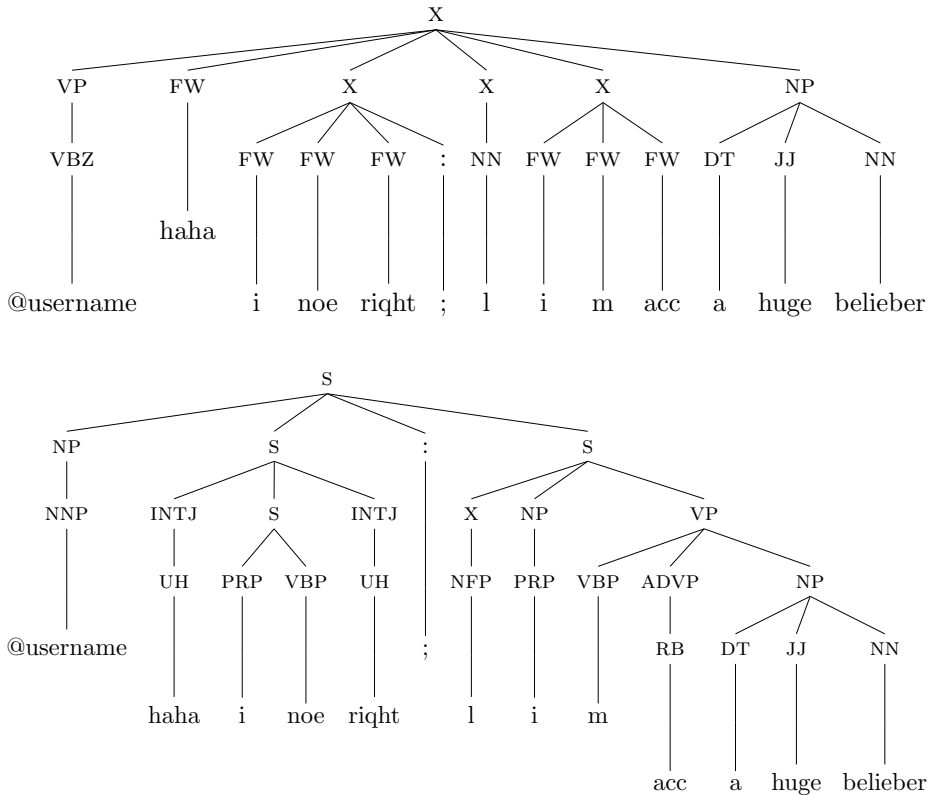


Figure 3.7: Output of Berkeley parser (top) and the correct constituency tree (bottom) for: @username haha i noe right ; l i m acc a huge belieber

Consequently, some parts of this sentence are parsed correctly, like ‘a huge belieber’. But most of this tree is also parsed incorrectly. The first token is already labeled wrong, the username in this sentence was lexically similar to a verb (note that it is replaced by ‘@username’ in Figure 3.7, for privacy reasons).

the first subsentence (‘haha i noe right’) is recognized correctly as being part of the same subtree. However, the inner structure is completely wrong. This is mainly due to ‘i’ in lowercase being present in the training data as foreign word (FW). This leads to a propagation of errors, because the

probability for unknown words to be a foreign word is also relatively high, especially if they occur next to other foreign words. Exactly the same mistake is made for the sequence “i m acc”. An unknown word that is tagged correctly is ‘belieber’. This is because the Berkeley parser contains a heuristic which groups unknown words ending in ‘er’. Most of these words will be nouns, so this word is also tagged as noun.

The ‘l’ in the middle of the sentence does not add anything to the syntactic structure of the sentence and is probably a typographical error. Tokens like this can probably better be ignored in the parsing process (as suggested by Kong et al. (2014)).

As a result of the errors made in the word labeling, there are too many foreign words in the tree. This is why the rest of the tree consists mainly of the phrase label x, indicating unknown, uncertain, or unbracketable subtrees.

These examples have shown that a constituency parser trained on news texts breaks down completely when encountering the substantially different language of social media. In the following section, we will describe the formalism of dependency trees and dependency parsing.

## 3.2 Dependency Parsing

This section starts with an explanation of the structure of dependency trees. After this, we will discuss transition-based algorithms, and explain the arc-standard transition system in more detail. Finally, we will shortly explain how neural networks can be used in transition-based algorithms.

### 3.2.1 Dependency Trees

In dependency trees, the words are directly connected to each other, i.e. the words are the nodes. The connections between the words are called arcs or edges. These connections are directional: the *dependent* is connected to its *head*. The type of relation between these is classified using a relation label. Each word can have multiple (or zero) dependents, whereas every word has exactly one head. The head of the sentence is usually connected to a dummy ROOT node, this is done to comply with the constraint that every word has exactly one head, and simplifies parsing. A well-formed

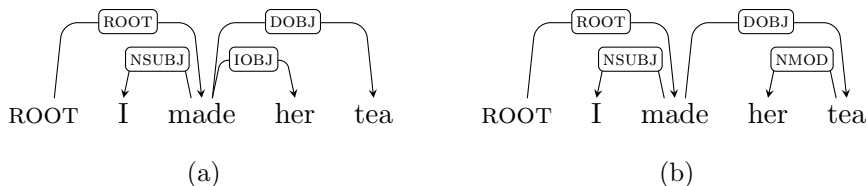


Figure 3.8: Dependency trees for both interpretations of the sentence “I made her tea”

dependency tree has to satisfy three constraints:

- **Connected:** Every word can be reached from the ROOT.
- **Acyclic:** It should not be possible to pass a word twice while traversing a directed path.
- **Single-Head:** Every node has exactly one incoming arc (one parent).

Additionally, another important property which may or may not be met is:

- **Projectivity:** For every arc in the tree, every node falling within the span of this arc can be reached from the head of the arc. In other words, the arcs do not cross each other when visualizing the dependency tree.

Projectivity is an important property because non-projective trees are more complex to parse. The importance of being able to parse non-projective trees depends on how prevalent they are in the data. The number of non-projective dependency structures differs per dataset and is mainly dependent on the language and annotation guidelines.

The two different dependency trees for both interpretations of the sentence “I made her tea” are shown in Figure 3.8. The left dependency tree corresponds to the interpretation that I made a tea for ‘her’, whereas the dependency tree on the right corresponds to the meaning that I made a tea, which now belongs to ‘her’.

The ROOT of the sentence is the main verb ‘made’ for both interpretations. The nominal subject (NSUBJ) of this verb is ‘I’, and the direct object (DOBJ)

is ‘tea’. The only differently attached word between these two trees is ‘her’; in the left tree, it is an indirect object (IOBJ) of made. In the right dependency tree, ‘her’ is a nominal modifier (NMOD). This difference can be observed more clearly by replacing the noun ‘tea’ with nouns which only allow one interpretation: “I made her an offer” would lead to the first dependency tree, whereas the dependency tree of “I made her closet” would look like the second dependency tree.

For dependency parsing, different parsing strategies are commonly used compared to constituency parsing due to the substantially different structures. In this thesis (Chapter 8), we will only make use of a transition based parser. Hence, this is the parsing system that we explain in the following section.

### 3.2.2 Transition-based Parsing

In transition based parsing, the parsing problem is simplified down to a prediction of a sequence of transitions. The parser starts in an initial state, and after each transition the state is changed, until a terminal state is reached. There are different transition systems, which use different definitions for a state and have different sets of transitions. In the remainder of this section, we will explain the arc-standard transition system (Nivre, 2004) in more detail, which is very similar to the arc-hybrid system (Kuhlmann et al., 2011) used in Chapter 8.

To simplify the algorithm, we insert an artificial ROOT token in front of the sentence, we then denote the root followed by all words of a sentence of length  $n$  with  $w_0, \dots, w_n$ , so  $w_0$  is the ROOT. A state is represented by a triple, containing:

- $B$ : buffer containing all words that still need to be processed.
- $S$ : stack, which stores words until their parent is found.
- $A$ : set of arcs  $a$ , where each arc has a begin position, end position and relation label.

The initial state looks as follows:

$$B = [w_1, \dots, w_n]$$

$$S = [w_0]$$
$$A = []$$

When the buffer  $B$  is empty, and only the `ROOT` ( $w_0$ ) is left in  $S$  the parse is completed. The terminal state looks like:

$$B = []$$
$$S = [w_0]$$
$$A = [a_1, \dots, a_n]$$

The transitions in the arc-standard system are:

- **Shift:** moves the top word from the buffer to the stack.
- **Left-arc:** adds an arc from the top of the stack to the second item of the stack. Also removes the second item from the stack, since it now has an incoming arc.
- **Right-arc:** adds an arc from the second item of the stack to the top of the stack. Additionally removes the top item from the stack.

So when an incoming arc is defined for a word, it is removed from the stack. Now the word is connected to its parent; this process is repeated until a word becomes dependent of the `ROOT` node. This is the reason that the root node is left in the stack in the terminal state; `ROOT` should not have an incoming arc.

The algorithm starts in the initial state, and runs a sequence of transitions until a terminal state is reached. The decision on which transition to apply next is made by a classifier which extracts features from the current state. This classifier is trained on the sequences of states that are necessary to produce the trees occurring in the training data. To be able to generalize, features based on POS tags are often used. Some parsers expect POS tagged input and are thus relying on an external tagger, whereas others have a built-in tagger.

The transitions that are required to parse our example tree in Figure 3.8a is shown in Table 3.2. ‘I’ and ‘made’ are first shifted onto the stack, followed by a left-arc transition, resulting in an arc from ‘made’ to ‘I’. After this, ‘her’ and ‘tea’ are connected to ‘made’ by a shift and right-arc transition

action	$B$	$S$	$a$
	[I, made, her, tea]	[root]	
shift	[made, her, tea]	[root, I]	
shift	[her, tea]	[root, I, made]	
left-arc	[her, tea]	[root, made]	made $\mapsto$ I
shift	[tea]	[root, made, her]	
right-arc	[]	[root, made]	made $\mapsto$ her
shift	[]	[root, made, tea]	
right-arc	[]	[root, made]	made $\mapsto$ tea
right-arc	[]	[root]	root $\mapsto$ made

Table 3.2: Sequence of transitions to parse “I made her tea”

for both. Finally, only ‘made’ is left in the stack and is connected to the artificial ‘root’ node by a right-arc transition.

For the alternative parse, after the first LEFT-ARC all words would be shifted on the stack, after which a LEFT-ARC transition is done to connect an arc from ‘tea’ to ‘her’. Then two RIGHT-ARC transitions are used to connect ‘tea’ to ‘made’, and ‘made’ to ‘root’.

There is still one thing missing: the labels. These are normally added when the transition left-arc or right-arc is done. This can be done using a separate classifier, or the choice can be combined with the action directly.

This basic version of the algorithm can only build projective trees. To extend this algorithm to also parse non-projective trees, another transition can be added: the SWAP transition (Nivre, 2009). In short, the SWAP transition moves the second node on the stack back to the buffer.

The arc-standard transition system builds the dependency tree completely bottom-up: an incoming arc can only be added to a node if all of its children are already known. This has potential disadvantages for finding the correct sequence of transitions, since the attachment of right dependents has to be postponed. There are numerous other shift-reduce systems which parse in a different order (Nivre, 2004; Kuhlmann et al., 2011; Goldberg and Elhadad, 2010). However, these are beyond the scope of this background chapter.

### 3.2.3 Neural Network Transition-based Parsers

Transition-based neural network parsers (Chen and Manning, 2014) are very similar to the feature based transition parsers described in the previous section. However, instead of processing words directly, these parsers process continuous vectors which represent the input words. The vectors can be constructed in different ways; the most straightforward is to derive them based on the training data. In this setting, the vectors are randomly initialized and are then optimized for parsing during training. The vector for a word can be complemented with vectors derived by running a separate neural network over the characters of the word (Ballesteros et al., 2015), or with vectors based on externally learned vectors, which are induced from large amounts of raw texts.

The algorithm requires only little adaptation to process vectors instead of words. Instead of processing words in the stack and the buffer, vectors are now used. The classifier which decides which transition to apply next is now replaced by a neural network which uses the sequence of continuous vectors, instead of features extracted from the current state, to predict the next action.

## 3.3 Treebanks

In this section we will review the treebanks used in the remainder of this thesis. For each treebank, we will start by listing some basic properties: the format, domain, size, and average sentence length in number of words. Secondly, we will shortly describe the treebank in more detail. Thirdly, we show some example sentences from the treebank to illustrate the nature of the data. All the treebanks discussed in this section contain English data.

### 3.3.1 Wall Street Journal (WSJ)

Format: Constituency

Size: 33,036 sents

Domain: News texts

Average sentence length: 24

The Wall Street Journal part of the Penn Treebank (Marcus et al., 1993) has been the default benchmark for comparing parsers since its release in 1992. Even for dependency parsing, automatic conversions of this

treebank have been used as a standard benchmark for a long time (more recently, the Universal Dependencies (Nivre et al., 2017) is starting to take over this role). As the name suggests, this treebank consists of news texts from the Wall Street Journal, more precisely 2,499 newspaper articles from 1989 annotated with POS tags and constituency structure, divided into 25 sections. Section 02-21 are usually used for training, section 22 for development and section 23 for testing.

### Example Sentences

Pierre Vinken , 61 years old , will join the board as a non-executive director Nov. 29 .

Tass said the final budget and economic plan calls for a sharp increase in the production of consumer goods .

Westinghouse Electric Corp. also won a \$ 75.5 million Navy contract for nuclear propulsion parts .

His interest in the natural environment dates from his youth .

### 3.3.2 English Web Treebank (EWT)

Format: Constituency/dependency	Size: 16,520 sents
Domain: 5 web domains	Average sentence length: 15

The shared task on parsing the web (Petrov and McDonald, 2012), hosted at SANCL 2012, introduced the English Web Treebank (Bies et al., 2012). This treebank consists of data from five different web sources: weblogs, reviews, newsgroups, emails, and question-answer websites. This treebank enables training and testing of constituency parsers on the non-canonical data of the web. The annotation format is very similar to the WSJ, with some adaptations for web phenomena (Bies et al., 2012).

For the Universal Dependencies project, the first English treebank was an automatic conversion of the English Web Treebank (Silveira et al., 2014) to the Universal Dependency format. By now, this treebank has undergone multiple rounds of corrections.

### Example Sentences

Filled up on too much beer and hence can not comment on the food .

We were familiar with Search Engine Optimization strategies, but new nothing about Social Media - we just heard that it was the next big thing .

If this coincides with rising interest rates and a setback in the housing market , American consumers will experience the hardest times since the Great Depression .

I HAD TO WAIT FOR MY WAITRESS .

#### 3.3.3 Web2.0 treebank

Format: Constituency

Size: 519 tweets

Domain: Twitter

Average tweet length: 33

Foster et al. (2011b) introduced a treebank containing data from Twitter and a football forum. In this thesis, we only use the Twitter data. For this treebank, the annotation guidelines for the Penn Treebank (Bies et al., 1995) were used, with some small adaptations for the Twitter domain (usernames, hashtags, and URLs are annotated as an NNP under an NP).

### Example Sentences

Does anyone else think Lloyds TSB went under because of the horrible music on their TV adverts ?

I 'm cheering on Wales while decorating again .

Hahaha nice fall from grace !

Cheap is all that it 's got going for it .

#### 3.3.4 MoNoise treebank

Format: dependency

Size: 632 tweets

Domain: Twitter

Average tweet length: 16

This treebank is created specifically to evaluate the method introduced

in Chapter 8. This treebank consists of a development split and a test split both originating from previous Twitter corpora. These tweets were selected to contain a certain level of non-standardness. Annotation was done in accordance with the Universal Dependencies format, similar to the EWT. The text in this treebank is also annotated with a lexical normalization layer. More details on this treebank can be found in Section 8.2.

### Example Sentences

```
@WilliefknUnique hahah well that a diff story then :P lmao he
doesnt wanna talk to ya. hes talkin to me foo! ;)
```

```
Iyaz is an ugly mofuhka. Lol iv seen him super up close.
#randomthought
```

```
I wud like to know why my lightbulb is behaving in such a
manner. The thing wn't turn on. Forces me to leave ma room
to read. Chah!
```

```
@inglewoodtip email comin yo way in 15 mins
```

## 3.4 Summary

In this chapter we gave an overview of the parsing systems used in Chapter 7 and Chapter 8 of this thesis. In particular, two different types of parsing are discussed: constituency parsing and dependency parsing.

Starting with constituency parsing, we overviewed the syntactic formalism behind constituency trees and discussed a probabilistic context-free grammar and how it can be used for parsing in the CYK algorithm. Finally, we discussed how latent annotation can be used to generate grammars with more specific rules, which leads to more accurate parsing. The resulting grammar quickly becomes too large for practical use, which can be solved with hierarchical parsing.

For dependency parsing, we also discussed the syntactic formalism first. This was then followed by shift-reduce parsing, in particular, the arc-standard system. Third, we showed how to use this in a neural network set-up, which will be the starting point in Chapter 8.

Finally, we shortly overviewed the treebanks used in the remainder of this thesis.



**Part II**

**Lexical Normalization**



## Chapter 4

# MoNoise: A Modular Approach to Normalization

Lexical normalization is the task of converting non-standard text to a standard register. This can be beneficial for natural language processing since models based on canonical language will perform better on standardized data. For more details on the task of normalization, we refer to Section 2.1. In this chapter we will motivate the design choices of our own model: MoNoise.

Previous work on lexical normalization can broadly be divided into two approaches, both making use of previously existing frameworks. The first group uses techniques from automatic spelling correction (Jin, 2015; Xu et al., 2015; Han and Baldwin, 2011); which can be considered a sub-task of normalization (Section 2.3). The second approach exploits machine translation models (Ljubešić et al., 2016; Li and Liu, 2012), where social media data is considered as the source language and canonical English as the target language.

Because machine translation models are limited to handle phenomena occurring in the training data, we choose to base our model on the traditional framework of automatic spelling correction. This framework consists of three steps: detection of erroneous tokens, generation of correction candidates and the ranking of these candidates. In our setup, we skip the detection step and consider all words for normalization, this is motivated in Section 4.2. For the generation step, we will use a traditional spelling correction algorithm

supplemented with, among other modules, a word embeddings module and a translation dictionary. For ranking, we exploit features extracted from the ranking components, and complement these with additional features, from which n-gram features are the most informative. This setup should be more robust to data outside of the scope of the training corpus since it exploits external raw data, which can easily be obtained for other domains or time-spans.

In this chapter, we will start with describing the framework used by most previous work on automatic spelling correction, which is a sub-task of normalization. Next, we will give an overview of MoNoise; our modular approach to normalization. Finally, we will explain the choices made for the two separate parts of MoNoise: candidate generation and candidate ranking.

This chapter is based on:

Rob van der Goot and Gertjan van Noord. MoNoise: Modeling noise using a modular normalization system. *Computational Linguistics in the Netherlands Journal*, 7:129–144, December 2017a

The most notable difference is the ORIGWORD feature, which is not present in the paper.

An earlier version of MoNoise was described in:

Rob van der Goot. Normalizing social media texts by combining word embeddings and edit distances in a random forest regressor. In *Normalisation and Analysis of Social Media Texts (NormSoMe)*, 2016

The source code of MoNoise is available at:

<https://bitbucket.org/robvandergerg/monoise>

## 4.1 Automatic Spelling Correction

The task of normalization has many similarities with the task of automatic spelling correction. More precisely, the replacements necessary for automatic spelling correction is a subset of all replacements required for normalization. Because of these similarities, the framework for automatic spelling correction is often used for the normalization task. This framework is as follows:

1. Error detection: detect which words are in need of normalization. For spelling correction, this is usually done with a dictionary. This approach has difficulties with detecting real-word errors; erroneous tokens which are included in a dictionary, e.g. ‘were’ $\mapsto$ ‘where’.
2. Candidate generation: generate potential normalization candidates for each word. Besides finding the correct candidate, it is also important to keep the list of normalization candidates small, because large lists of candidates would complicate the next step.
3. Candidate ranking: takes the generated candidate list from the previous step as input, and ranks these candidates. The highest ranked candidate can then be used as the correct candidate. Note that in commonly used spelling correction systems, user intervention is still required to pick the correct candidate from this list.

For automatic spelling correction, the generation and ranking step are often based on edit distances. The most standard edit distance is the Levenshtein distance (Levenshtein, 1966); which defines the distance between two words as the minimal number of single character edits (insertion, deletion or substitution) it takes to transform one word into the other. To use this for spelling correction, the original word is compared to all words in a dictionary, and the candidate with the lowest Levenshtein distance is then considered to be the best candidate. A wide variety of different alternations of the Levenshtein is used for the task of spelling correction. The weight for different character edits can be tuned based on training data or be set manually (based on phonetics, shape, distance on keyboard, etc.). Another popular variety is to calculate the edit distance on a phonetic transcription instead of the words directly. Commonly used transcriptions include the Double Metaphone algorithm (Philips, 2000), Soundex (Odell, 1956) and ARPAbet (Gillman, 1974). For an elaborate overview of approaches to spelling correction, we refer to Kukich (1992).

If only one edit distance is used, the candidate ranking can simply be based on this distance. When a combination of edit distances is used, they should be combined for the ranking. This can be done based on training data (supervised), or they are combined using a formula which assigns a weight to the different distances.

Because the normalization task includes a wider variety of replacements compared to spelling correction, the aforementioned approaches are inadequate. As seen in Section 2.3, spelling mistakes are only a subset of all normalization replacements. Many intentional anomalies like ‘ppl’ $\mapsto$ ‘people’ and ‘cuz’ $\mapsto$ ‘because’ are not handled in spelling correction models. These replacements have relatively large edit distances. For this reason, supplementary methods are necessary for normalization; these can be used within the same framework.

## 4.2 MoNoise: Overview

In this section we propose the main framework of our normalization model. The two main parts are then discussed in more detail in the following two sections.

The traditional approach to automatic spelling correction contains three steps, which are detailed in the previous section. In contrast to this approach, we will skip the error detection step and assume that every word can be erroneous. Because of this, candidates are generated for every word and the decision whether to normalize is postponed until the ranking step. The two main advantages are that no errors can be propagated from the error detection step and that a more informed decision can be made during the ranking step with all features for the original word and the normalization candidates available. The main disadvantage is that it may be less efficient, since we have to generate and rank candidates for every word. This has been done in previous work (e.g. Jin (2015); Schulz et al. (2016)), but these models have a much smaller number of candidates compared to our approach. We examine the effect of having a separate error detection step in more detail in Section 5.4.4.

Table 4.1 shows the outcome of the generation step for an example tweet. Here, we only show the top 4 candidates, in practice a much larger number of candidates is generated. Several normalization candidates are generated from a variety of modules. These modules are described in more detail in the next section.

The task of picking the correct candidate can be seen as a binary classification task; a candidate is either the correct candidate or not. However, we cannot use a binary classifier directly because we need exactly one ‘correct’

original word	mostt	social	ppl	r	troublesome
candidates	mostt	social	ppl	r	troublesome
	most	socials	pol	ri	trouble some
	misty	media	people	rnt	bothersome
	mosttt	socially	ppl	ra	troubles

Table 4.1: Example of Candidate Generation

candidate per position, whereas the classifier might classify multiple or zero candidates per position as correct. Instead, we use the confidence of the classifier that a candidate belongs to the ‘correct’ class to rank the candidates. This has the additional advantage that it enables the system to output lists of top-N candidates. An example of the training data is shown in Table 4.2; these are some of the datapoints generated from the word ‘ppl’. The details of the features as well as the classifier are discussed in more detail in Section 4.4.

### 4.3 Candidate Generation

In this section, we will first discuss how candidates are generated in previous work and motivate how this leads to the choice of modules used in MoNoise. Then, we discuss all of our generation modules in more detail.

Candidate	Feat1	Feat2	Feat3	...	Gold label
ppl	1.0	0.01	0.42	...	0
pol	0.0	0.00	0.03	...	0
people	0.0	0.24	0.12	...	1
ppl	0.0	0.05	0.08	...	0

Table 4.2: An example of input for the classifier for the word ‘ppl’

### 4.3.1 Previous Work

As discussed in the beginning of this chapter (Section 4.1) spelling correction systems can already solve a subset of the problems of lexical normalization. Much of the previous work on normalization borrows techniques from spelling correction systems. However, the normalization task includes a wider range of replacements types compared to spelling correction, so complementary methods are necessary.

As explained in Section 4.1 previous work on spelling correction usually exploits edit distances to generate correction candidates. Some previous work on normalization defined and tuned their own methods for lexically similar normalization replacements. These include edit distances on the character level (Han and Baldwin, 2011; Hassan and Menezes, 2013), or on a phonetic transcription of the word, like ARPAbet (Xu et al., 2015) or the Double Metaphone algorithm (Han and Baldwin, 2011; Mosquera et al., 2012). Often, repetitions of characters are first removed; Jin (2015) instead uses overlap of character n-grams to compare words, which intrinsically handles this phenomenon.

Because these different methods require extra tuning steps, and the problem of spelling correction has already been studied for decades, existing modules can be used for this sub-problem of normalization. In previous work, spelling correction systems like Hunspell (Schulz et al., 2016), Aspell (Sharma et al., 2016) or the Jazzy spell checker (Liu et al., 2012; Li and Liu, 2012) are used to this end. Our motivation to use an existing spell checker is threefold: we do not have to reinvent the wheel, no extra tuning parameters are added, and models are already available for multiple languages. Since the performance of Jazzy, Hunspell, and Aspell is very competitive, we choose to use Aspell for practical reasons (it has a c++ API and is available for many languages).

As shown in Section 2.3, not all normalization replacements fall in the spelling correction category. Some replacements look quite different on the surface (e.g. `cuz`→`because`, `ppl`→`people`, `cud`→`could`). For these anomalies, the previously mentioned methods are inadequate. Some of these anomalies are very frequent, and can simply be looked up in the training data. For the remainder of the more lexically distant cases, supplementary methods are necessary. In previous work, machine translation models on the character level are used to learn how to ‘translate’ these domain-specific anomalies

to their normal equivalent (Li and Liu, 2012; Schulz et al., 2016; Ljubešić et al., 2016). More concretely, these are sequence-to-sequence translation models which operate on the character level. Other work uses bipartite graphs (Hassan and Menezes, 2013; Ren et al., 2016), which model how often words occur in similar contexts. The rationale behind this, is that the anomaly often occurs in similar contexts as its normalized counterpart.

More recently, word embeddings have shown to be able to effectively model distributional similarity for a variety of tasks. In word embeddings, words are represented using continuous vectors. In other words, each word is mapped to a shared vector space, in which similar words are positioned close to each other.

There are several ways in which these continuous vectors can be derived. In this thesis, we will make use of the popular skip-gram implementation of word2vec (Mikolov et al., 2013a,b) to derive the word embeddings. These embeddings are motivated by the distributional hypothesis (Harris, 1954); which states that words which occur in similar contexts have a similar meaning. In this training procedure, the continuous vector for each word is constructed using its context: the  $K$  neighbouring words to the left and right, also referred to as window. The objective function during the learning of the embeddings is to maximize the likelihood of the prediction of the context based on the word itself. By learning how to predict the surroundings of a word, the representation of each word is based on context. This leads to a vector space where words occurring in similar contexts are positioned close to each other.

To the best of our knowledge, word embeddings are only used once before for the task of normalization, in an unsupervised setting (Rangarajan Sridhar, 2015). In their model, a mapping between anomalies and standard words is derived from the vector space. This is done by finding the 25 closest candidates in the vector space for canonical words using cosine distance, and filter these based on lexical edit distance. An advantage of their approach is that it can easily be adapted to a new timespan or domain, since only raw data is used.

### 4.3.2 Modules

We use several different modules for candidate generation. Each module is focused on a different type of anomaly. In this section we give a short

description of each module and illustrate their strengths with some examples.

**Original token** Because we do not include an error detection step, we need to include the original token in the candidate list. This should provide the correct candidate in approximately 90% of the cases for our corpora (see Table 2.1 on page 19).

**Word embeddings** We induce word embeddings from our social media data. In the resulting vector space we look for words which have a small cosine distance to the original word. These words are used in similar contexts, and are thus likely to be good normalization candidates. We use the skip-gram model provided by word2vec (Mikolov et al., 2013a) with default settings, except for a vector size of 400, which empirically showed slightly better performance compared to the default of 100. We also tried to use a smaller context window, but this led to slightly lower performance. Some examples of correctly found normalization replacements are shown below:

	u	abt	lil
1.	yu	about	little
2.	you	abut	lul
3.	ypu	abt	lor

**Aspell** We use the Aspell spell checker to repair typographical errors. Aspell uses a combination of a weighted character edit distance, and the Double Metaphone algorithm (Philips, 2000) to generate similar looking and similar sounding words. We also experimented with enabling the BAD-SPELLERS mode, which uses higher thresholds to find candidates and thus results in much larger candidate lists. For some corpora, this resulted in a slightly higher performance, whereas for other corpora it resulted in a slightly lower performance. However, the larger number of candidates results in a much slower model. Hence, we decided not to use the BAD-SPELLERS mode in our experiments.

	brotha	dressin	definitely
1.	broth	dress-in	definitely
2.	broths	dressing	defiantly
3.	brother	treason	definite

**Lookup list** We generate a list of all replacement pairs occurring in the training data. During run-time, all potential replacements which were found in the training data are used as candidates.

	ur	usa	lol
1.	your	usa	laughing out loud
2.	you're	use	
3.	you		

**Word.\*** As a result of space restrictions and input devices native to this domain, Twitter users often use abbreviated versions of words. To capture this phenomenon, we include a generation module that simply searches for all words in the Aspell dictionary which start with the character sequence of our original word. To avoid large candidate lists, we use this module only for words longer than two characters.

	cont	rec	def
1.	context	recipe	definitely
2.	continued	recoloured	defeated
3.	contact	recorder	defence

**Split** We generate word splits by splitting a word on every position and checking if both resulting words are canonical according to the Aspell dictionary. To avoid over-generation, this is only considered for input words longer than three characters.

	alot	bestfriends
1.	a lot	best friends
2.		

## 4.4 Candidate Ranking

In this section, we will first give an overview of methods used for the ranking of candidates in previous work. Secondly, we describe the features and the classifier that we use.

### 4.4.1 Previous Work

The strategies used for ranking can broadly be divided in unsupervised and supervised methods. Unsupervised methods often exploit n-gram frequency information. These can be used directly in a language model (Mosquera et al., 2012; Xu et al., 2015; Yang and Eisenstein, 2013; Schulz et al., 2016), or in a Viterbi decoding (Hassan and Menezes, 2013; Li and Liu, 2012). Li and Liu (2015) go one step further and model words paired with potential POS tags in a joint Viterbi decoding. The main advantage of these approaches is that n-gram probabilities can reliably be estimated due to the large amounts of publicly available data.

In supervised approaches, features are often extracted from the generation modules. Most types of generation modules naturally provide some type of ordering or scoring. Since this is also the case for our generation modules, we choose to use a supervised classifier. Furthermore, annotated datasets are available for multiple languages, and are relatively fast and cheap to create; no specially trained experts or linguists are required for this task. Previously used supervised methods include conditional random field classifiers (Liu et al., 2012; Chrupała, 2014; Akhtar et al., 2015) and a random forest classifier (Jin, 2015). Commonly used features include n-gram probabilities, different types of edit distances and frequency counts in training data. Unsurprisingly, these approaches commonly achieve much better performances compared to the unsupervised settings, which is another motivation to exploit the available data.

### 4.4.2 Features

We decided to use a random forest classifier to rank normalization candidates, which is motivated in more detail in the next section. However, we would like to note here that missing features are automatically recognized by the classifier. For example for distance-based features (lexical or distributional),

a low value would be an indication that the candidate is a good candidate. If we cannot score a candidate for a specific feature, for most classifiers it would be beneficial to give them an artificial high value, indicating that the candidate is very distant. However, in our setup we use 0 as a value for that feature; the random forest classifier learns to recognize these cases automatically.

Below, we list all our feature (groups), together with a description. We start with features originating from the generation step.

**Original** A binary feature which indicates if a candidate is the original token.

**Word embeddings** We use the cosine distance between the candidate and the original word in the vector space as a feature. Additionally, the rank of the candidate in the returned list is used as feature. For the words generated by other modules, we also calculate the embeddings distance.

**Aspell** Aspell returns a ranked list of correction candidates. We use the rank in this list as a feature. Additionally, we use the internally calculated distance between the candidate and the original word; this distance is based on lexical and phonetic edit distances. Both of these features are only used for candidates generated by the Aspell module.

**Lookup-list** In our training data, we count the occurrences of every correction pair. This count is used as feature for the normalization candidate. We also use the counts of words which are not normalized, which is a feature for the original word. In this case, high frequencies indicate that normalization is probably not necessary.

**Word.\*** In social media data, people often used shortened versions of words. This binary feature indicates whether the original word is a prefix of the candidate.

**Split** A binary feature indicating if the insertion of a space into the original word can lead to this candidate.

**N-grams** We use two different n-gram models from which we calculate the unigram probability, the bigram probability with the previous word and the bigram probability with the next word. For each language we calculate these probabilities using the corpora described in Section 2.2.2: one corpus consisting of social media texts and one more canonical corpus.

**Dictionary lookup** A binary feature indicating if the candidate is present in the Aspell dictionary.

**Character order** We also include a binary feature indicating if the characters of the original token occur in the same order in the proposed candidate. In other words, the correct normalization can be obtained by only inserting characters into the anomaly. This feature is indicative for different shortening strategies which are common in social media data. For example, ppl→people would match this criterion:

```
p           p l
p e o p l e
```

**Length** A feature indicating the length of the candidate in number of characters. The length of the original word is taken into account in the last feature group described in this section. These two features are probably interactive; the correct normalization is usually a few characters longer compared to the original word.

**ContainsAlpha** A binary feature indicating whether a token contains any alphabetical characters. In some corpora, punctuation is kept untouched whereas in other it is normalized (e.g. '!!!'→'!'). This feature is added to tune the model towards these annotation decisions.

**OrigWord** For the task of normalization, usually only approximately 10% of all words need to be replaced. In our setup we treat the original words similar as any other normalization candidate that has to be ranked, with the only difference that this is indicated in a binary feature. However, the decision whether to normalize is based solely upon the original word, hence this is a special candidate. In other words, for every candidate the original

word is relevant to decide their probability. To reflect upon this in our model, we copy the values of features of the original word to every candidate. For some features, however, the original words will always have the value 0, these are not copied. Feature groups which are copied are: LOOKUP-LIST, N-GRAMS, DICTIONARY LOOKUP, CHARACTER ORDER, LENGTH, and CONTAINSALPHA. In this setup, the prediction of each candidate is also based on properties of the original word. To the best of our knowledge, we are the first to use this method for the normalization task nor any other task. This method could be beneficial for tasks in which one of the candidates has a special status, or a high prior probability.

### 4.4.3 Classifier

To recall, we consider the task of finding the correct normalization candidate as a binary classification task: a candidate is either correct or incorrect. However, a binary classifier is not guaranteed to only assign one normalization candidate to the correct class. To circumvent this issue, we use the confidence score of the classifier to rank the candidates, and use the highest ranked candidate as final normalization.

We choose to use a random forest classifier (Breiman, 2001) for the ranking of candidates. This choice is motivated by the observation that the problem of normalization can be divided into a variety of categories of normalization replacements (see Section 2.3). Each of these categories will probably behave different feature-wise, whereas in our training data this is not taken into consideration. Our hypothesis is that the random forest classifier will learn to model some of these categories intrinsically. Different sub-forests will then be used for different types of normalization replacements. More concretely: if a candidate scores high on the Aspell feature (it has a low edit distance), this can be an indicator for a specific set of trees to give this candidate a high score. At the same time, the model can still give very high scores to candidates with low values for the Aspell features by using another sub-set of trees. Another advantage is that a random forest classifier can handle binary, integer and floating point feature values. We use the random forest implementation of Ranger (Wright and Ziegler, 2017), with default parameters.

The main disadvantage of using a random forest classifier is that it cannot directly model the decisions made on neighbouring words, whereas

this is more easily done by a conditional random fields classifier or the Viterbi algorithm. In a random forest classifier, context can be modelled by making use of features based on n-grams, but this is non-trivial to do for multiple neighbouring normalization candidates. In the normalization corpora used in this thesis, approximately 1% of all words are an anomaly followed by another anomaly. This shows that the problem is present, but not very frequent. If we take into account that the prediction for neighbouring candidates might also be wrong, and taking them into account will only be beneficial in some cases, this disadvantage is relatively small.

## 4.5 Summary

In this chapter we have discussed several datasets annotated for normalization. Furthermore, we have described methods used by previous work and motivated the approaches chosen for our proposed normalization model: MoNoise.

MoNoise consists of two main parts: candidate generation and candidate ranking. Compared to the traditional framework for automatic spelling correction, we skip the error detection step. This step is normally used to identify erroneous words. The main motivations to skip this step are that we avoid error propagation, and a more informed decision can be made when postponing the decision whether to normalize to the ranking step.

For the generation step we use a combination of a traditional spelling correction approach (Aspell) with word embeddings, which generates candidates which occur in a similar context as the original word. On top of these, we use a lookup list generated from the training data, a module to complete shortened words (WORD.\*) and a module which splits words.

For the ranking of these candidates, we exploit a wide variety of features, partly originating from the ranking modules. These features are combined in a random forest classifier which predicts the probability that a candidate is the correct normalization. Besides the features from the ranking modules, we use n-gram probabilities from non-standard data as well as standard data. On top of these, we add some binary features indicating whether 1) the candidate is present in a standard vocabulary 2) the character order of the candidate is the similar to the original word 3) the candidate contains any alphabetical characters. Because the original word is a ‘special’ candidate, we

also copy features from the original word to every normalization candidate.

The proposed model will be intrinsically evaluated in the following chapter. This model will be extrinsically evaluated for POS-tagging (Chapter 6), constituency parsing (Chapter 7) and dependency parsing (Chapter 8) in the following chapters.



## Chapter 5

# Evaluation Of MoNoise

In the previous chapter we introduced MoNoise; a modular approach to normalization, consisting of two parts; candidate generation and candidate ranking. This chapter is devoted to the evaluation of MoNoise.

There is no clear consensus on which evaluation metric to use for the normalization task. For almost each of the corpora we use, a different metric is used as default. We start this chapter with an overview of these different evaluation metrics. Unfortunately, none of these evaluation metrics is normalized for the number of required normalizations, making cross-corpus comparison difficult to interpret. For this reason, we introduce a new evaluation metric: error reduction rate.

In Section 5.2 we use error reduction rate to evaluate how well MoNoise performs on the test datasets. In addition, we compare the performance of MoNoise to the state-of-the-art for a variety of benchmarks, using the evaluation metrics which are the standard for each dataset.

To gain more insights in the type of mistakes MoNoise makes, we evaluate performance per type of normalization correction in Section 5.3, using the taxonomy proposed in Section 2.3. In addition to testing the complete system on each category, we also compare the most important generation modules, to compare which modules perform best for which categories.

As explained in the previous chapter, MoNoise consists of two parts: candidate generation and candidate ranking. To inspect in which of these has the most potential for improvement, we evaluate these two parts separately in Section 5.4. In this section, we also examine the effect of using a separate

error detection step.

Finally, in Section 5.5 we examine what happens when we run our normalization model on texts containing fewer anomalies. Since in a realistic setting it is not always known in advance how non-standard a text is, the model should be robust, and not over-normalize on this type of data.

This chapter is based on:

Rob van der Goot and Gertjan van Noord. MoNoise: Modeling noise using a modular normalization system. *Computational Linguistics in the Netherlands Journal*, 7:129–144, December 2017a

Compared to the original paper, this chapter includes a more extensive evaluation. The most important differences are that in this chapter:

- a novel evaluation metric is used.
- we use more evaluation datasets
- we report slightly higher scores, because of a newer version of MoNoise
- we evaluate the effect of a separate error detection step.

We would like to thank Hessel Haagsma, for suggesting the name “error reduction rate” for our new evaluation metric.

The code to reproduce the results of this chapter is available at:  
<https://bitbucket.org/robvandergerg/monoise>

## 5.1 Evaluation Metrics for Normalization

In previous work, a wide variety of evaluation metrics is used for the normalization task. In this section, we first discuss the previously used metrics, and then motivate our main evaluation metric. We will start by discussing the metrics which evaluate beyond the word level. Then, the commonly used F1 score and accuracy are discussed. Finally, we explain and motivate our proposed evaluation metric: error reduction rate.

### 5.1.1 Evaluation beyond the word level

Some previous work on normalization uses word error rate or character error rate for evaluation. These evaluation metrics are based on the Levenshtein distance (Levenshtein, 1966) (see also Section 4.1). For the word error rate, the minimum number of deletions, insertions or substitutions of words which are necessary to convert the output of the normalization model to the reference normalization is the Levenshtein distance. To obtain the word error rate, the Levenshtein distance is divided by the total number of words in the annotated normalization. Character error rate is calculated in the same manner, but with using characters as units.

If splitting and merging is not annotated (only 1-1 replacements are annotated), word error rate is overly complicated, since only substitution occurs. Furthermore, in this case, word error rate is similar to 1-accuracy. For datasets which do include 1-N and/or 1-N replacements, this metric gives extra weight to these specific cases, whereas it is questionable if these cases are actually more important.

Character error rate is arguably a better metric. However, in this case, replacements like ‘lol’ $\mapsto$ ‘laughing out loud’ and ‘neb’ $\mapsto$ ‘nebraska’ are being weighted much heavier compared to the correction of closer words. However, these are not necessarily more important.

Previous work using machine translation methods for normalization makes use of BLEU (Papineni et al., 2002) for evaluation. BLEU calculates the precision of n-grams on the word level of varying sizes. Ljubešić et al. (2016) have shown that BLEU correlates almost perfectly with character error rate for the normalization task. In our opinion, BLEU is needlessly complex for a task in which the word order never changes.

### 5.1.2 F1 score

The F1 score is the harmonic mean (Rijsbergen, 1979) between precision and recall. Below we first introduce some definitions, which are then used to explain precision, recall, F1 score and accuracy in more detail. A schematic overview of the used classes is shown in Figure 5.1.

True negative ( $TN$ ) = Annotators did not normalize, system did not normalize

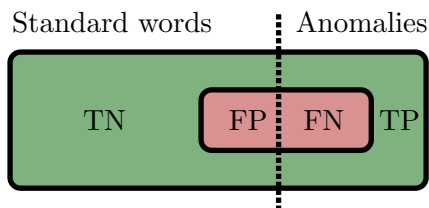


Figure 5.1: Schematic overview of the evaluation categories.

False positive ( $FP$ ) = Annotators did not normalize, system normalized  
 False negative ( $FN$ ) = Annotators normalized, but system did not find the correct normalization. This could be because it kept the original word, or proposed a wrong candidate.  
 True positive ( $TP$ ) = Annotators normalized, systems normalized correctly

## Precision

Out of all replacements made by the normalization model, how many are correct?

$$Precision = \frac{TP}{TP + FP} \quad (5.1)$$

Note here that this interpretation slightly differs from previous work. When a model normalizes an anomaly to the wrong word, it is not accounted for in the precision. These cases are accounted for in the recall (we classify them as  $FN$ ). In some previous work, the model was penalized for these cases in both the precision ( $FP$ ) and the recall ( $FN$ ). A closer look at the evaluation of the shared task on lexical normalization held at WNUT15 (Baldwin et al., 2015a) reveals that they evaluated in this manner. This can be considered to be incorrect, since one replacement counts double, whereas the decision whether to normalize is actually correct. However, this is an easy mistake to make. Actually, we made the same mistake in van der Goot and van Noord (2017a), and Reynaert (2008) report that the same mistake was made in previous work.

## Recall

Out of all anomalies, how many are correctly normalized by the normalization model?

$$Recall = \frac{TP}{TP + FN} \quad (5.2)$$

## F1

The F1 score is the harmonic mean of precision and recall. The harmonic mean is an average between two values which rewards performance when both values are closer to each other. The F1 score is defined as:

$$F_1 = 2 * \frac{precision * recall}{precision + recall} \quad (5.3)$$

Some early work on normalization used F1 score while assuming gold error detection. This is an odd metric, since precision, recall, and F1 are all the same. Furthermore, this metrics is then also equal to the accuracy over all the anomalies, which is more straightforward to calculate and interpret.

Our main reasons we do not use of F1 score for evaluating normalization, is that it is not directly interpretable, and unnecessarily complex. The complexity is proven by the fact that it has been used wrongly multiple times in previous work (as explained on page 76).

### 5.1.3 Accuracy

Early work on normalization assumed gold error detection. So they focused only on the task of finding the correct normalization for anomalies. In this simpler setup, accuracy on the anomalies was commonly used to evaluate. However, accuracy can also be used to evaluate the complete normalization task. Then it represents the percentage of correct words in the normalized output:

$$Accuracy = \frac{TP + TN}{TP + FP + TN + FN} \quad (5.4)$$

The numerator ( $TP + TN$ ) is the number of correctly predicted words. The denominator is the total numbers of words in the corpus. If we consider

a baseline system which always returns the original word,  $FP$  and  $TP$  will always be 0, and the accuracy will be equal to the ratio of words which do not need normalization.

#### 5.1.4 Error Reduction Rate (ERR)

One downside of accuracy is that it is hard to compare across datasets, since different numbers of candidates are in need of normalization. An accuracy of 93% might be a very good score on one dataset, whereas on another dataset a normalization model which scores 93% might be completely useless. Hence, we propose a new metric; the error reduction rate. This metric is accuracy normalized for the number of words in need of normalization. It is similar in spirit to measures of inter-annotator agreement like Cohen’s Kappa (Cohen, 1968), in that it takes the difficulty of the task into account. The error reduction rate can be calculated using the accuracy of a normalization system ( $Accuracy_{system}$ ) and the accuracy of a baseline system which always returns the original word ( $Accuracy_{baseline}$ ):

$$ERR = \frac{Accuracy_{system} - Accuracy_{baseline}}{1.0 - Accuracy_{baseline}} \quad (5.5)$$

Using the notation introduced earlier, the error reduction rate can also be calculated as follows:

$$ERR = \frac{TP - FP}{TP + FN} \quad (5.6)$$

This formula is equivalent to the previous formula. The proof that they are equivalent can be found in Appendix A.

The ERR will normally have a value between 0.0 and 1.0. A negative ERR indicates that the system normalizes more words wrongly than correctly. A baseline which always keeps the original word scores exactly 0.0, and a perfect system will score 1.0.

ERR has multiple advantages compared to the previously discussed metrics:

- Easily compare across multiple corpora: because ERR normalizes for the percentage of words in need of normalization, the results can be interpreted similarly.

- Easily interpretable: ERR directly shows the percentage of the problem which is solved. A negative ERR indicates that a system makes more mistakes than correct normalizations.
- Evaluates the complete normalization task: this metric includes the effect of the error detection step.

We consider ERR as main evaluation, and use it to compare the different models. However, ERR fails to distinguish between *FP* and *FN*; it does not tell us whether the system normalizes too aggressively, or is too careful. This is why we will also use precision and recall.

In the following section we will use the ERR to evaluate the performance of MoNoise on the test datasets. Additionally, we will use benchmark-specific metrics to compare to the state-of-art systems for the different datasets.

### 5.1.5 Area Under the ROC Curve

One member of the reading committee suggested Area Under the ROC Curve as an alternative to ERR. The Area under the ROC Curve is not very straightforward to obtain for normalization, since we need to plot a line by changing a threshold, which is much easier for binary classification. However, there is a clear relation between the ROC curve and ERR. The distance between the system and the plot of the the baseline in the ROC space can be used as evaluation metric, and has a very high correlation to ERR. This distance is also called Youden’s J statistic (Youden, 1950) or informedness. The relation between these two metrics is shown in more detail in Appendix B.

## 5.2 Test Data

In this section we evaluate MoNoise on the test data. We do this twofold: first using our preferred evaluation metric: error reduction rate. Additionally, we provide a comparison with the state-of-the-art of a variety of benchmarks for normalization, each using a different metric for evaluation. For completeness, we include all evaluation metrics for all the corpora in Appendix C. In this section, we always train on the concatenation of the training and

Corpus	ERR	Precision	Recall
GhentNorm	44.62	86.84	50.77
TweetNorm	35.86	90.05	37.09
LexNorm1.2	60.61	78.03	79.12
LexNorm2015	76.15	91.98	80.58
Janes-Norm	67.15	89.62	70.81
ReLDI-hr	51.73	92.17	54.23
ReLDI-sr	57.48	86.43	60.78

Table 5.1: Results of MoNoise on the test data.

development data, and report results on the test split. Because the inclusion of the split module (Section 4.3.2) does not lead to a higher performance (Section 5.4.1) and annotation for splitting is not included in most of our corpora (Section 2.2.1), this is the only module that we disabled for the runs on the test data.

### 5.2.1 Error Reduction Rate per Corpus

Table 5.1 shows the precision, recall and ERR for the different corpora on the test data. Note that the corpora are ordered by size, as the amount of training data could potentially affect their performance (we test this in Section 5.4.3). Overall, precision is higher compared to recall, which is arguably a desirable result for this task because we do not want to replace ‘correct’ words. The ERR shows a bit lower scores compared to the recall, this is expected since it takes into account one more error type (FP). For all corpora, except the two smallest, MoNoise scores an ERR above 50%. This shows that it solves more than half of the normalization issues, even when taking wrong normalizations into account.

The effect of the size of the training data is somewhat visible in the results; except for the drop in performance for the South Slavic languages (Janes-Norm, ReLDI-hr, and ReLDI-sr). This can partly be explained by the smaller and more distant raw data we used (see section 2.2.2). We investigate the effect of the size of the training data in more detail in Section 5.4.3.

On LexNorm2015 we achieve the highest score; this can be due to several

Corpus	Prev. state-of-the-art	Metric	Prev.	MoNoise
LexNorm1.2	Li and Liu (2015)	Accuracy	87.58	87.63
LexNorm2015	Jin (2015)	F1	84.21	86.61
GhentNorm	Schulz et al. (2016)	WER	3.2	1.36
TweetNorm	Porta and Sancho (2013)	OOV-Precision	63.4	70.57
Janes L1	Ljubešić et al. (2016)	CER	0.38	0.55
Janes L3	Ljubešić et al. (2016)	CER	1.58	2.38

Table 5.2: Results on test data compared to the previous state-of-the-art.

factors:

- Size of training data
- The raw data used to train the word embeddings: similarity to the test data as well as the quantity are important factors.
- Annotation guidelines: mainly the inclusion of phrasal abbreviations adds a lot of easy normalizations

### 5.2.2 Comparison with Previous work

In this section we will compare our approach to previous work. Table 5.2 lists all benchmarks for which previous results have been reported. Below, we discuss the approaches of the previous state-of-the-art, and compare it to our own performance.

Li and Liu (2015) built an ensemble model of six different normalization models, including a spell checker, machine translation models, and lookup lists. Each model proposes one candidate, which are re-ranked in a Viterbi encoding based on the candidates and their potential POS tags. Li and Liu (2015) assume gold error detection, like most previous work on this dataset. Therefore, the reported accuracy is on only the anomalies. Li and Liu (2015) uses a slightly different version of the LexNorm1.2 corpus, where some of the words are removed; MoNoise reaches an accuracy of 88.26 on this data<sup>1</sup>.

<sup>1</sup>this version can be downloaded from: [http://www.hlt.utdallas.edu/~chenli/normalization\\_pos/test\\_set\\_2.txt](http://www.hlt.utdallas.edu/~chenli/normalization_pos/test_set_2.txt)

The model proposed by Jin (2015) has some similarities to our proposed model. They use a lookup list and a ‘split module’ combined with a novel distance metric to find spelling variations for candidate generation. Features from these modules are then combined with confidence scores of a POS tagger and combined in a random forest classifier to get a final prediction. The main difference with our model is that we exploit raw data for our generation as well as our features. This leads to a more generic model, which can more easily adapt to other datasets. In the table we used the F1 score as implemented for the shared task (see also Section 5.1), to compare with previous work.

The Spanish dataset originates from a shared task, for which a wide variety of approaches is tested (Alegria et al., 2013). Due to the small size of the training data, more rule-based approaches were used<sup>2</sup>, which are tuned towards Spanish. For the shared task, only out of vocabulary words are considered for normalization (but they are not all normalized). Our results in the table are obtained using the same heuristic.

Schulz et al. (2016) built a multi-modular model, in which each module accounts for different normalization problems, including machine translation modules, a lookup list, and a spell checker. They also report improved results for extrinsic evaluations on three tasks: POS tagging, lemmatization and named entity recognition. The results on this dataset are not directly comparable, since Schulz et al. (2016) do not assume gold tokenization and have different dev-test splits.

For the South Slavic languages, we are only aware of results published for Slovene; Ljubešić et al. (2016) experiment with token and character level machine translation, and show that character level information is especially beneficial for text with a high level of non-standardness. Additionally, they show that using raw data in a semi-supervised setup can improve the performance of a normalization model. They split their dataset in a canonical subset (only 3% is normalized, L1 in the table) and another subset which is much noisier (17% is normalized, L3 in the table). This model performs slightly better compared to MoNoise. After a closer inspection of the output of both models for the development data, we<sup>3</sup> saw that the main

---

<sup>2</sup>Unfortunately, most shared task papers were written in Spanish, so the details were hard to grasp.

<sup>3</sup>‘we’ includes the first author of Ljubešić et al. (2016) in this case

difference is that MoNoise cannot handle uppercase letters with diacritics. Beyond this difference, the performance of our models is remarkably similar, even though the approaches are completely different.

Overall, our proposed model reaches a new state-of-the-art performance for most benchmarks, moreover, all previous work is focused on one dataset only. In contrast, we did not tune MoNoise towards specific corpora or evaluation metrics, which would probably lead to slightly higher scores.

In the following sections we will evaluate the separate parts of our model in more detail.

### 5.3 Type of Errors

In this section, we evaluate MoNoise in more detail using the taxonomy proposed in Section 2.3.2. Firstly, we test the performance of the entire model on the different categories. Next, we test the focus of the main generation modules. Our hypothesis is that Aspell performs better on unintended anomalies, as its focus is on spelling correction, whereas the lookup list should perform better on the intended anomalies since these contain less variety. Word embeddings are probably also more effective on the intended anomalies, because these are more frequent in the raw data, which leads to higher quality vectors.

#### 5.3.1 Performance per Category

Recall from Section 2.3.3 that we annotated the entire training split of LexNorm2015 with the normalization categories. Hence, we run MoNoise in a 10-fold cross validation setup to get predictions for the entire training set. For each category, we plotted the number of correctly normalized pairs compared to the total number of replacement pairs (Figure 5.2). Note that besides these errors, 198 words that were not in need of normalization are still replaced by our model, so these cases still account for a large part of the errors.

Looking at which categories are most often normalized correctly, we can conclude that none of the categories is completely solved. Nevertheless, for some categories the model normalizes up to 95% correctly. On the larger categories, the model performs better, this is an effect of having a few

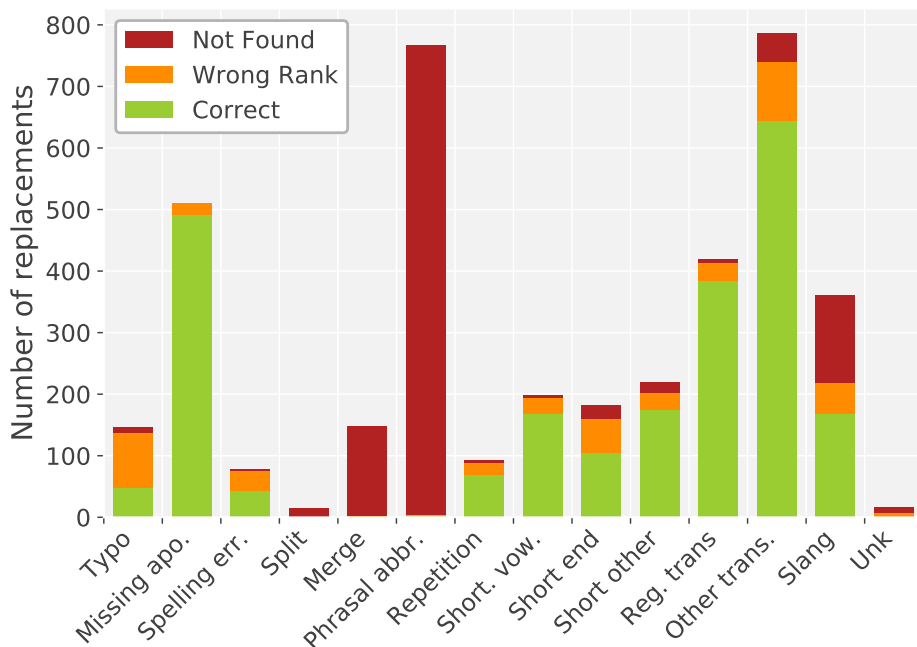


Figure 5.2: Performance of MoNoise on the different categories in a 10-fold experiment on the training part of LexNorm2015.

very common replacements, which are always done correctly. As expected, MERGE and SLANG are difficult categories for the system. However, two other bad performing categories are somewhat surprising: TYPO and SHORT END. The pairs in these categories usually have a short edit distance compared to the original word. A closer look at the mistakes made on these categories shows that these are mainly on short words. These words often lead to long candidate list, mainly because of the ASPELL and the WORD.\* module. For example for the replacement ‘pre’ $\mapsto$ preorder, the WORD.\* module generated 753 candidates. Unsurprisingly, on the two categories which were not handled by our model (SPLIT and UNK) zero instances are normalized similar to the annotation.

For most categories, the ranking of the candidates is the bottleneck, shown by the large orange bars in Figure 5.2, except for the PHRASAL

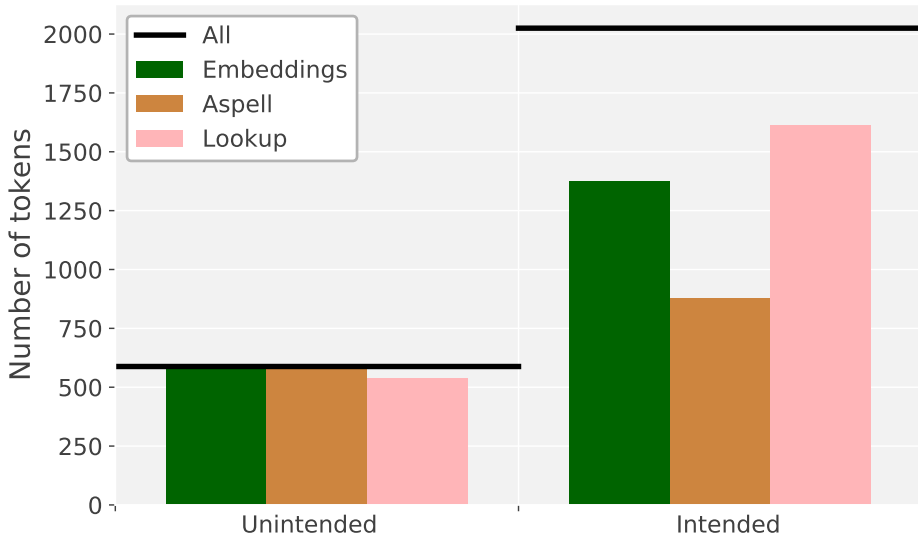


Figure 5.3: Recall achieved by the three best performing generation modules in isolation. COMBINATION shows the combined recall.

ABBREVIATIONS and SLANG category. For the PHRASAL ABBREVIATION category, this is due to unknown abbreviations which were not seen in the training data as well as the need of two subsequent normalization actions (‘loool’→‘laughing out loud’). For the SLANG category, this is mainly due to quite distant replacements, in which the edit distance between the original word and the correct normalization is relatively high.

### 5.3.2 Performance per Module

In the introduction of this section, we hypothesized that Aspell would perform best for the unintended normalization replacements, whereas the lookup list would most likely perform well for the intended replacements. We test this only on a higher level of the taxonomy (intended vs. unintended), to simplify interpretation. The recall of our main generation modules is shown in Figure 5.3. We use recall as metric here, because the modules do not rank very well on their own. As expected, Aspell does indeed perform very well on the unintended category, and reaches a recall remarkably close

to the full model. The word embeddings probably suffer from data sparsity here, since a mistyped word has to occur multiple times in our raw data to get a good quality vector.

On the intended anomalies, the word embeddings perform better than Aspell. This is because these categories are more often closed classes, and occur more frequently. Exceptions are the REPETITION and REGULAR TRANSFORMATION categories. The lookup module generates a lot of correct candidates for the intended categories as well, but a large portion of these are due to frequent phrasal abbreviations.

## 5.4 Evaluation of Sub-tasks

In this section, we will take a closer look at the performance of the different parts of MoNoise; the generation step and the ranking step. Additionally, we plot a learning curve, to see how much training data is required to train a normalization model. Finally, we test the effect of having a separate error detection step.

### 5.4.1 Candidate Generation

In this section we evaluate the generation modules. The performance of the generation modules is very important for the final normalization model, since the recall of this step is responsible for an upper bound of the performance of the final model. We evaluate the generation threefold; first, the performance of each module is tested. This is done by evaluating the performance of the module in isolation as well as in an ablation experiment to test the number of unique candidates each module contributes to the complete model. Second, we examine how many candidates each module generates, because small candidate lists are preferable. After this qualitative analysis, we will look at some examples of normalizations which are not generated by any of our modules.

#### Performance per Module

The recall of the generation modules in isolation is plotted in Figure 5.4. The best modules are Aspell, word embeddings and the lookup module.

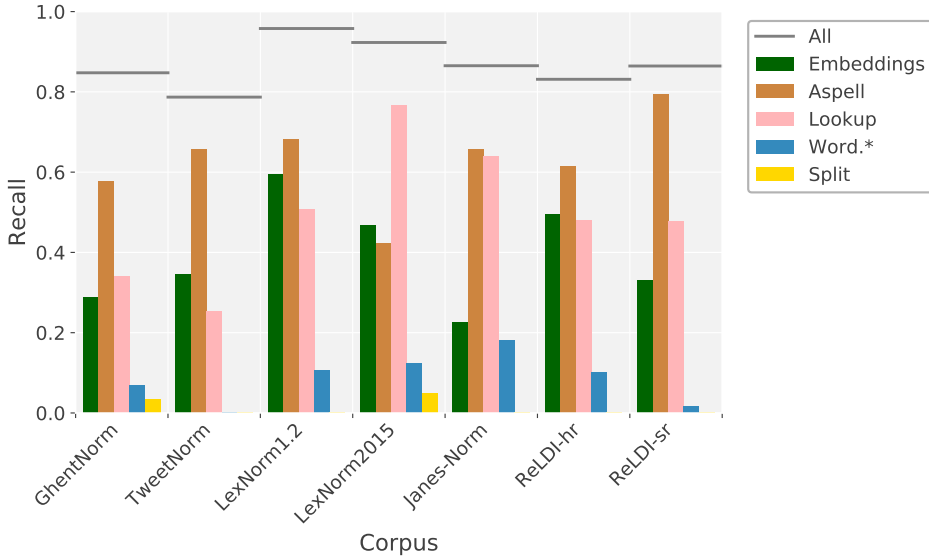


Figure 5.4: Recall of generation modules on the development corpora. ‘all’ is the combination of all modules.

Figure 5.4 also shows that the recall of ‘all’ modules is quite a lot higher compared to the single modules, which indicates that they are complementary. The lookup module performs especially well on the LexNorm2015 corpus. This is due to a couple of correction pairs which occur very frequently (u, lol, idk, bro). On all other corpora, the Aspell module performs best. The split module can only generate correct candidates for corpora that contain 1-N word replacements. However, even for these corpora it generates only a few correct candidates, just like the word.\* module. Performance differences between corpora can be explained by:

- Language: the different languages evolve in different ways online. For example, in Dutch it is common to merge the pronoun ‘ik’ (en: ‘I’) to a verb. e.g. ‘kheb’, which normalizes to ‘ik heb’, meaning ‘I have’.
- Annotation guidelines and annotators: especially the decision whether to include phrasal abbreviations (‘lol’→‘laughing out loud’) has a large influence on the importance of the lookup list.

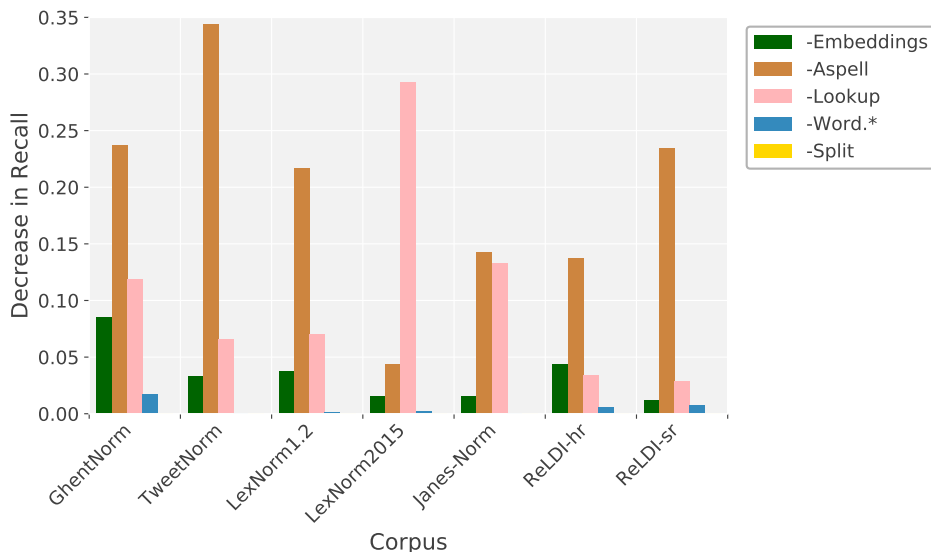


Figure 5.5: Results of ablation test. Decrease in recall on the development data when excluding a module is plotted; so higher scores indicate more important modules.

- Size training data: Our largest dataset is seven times larger compared to our smallest dataset. The effect of the size of the training data is evaluated in more detail in Section 5.4.3.
- Size and domain of the raw data (see Section 2.2.2) which is used to train the word embeddings.

To measure the influence of each generation module in the final model, we ran an ablation experiment. Here, the performance degradation when a module is excluded from the model is measured. This is plotted in Figure 5.5. Similar to the previous experiment, the most important module is ASPELL, followed by LOOKUP and EMBEDDINGS. However, the word embeddings seem to be much less important compared to Figure 5.4. This means that this module provides a lot of candidates which are also generated by other modules. Presumably, this is because the candidates generated by the word embeddings have overlap with both Aspell and the lookup list.

Module	Avg. Candidates
Embeddings	35.0
Aspell	22.5
Lookup	0.668
Word.*	50.4
Split	0.325

Table 5.3: The average number of candidates per word for each module, averaged over all corpora.

### Number of Candidates per Module

Recall of the modules are not the only important criteria, a module which returns all words of the dictionary would score very well. Small candidate lists are preferable for two reasons: it simplifies the process of finding the correct candidate and makes the model more efficient.

The average number of candidates for all corpora per module is reported in Table 5.3. Most of the candidates are generated by the EMBEDDINGS, ASPELL, and WORD.\* modules. The first two of these are also important for the recall of the final model, whereas the WORD.\* module does not generate many correct candidates. As expected, the LOOKUP module is the most effective, it generates a small number of candidates (Table 5.3) from which many are correct normalizations (Figure 5.4).

### Examples of Missing Candidates

The SPLIT module does not provide any unique candidates for our development sets. This module only overlaps with the lookup list, since this is the only module that generates 1-N replacements. Surprisingly, the WORD.\* module provides relatively many unique candidates for the GhentNorm corpus, this is due to a smaller lookup list (the training data is smaller) and this style of abbreviating is more common in this dataset.

To analyze the weaknesses of our generation modules in more detail, we will discuss some of the replacements which are not found by our generation modules. Table 5.4 shows five normalization replacements from our Dutch and English development sets for which the correct candidate was not found

GhentNorm		LexNorm1.2		LexNorm2015	
neeneenee	nee nee nee	sowi	sorry	trynna	trying to
zijt	bent	neb	nebraska	skepta	sunglasses
bij	die	mo'd	mowed	satnite	saturday night
bwoaja	ja	sumwer	somewhere	tbf	to be fair
jana's	jana 's	thuur	thursday	wada	water

Table 5.4: Examples of missing normalization replacements for our English and Dutch development sets.

by our generation modules.

Some of these instances are not found because they are odd annotations. Like ‘zijt’ $\mapsto$ ‘bent’ is also a translation from Flemish Dutch to standard Dutch, and the normalization of ‘skepta’ $\mapsto$ ‘sunglasses’ is doubtful. Another source of mistakes is 1-N replacements, especially when the normalized word is not a direct split of the source. For example for ‘trynna’, which needs two normalization actions: ‘trynna’ $\mapsto$ ‘tryna’ $\mapsto$ ‘trying to’. Besides these cases, there are also cases where the normalized word is a phonetic transformation of the original but the distance is too high, like ‘bwoaja’, ‘sowi’ and ‘wada’. The correct word ‘nebraska’ was not found because the Aspell dictionary only contains this word with proper capitalization. In other words, this is an effect of a mismatch in the annotation of the dataset and the Aspell dictionary. This can be considered a bug, and can be fixed by simply lowercasing the whole Aspell dictionary.

### 5.4.2 Candidate Ranking

In this section we will evaluate the candidate ranking, which is the final step of our normalization model. We will first evaluate the importance of different feature groups with respect to the model. Secondly, we evaluate beyond the top-normalization candidate by evaluating the recall for the top-N candidates.

#### Feature Analysis

To test the effect of our feature groups on the final model, we perform an ablation experiment and average the scores over all corpora. Results are

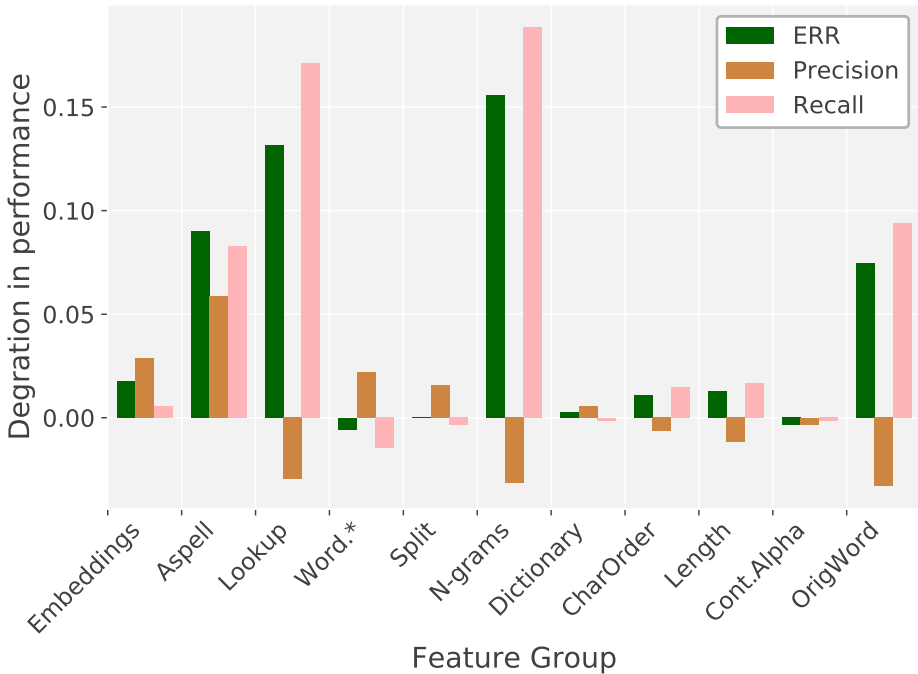


Figure 5.6: Degradation in performance for all metrics when excluding a feature group. A higher score means a more important feature group. The plotted metrics are the average over all our development splits.

plotted in Figure 5.6. The plotted values are the degradation in performance when excluding a feature group. Thus, it represents the amount of performance which a feature group is responsible for in the final model.

The results show a correlation between the metrics; when the loss in recall decreases more, loss in precision tends to decrease less, or even increases. This is an effect of the number of words which are normalized, when the model is more aggressive it finds more necessary replacements (recall), but also normalizes some standard words (precision). A decrease in precision indicates that a higher percentage of normalized words are wrongfully normalized. However, for our feature groups this is always combined with an increase in recall which is larger than the decrease in precision, which thus leads to a higher ERR.

For the complete system, the most important feature groups are the lookup list and the n-gram features, closely followed by Aspell. The length features are surprisingly effective. Word embeddings are not as important as Aspell and the lookup module, which confirms our findings in section 5.4.1. The least important feature groups are `WORD.*`, `SPLIT`, `CONTAINSALPHA`, and `DICTIONARY`. For the first two, this comes as no surprise, as they were also not very beneficial for the generation step (Section 5.4.1). `CONTAINSALPHA` is a very simple feature, which is corpus-specific, whereas the `DICTIONARY` feature is probably made irrelevant by also using n-gram features.

## Top-N

Since our ranking step scores all candidates (Section 4.4), it is straightforward to output a list of top-N ranked candidates. This can be used to gain more insights into the effectiveness and difficulties of the ranking step. It can also be useful for a natural language processing system to exploit the top-N candidates to avoid error propagation.

Figure 5.7 shows the recall of the top-N candidates. It becomes clear that most of the mistakes made by the classifier are actually among the first and the second candidate, i.e. beyond the second candidate, only a few correct candidates are found. Manual inspection reveals that in most of these cases, the model decided to keep the original word where normalization is necessary (approximately two thirds). Another major source of errors is when the original word should be kept, but is ranked second (approximately one third). From these results, we can conclude that the task of error detection is a bottleneck for this model.

Beyond the second candidate, the improvements converge. The recall of the upper bound is still a bit higher compared to the ninth candidate, which means that a small amount of candidates is ranked very low. These are often short words. This is a result of confusion due to large candidate lists, which in turn is due to the fact that short words have relatively many words with a small edit distance. All of our corpora show a similar trend for this experiment.

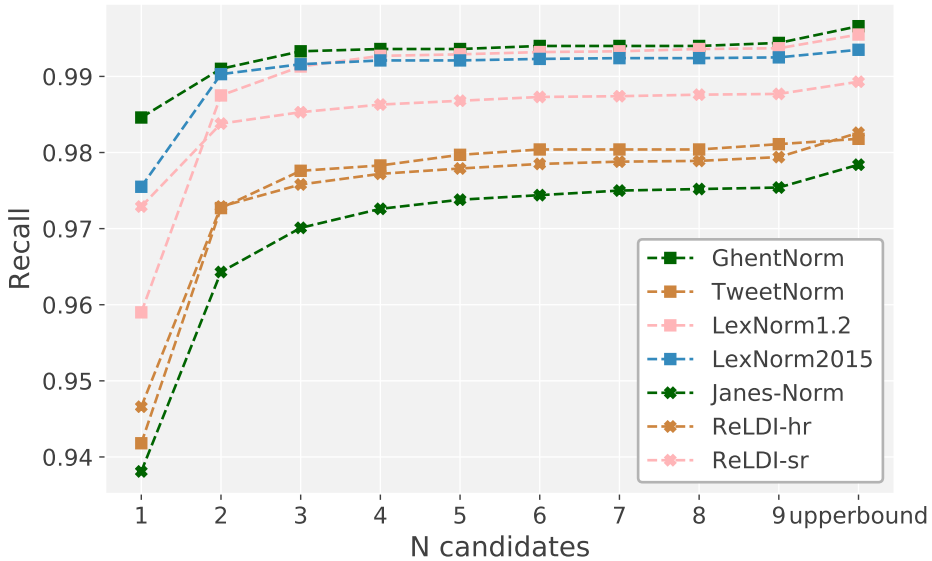


Figure 5.7: Recall of the top-N candidates of the normalization model for our dev-sets.

### 5.4.3 Amount of Training Data

To test how much annotated data is required to train a normalization model, we tested the effect of the size of the training data on the performance for all of our development sets. The results are plotted in Figure 5.8.

For all corpora, the biggest gains in ERR are already gained by using only 5,000 words for training. After that, the improvement converges, even though there is still a slight upward trend visible after adding more data. From this graph, we can conclude that for the two smaller corpora we do not have enough training data. When using very small amounts of training data, the differences between the corpora are already large, indicating that the datasets have a different difficulty. This can be due to differences between the languages, but also other factors have influence. For example, different annotators, annotation guidelines, or data selection.

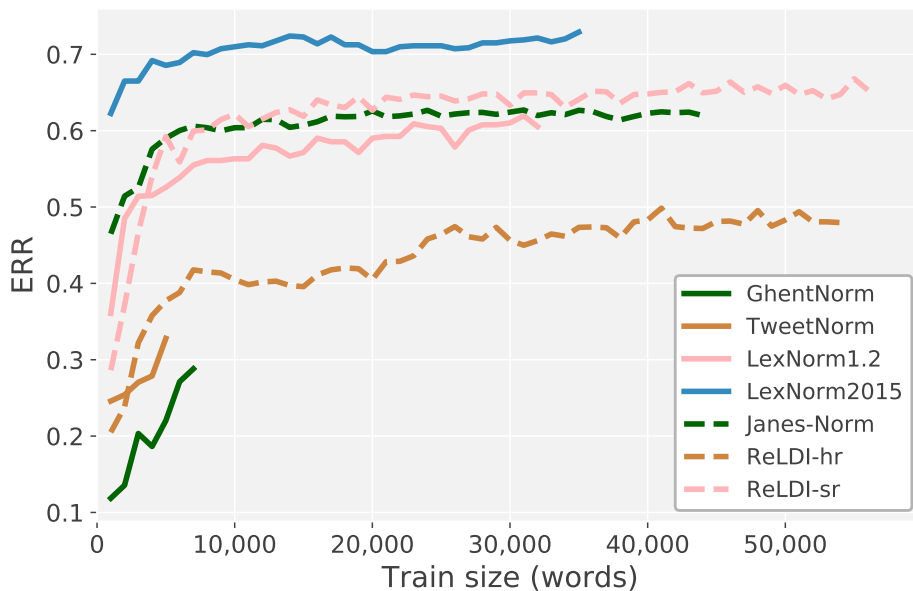


Figure 5.8: Learning curves for the different corpora

#### 5.4.4 Separate Error Detection

Most of the early work on normalization assumed gold error detection, and thus only focused on finding the correct replacements. However, more recently, the full task of normalization is often benchmarked. Since error detection is also part of the task, some previous work uses a separate error detection step, similar to traditional spelling correction systems. In this setup, only for the words detected as “error”, replacement candidates are generated. The motivation for using a separate error detection is twofold: it is more efficient to only consider erroneous words and it can prevent over-correction.

To the best of our knowledge, Schulz et al. (2016) is the only work on normalization actually testing the effect of having this separate detection step. They show that filtering which words to normalize leads to a small performance improvement. However, their generation modules are quite different compared to ours, leading to much smaller candidate lists. Further-

more, they rank candidates using a language model instead of a classifier. In this section we will explore a variety of approaches to improve upon the error detection task in our setup.

Previously, we assumed that postponing the decision whether to normalize leads to a more informed decision (Section 4.2). However, in Section 5.4.2 it became apparent that the main weakness of the normalization model is that it does not know when to normalize: it often keeps the original word and ranks the correct candidate second, and it sometimes over-normalizes words which are not in need of normalization.

We implemented two approaches to test the effect of using a separate error detection step on performance as well as runtime. Firstly, we use the traditional method of detecting anomalies using a dictionary. In this setting, we only allow the model to normalize words which are not present in the Aspell dictionaries. Secondly, we attempt to automatically detect anomalies by training a separate classifier for the error detection task.

### **Error Detection Based on Vocabulary**

In traditional spelling correction systems, only words which are not included in a dictionary were considered for correction. In this subsection, we examine whether this heuristic is beneficial for MoNoise. We use the Aspell dictionaries to filter out in-vocabulary words, which are then not considered for normalization.

The effect of this method on the ERR is shown in Table 5.5. For most datasets, this has a negative effect on performance, which can be explained by the fact that the information whether a word is present in the Aspell dictionary was already represented by a feature. The only dataset on which error detection is beneficial is the Spanish dataset (TweetNorm). This is due to the annotation scheme; for this dataset, annotators were only allowed to normalize out-of-vocabulary words. In the following section, we examine whether a more sophisticated approach to error detection can be beneficial in our setup.

Considering the runtime, we can see that the error detection has a large effect; a speed-up of a factor of 5-10 is shown in the table. For our datasets, approximately 25-45% of the words do not occur in the Aspell dictionary. The runtime is thus also faster per word which is considered for normalization. Closer inspection revealed that this is the case because the

Corpus	MoNoise		+errDet	
	ERR	words/ second	ERR	words/ second
GhentNorm	30.51	23	30.51	141
TweetNorm*	31.97	35	32.79	59
LexNorm1.2	61.24	33	48.83	224
LexNorm2015	71.12	40	54.69	173
Janes-Norm	61.37	20	49.96	110
ReLDI-hr	48.31	37	40.03	225
ReLDI-sr	65.60	29	33.69	254

Table 5.5: The effect of using a vocabulary to filter words to consider for normalization on the ERR (+errDet). \*the comparison is not completely fair on this dataset, since the annotation guidelines enforced that only out of vocabulary words are normalized.

classifier is the slowest part of MoNoise. The words which are skipped when error detection is used, had larger candidate lists, thus leading to a slower throughput.

In most cases, the full model should probably be the preferred option, since the performance difference is rather large. However, the speed-up is substantial, and when processing huge amounts of texts, which is readily available for the social media domains, enabling the error detection can be considered.

### Automatic Error Detection

A more elaborate method of doing error detection is to train a separate classifier which optimizes on this task. We test if the features used by MoNoise can be used more effectively if we divide normalization into multiple sub-tasks. In this setup, we generate features only for the original word, and run a binary classifier which predicts whether it is in need of normalization. We discard features from the generation step which are always zero for the original word. The feature groups which are left are LOOKUP-LIST, N-GRAMS, DICTIONARY, LENGTH and CONTAINSALPHA. We use the confidence scores

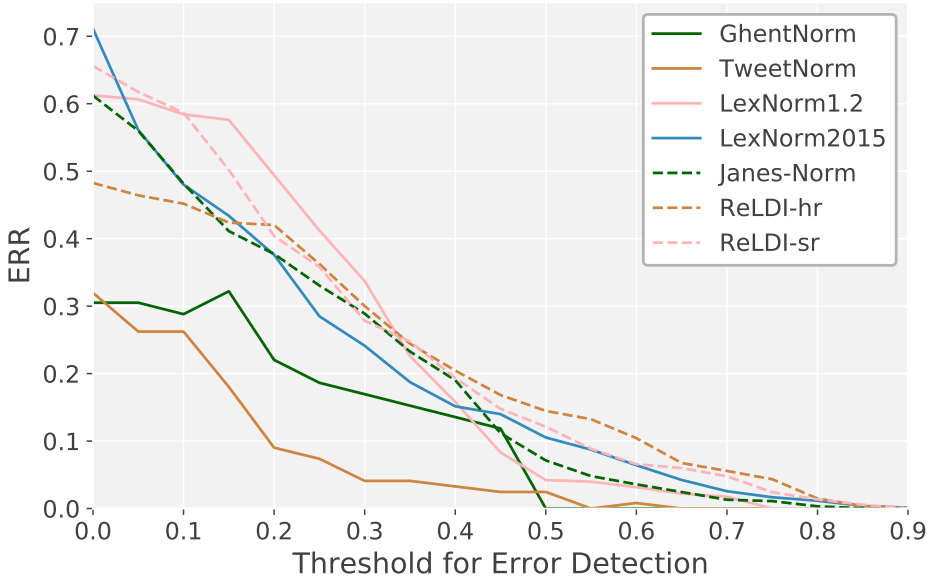


Figure 5.9: The effect of tuning the extra error detection step on the ERR.

of the classifier so that we can tune how many words the error detection filters out. After this first step, we let MoNoise generate candidates only for the words which have a confidence score below a particular threshold. Note that this model can still decide not to normalize.

The results are plotted in Figure 5.9. With a threshold of 0.0, all words are considered for normalization, which is practically the same as not having an error detection step. With a high threshold ( $>0.9$ ) no words are considered for normalization, so the ERR is equal to 0.

For almost all corpora, the best performance is obtained with a threshold of 0.0. This means that, similar to the previous method, error detection is not beneficial for performance. Only for the Dutch corpus (GhentNorm), there is a small increase in performance visible when using low thresholds, this is probably an effect of not having enough training data.

We also plotted the number of words per second with respect to the threshold, which showed no big gains for increasing the threshold from 0.0 to 0.9. The gain from considering fewer words for normalization candidates

is approximately equal to the cost of running an additional classifier.

It should be taken into account that we experiment with the same features as used for the normalization model. So no big improvements were to be expected. However, if another feature set performs better for the error detection task, this set can also be included in the model directly.

## 5.5 Robustness

In the previous chapter we have seen that the model works well on the annotated datasets. These datasets are filtered to contain a certain level of non-standard language. However, in many scenarios it is not known in advance how non-standard the input is. Even on Twitter, many of the utterances are canonical. Because it is undesirable to have a model which is sensitive to over-normalize on this type of data, we analyze the performance of MoNoise on more standard data in this section.

To this end, we use the data from two of the treebanks described in Section 3.3, the English Web Treebank (EWT) and the Wall Street Journal part of the Penn Treebank (WSJ). The WSJ consists of well-edited news texts, and should only contain a very small amount of anomalies. The data from the EWT should be somewhere between the very non-standard data from Twitter and the very clean data from the WSJ: it contains a few anomalies, but also much standard language. We only evaluate robustness for English because manual annotation is required.

We ran our normalization model using the default settings, trained on the LiLiu data (Section 2.2.1). We then compared the output against the original texts. We annotate unique replacement pairs, which we call replacement types, to make the annotation more efficient. This generalization is usually justified, since most words in these datasets do not have different normalizations for different contexts (this only occurs only for seven words). We manually annotated each replacement pair in one of the following four categories (examples of each category are shown in Table 5.6):

- +: A correct normalization replacement
- -: An incorrect normalization replacement
- +-: Not necessary, the original word is replaced for a similar word, e.g.: n't $\rightarrow$ not

+	thru $\mapsto$ through, becuse $\mapsto$ because
-	lease $\mapsto$ least, inn $\mapsto$ in
+ -	tv $\mapsto$ television, ad $\mapsto$ advertisement
Partly	@ $\mapsto$ at, appy $\mapsto$ happy

Table 5.6: Examples of the categories used

- Partly: Replacement pair is correct in some contexts. This category is necessary because we annotate unique replacement pairs, whereas words might require a different normalization based on their contexts.

In Table 5.7 we show the results of our annotation efforts. On the EWT, the model normalizes more correct than incorrect, whereas on the WSJ this is the other way around. The high number of total replacements might seem like a problem at first sight. However, these are mainly due to some very frequent replacements pairs, so a simple lookup list could avoid most of these. For example, the replacement of its $\mapsto$ it’s occurs respectively 155 and 2,289 times in the EWT and WSJ data. During annotation, we also found that many mistakes are made for words shorter than 3 characters. These are often used for proper nouns in news texts, whereas on the social media domain these are usually abbreviations of words. This can probably be circumvented by training a normalization model that also corrects capitalization, which is missing in our training data. The ‘Partly’ category is rather small, justifying our shortcut of annotating unique pairs.

To conclude, the model makes a lot of mistakes on the standard corpora. However, this is mostly due to a few very frequent replacement types. So most mistakes could easily be circumvented by using a list of words which should be ignored during normalization. On the unique replacement pairs, the model still normalizes more correct than wrong for web data, but on news texts the model over-normalizes slightly.

## 5.6 Conclusion

In this chapter we evaluated MoNoise from a variety of angles. First, we compared MoNoise to previous work on a wide variety of benchmarks for multiple languages, showing that it improves upon the state-of-the-art on

	Replacements Types		Total replacements	
	EWT	WSJ	EWT	WSJ
Size (words)	19,671	35,934	204,607	731,678
+	271	57	384	80
-	73	143	446	3,058
+−	8	18	51	348
Partly	7	0	64	0

Table 5.7: Evaluation of normalization replacements on more standard data.

almost all benchmarks.

To gain more insights on the weaknesses of MoNoise we evaluated its performance on the taxonomy of normalization replacement categories proposed in Section 2.3. Interestingly, this evaluation revealed that some of the categories for which the normalization is lexically relatively similar to the original words actually are difficult (TYPO and SHORT END). Furthermore, we confirmed the hypothesis that the word embeddings module is especially useful for intended anomalies, whereas Aspell almost reaches the same recall as all the modules combined for the unintended anomalies.

When testing the separate parts of MoNoise, namely the candidate generation and candidate ranking, we saw that the most promising direction for improvement is in the ranking. More specifically, most mistakes are made on the decision whether to normalize or not. Using a separate error detection step did not lead to a higher performance. Additionally, we showed that MoNoise needs approximately 10,000 annotated words to reach a good performance.

Finally, we tested the robustness of our model on text containing fewer anomalies. On the web data from the English Web Treebank, a small number of correct replacements were found compared to an even smaller amount of erroneous replacements. Conversely, on the news texts from the WSJ, MoNoise makes a lot of wrong replacements. This mainly due to a couple of very frequent mistakes and could easily be solved by incorporating some rules to ignore these words.

In the following chapter we will perform an extrinsic evaluation for the normalization model, by testing its effect on the task of POS tagging.

## Chapter 6

# The Impact of Normalization on POS Tagging

As explained in Section 3.1.1 and Section 3.2.2, most modern parsers expect POS tagged input or include an internal POS tagger, which tags the input before it is parsed. In this chapter, we will evaluate MoNoise extrinsically, by testing the effect of normalization on the task of POS tagging.

Performance for POS tagging on news text has been higher than 97% for a while now (Toutanova et al., 2003). For this domain, the remaining problems are mainly due to inconsistencies in annotation (Manning, 2011). However, for domains containing more noisy and diverse language use, like Twitter, performance is much lower. Furthermore, for Twitter, POS corpora are usually created using an idiosyncratic tagset, so very little training data is available.

To illustrate why normalization might help for the task of POS tagging, consider the following tweet: “new pix comming tomoroe”. An off-the-shelf system such as the Stanford POS tagger (Toutanova et al., 2003) makes several mistakes on the raw input, e.g., the verb ‘comming’ as well as the plural noun ‘pix’ are tagged as a singular noun. Instead, its normalized form is analyzed correctly, as shown in Figure 6.1.

We see at least two issues with the previous work on the assessment of normalization as a successful step in POS tagging non-canonical text. Firstly, normalization experiments are usually carried out assuming that the tokens to be normalized are already detected (Li and Liu, 2015). Thus

---

new	pix	comming	tomoroe
JJ	NN	NN	NNS
new	pictures	coming	tomorrow
JJ	NNS	VBG	NN

Figure 6.1: Example tweet from the test data, raw and normalized form, tagged with Stanford NLP.

little is known on how normalization impacts tagging accuracy in a real-world scenario (not assuming gold error detection). Secondly, normalization is one way to go about processing non-canonical data, but not the only one (Eisenstein, 2013; Plank, 2016). Indeed, alternative approaches include leveraging the abundance of unlabeled data kept in its raw form. Recently introduced neural network parsers allow for new methods to effectively incorporate unlabeled data into the training process. These observations lead us to the following research questions:

- Q1 In a real-world setting, without assuming gold error detection, does normalization help in POS tagging of tweets?
- Q2 In the context of POS tagging, is it more beneficial to normalize input data or is it better to work with raw data and exploit large amounts of it in a semi-supervised setting?
- Q3 To what extent are normalization and semi-supervised approaches complementary?

To answer these questions, we run a battery of experiments that evaluate different approaches. Specifically:

1. We study the impact of normalization on POS tagging in a realistic setup, i.e., we compare normalizing only unknown words, or words for which we know they need correction; we compare this with a fully automatic normalization model (Section 6.2).
2. We evaluate the impact of leveraging large amounts of unlabeled data by deriving various types of word representations, and by studying their effect on model initialization (Section 6.3).

3. We experiment with combining the most promising methods from both directions, to gain insights on their potential complementarity (Section 6.4). Furthermore, the combined model is compared to an orthogonal approach (Section 6.5): the ARK tagger (Gimpel et al., 2011; Owoputi et al., 2013), a POS tagger specifically designed and tuned towards Twitter data.

This chapter is based on:

Rob van der Goot, Barbara Plank, and Malvina Nissim. To normalize, or not to normalize: The impact of normalization on part-of-speech tagging. In *Proceedings of the 3rd Workshop on Noisy User-generated Text*, pages 31–39, Copenhagen, Denmark, September 2017. Association for Computational Linguistics

The research described in this chapter is based on joint work with Barbara Plank and Malvina Nissim. Barbara tuned the POS tagger, whereas I ran all the normalization experiments. For the design and interpretations of the experiments as well as the writing, we all contributed equally.

The results in this chapter slightly deviate from the results reported in the paper, which is due to two main reasons. In the original paper, an older version of DyNet and MoNoise were used. MoNoise did not contain the `ORIGWORD` and `CONTAINSALPHA` features yet. Besides this, in the original paper the results for `TEST_LEXNORM` are a bit higher. This is because the original LexNorm data was pre-processed; in the paper we reverted this by retrieving the actual tweets. However, in this chapter, we use data from the LexNorm corpus, because the same data is also used in Chapter 5, and it simplifies comparison with other work.

The code to reproduce the results of this chapter can be found at:  
<https://bitbucket.org/robvanderger/chapter6>

## 6.1 Experimental Setup

We run three main sets of POS tagging experiments. In the first one, we use normalization in a variety of settings (Section 6.2). In the second one, we leverage large amounts of unlabeled data that does not undergo any normalization but is used as extra information source in a semi-supervised

	OWOPUTI	LEXNORM
Train	1,576	–
Dev	249	–
Test	549	549

Table 6.1: Number of tweets for our training, development and test sets.

setting (Section 6.3). In the third set of experiments, we examine the complementary of both approaches by combining them. In this section, we will describe the POS corpora and the POS tagger that is used for the experiments.

### 6.1.1 Data

The annotated datasets used in this chapter originate from two sources: Owoputi et al. (2013) and Han and Baldwin (2011), which we will refer to as Owoputi and LexNorm, respectively. All datasets used in this study are annotated with the 26 Twitter-specific POS tags as described in Gimpel et al. (2011), see for an overview of the tags Appendix D. Owoputi was originally only annotated with POS labels, whereas LexNorm was solely annotated for normalization. Li and Liu (2015) added a POS tag layer to the LexNorm corpus, and a normalization layer to 798 tweets from Owoputi, which we split into a separate development and test part of 249 and 549 tweets, respectively, keeping the original POS labels. We outlined the new data splits in Table 6.1. We will refer to the test sets from Owoputi and LexNorm as respectively `TEST_OWOPUTI` and `TEST_LEXNORM`.

For the improvements of the baseline tagger in semi-supervised settings, we use the raw Twitter data described in Section 2.2.2.

### 6.1.2 Bilty

We use Bilty, an off-the-shelf bi-directional Long Short-Term Memory (bi-LSTM) tagger which utilizes both word and character embeddings (Plank et al., 2016). The tagger is trained on 1,576 training tweets (Section 6.1.1). We tune the parameters of the POS tagger on the development set to derive the following hyperparameter setup, which we use throughout the rest of

the experiments: 10 epochs, 1 bi-LSTM layer, 100 input dimensions for words, 256 for characters,  $\sigma=0.2$ , constant embeddings initializer, Adam trainer, and updating embeddings during backpropagation.

## 6.2 The Effect of Normalization on POS Tagging

For normalization, we use the MoNoise model from Chapter 4 trained on the data from LiLiu (see also Section 2.2.1).

To obtain a more detailed view of the effect of normalization on POS tagging, we investigate five experimental setups:

- `NO_NORM`: using the raw text (no normalization)
- `NORM_UNK`: normalizing only unknown words;
- `NORM_ALL`: considering all words: the model decides whether a word should be normalized or not;
- `NORM_GOLDERRDET`: assuming gold error detection: the model knows which words should be normalized;
- `NORM_GOLD`: gold normalization; we consider this a theoretical upper bound.

Traditionally, the goal of normalization is to transform the test data to be more similar to the training data. Since in our setup, we train our tagger on social media data, the normalization of only the test data might actually result in more distance between the train and test data. Therefore, we also train the tagger on normalized training data, and on the concatenation of the normalized and the original training data. For the training data, the only normalization strategy we tested is `NORM_ALL`, because the others are not available.

The effects of the different normalization strategies on the development data are shown in Table 6.2. Throughout this chapter, we report average accuracies over 10 runs using different seeds for the bi-LSTM.

The first column shows the effect of normalization at test-time only. From these results we can conclude that it is beneficial to let the normalization model decide whether to normalize over normalizing only unknown words;

Development	Training data		
	NO_NORM	NORM_ALL	CONCAT
NO_NORM	83.97	82.29*	84.10
NORM_UNK	85.89*	84.56*	86.19
NORM_ALL	86.76*	86.80	87.17*
NORM_GOLDErrDET	86.92	86.97	87.21
NORM_GOLD	87.87*	87.86	88.09

Table 6.2: Results of normalization on the development data (macro average over 5 runs). CONCAT stands for the training set formed by the concatenation of both normalized and original raw data. \* Significantly different in a paired t-test at  $p < 0.01$ ; for the NO\_NORM column compared to the previous row and for the other columns compared to the NO\_NORM column.

this shows that normalization has a positive effect that goes beyond replacing unknown words to known words. The results of NORM\_GOLD suggest that there is still more to gain by improving the normalization model.

Interestingly, the results for gold error detection (NORM\_GOLDErrDET) show that error detection is not the main reason for this difference, since the performance difference between NORM\_ALL and NORM\_GOLDErrDET is relatively small compared to the gap with NORM\_GOLD. This is not in line with the evaluation of the normalization model, where error detection was a major weakness of the model (Section 5.3). Upon inspection of the data, we think that the main reason for this is that the most difficult cases for the normalization are also most problematic for the POS tagger. The normalizations which are not found by MoNoise even when using gold error detection, are indeed very distant normalization replacements. Even though there are not many of these, their influence on the final performance is quite substantial.

Considering the normalization of the training data, we see that using only the normalized training data has a negative effect. However, when concatenating the normalized training data with the original training data, performance improves. This is in contrast with the original paper (van der Goot et al., 2017), where CONCAT also did not lead to a performance

improvement. This is an effect of having a stronger normalization model, which is better at deciding when to normalize (the `ORIGWORD` feature was added later, see Section 4.4).

To sum up, normalization improved the base tagger by 3.4 percentage points on the development data, reaching 87.17% accuracy. Normalizing the training data is only beneficial when used in combination with the original training data. Overall, our state-of-the-art normalization model reaches 75% of the theoretical upper bound of using gold normalization. We next investigate whether using large amounts of unlabeled data can help us to obtain a similar effect.

### 6.3 Semi Supervised Settings

An alternative option to normalization is to leave the text as is, and exploit large amounts of raw data via semi-supervised learning. An easy and effective use of word embeddings in neural network approaches is to train embeddings on external raw data, and use these embeddings to initialize the word lookup parameters. Since large amounts of raw data is readily available for web domains, this is a promising direction. In this section, we compare the effect of different settings for the word embeddings without using any normalization, in the next section we will compare and combine both approaches.

For the initialization of the POS tagger, we experiment with a skip-gram word embeddings model using `word2vec` (Mikolov et al., 2013a) on the same tweets as used for the normalization model (as described in Section 2.2). We also experiment with structured skip-grams (Ling et al., 2015), an adaptation of `word2vec` which takes word order into account. It has been shown to be beneficial for syntactically oriented tasks (Ling et al., 2015; Lin et al., 2015). We experiment with embeddings using the default hyperparameters, and try to tune the dimensions and the window size.

Table 6.3 shows the results of using the different skip-gram models for initialization of the word embeddings layer. Structured skip-grams perform slightly better compared to normal skip-grams, confirming earlier findings. Using a smaller window is beneficial, probably because of the fragmented nature of Twitter data.

Structured skip-grams of window size 1 and 400 dimensions result in

---

Dimensions	100		400	
Window size	1	5	1	5
Skip-grams	89.91	89.13	90.04	89.24
Structured Skip-grams	90.25	89.96	90.29	90.13

---

Table 6.3: Accuracy on the development data (not normalized): various pre-trained skip-gram embeddings for initialization of the tagger.

the best embedding model. This results in an improvement from 83.83% (Table 6.2) to 90.29% accuracy. However, using only 100 dimensions results in a much smaller model with only a minimal, not significant performance loss. Hence, we will use structured skip-grams with a window of 1 and 100 dimensions in the rest of this chapter. The performance improvement is considerably larger than what was obtained by normalization (87.17). The advantage of word embeddings is that they add information for every word, whereas the normalization only replaces approximately 5% of the words. In the next section we will examine these differences in more detail, and test to what extent the approaches are complementary.

## 6.4 Combining Normalization and Semi Supervised Learning

In the previous sections, we explored ways to improve the POS tagging of tweets. The most promising directions were initializing the tagger with pre-trained embeddings and using normalization. Self-training was not effective. In this section, we report on additional experiments on the development data aimed at obtaining insights on the potential of combining these two strategies. We use only the best performing settings for both the pre-trained embeddings and the normalization. To recall: the best normalization setting was to use the concatenation of the normalized and raw training data, considering all words for normalization at test time; the best word embeddings model was a structured skip-gram embeddings model with a window of 1 and 100 dimensions.

	% of data	Bilty	+Norm	+Embeds	+Comb
Known	78.08	91.79	92.97	93.96	94.40
Unknown	21.92	55.92	64.68	76.83	78.45
All	100.0	83.93	86.77	90.21	90.90

Table 6.4: Effect of different models on known and unknown words on development data (accuracy). Known words are words which occur in the training data, whereas unknown words are not seen by the tagger during training (they might be present in the external embeddings though).

#### 6.4.1 Effect on Known Words Versus Unknown Words

Table 6.4 shows the effect of the two approaches separately as well as combined on the development data for two subsets of tokens: known and unknown, words which are respectively present and absent in the training data. Word embeddings have a higher impact on both the known and unknown tokens. Unknown words are clearly a challenge for the tagger, even when external embeddings, character embeddings, and normalization are used, accuracy is still only 78.45%. Interestingly, normalization also improves the scores for known words, confirming our earlier observations that the improvements go beyond replacing unknown words with similar known words (Section 6.2). Another, perhaps surprising, observation is that external embeddings are actually more beneficial for unknown words compared to normalization. This shows the strength of the external embeddings. The final column shows that both approaches are complementary, indicating that they do target different types of errors.

#### 6.4.2 Performance per POS

We plotted the type of confusions made by our combined model in Figure 6.2 (an overview of the tagset can be found in Appendix D). The most prominent confusions are between proper nouns (^) and nouns (N) in both directions, which are often due to non-standard capitalization even though this also occurs in the training data. Unfortunately, the normalization model cannot handle these cases very well, due to the fact that capitalization is not consistently annotated in the training data.



	DEV_O	TEST_OWOPUTI	TEST_LEXNORM
Bilty	83.93	82.58	77.89
+Norm	87.10*	86.35*	82.51*
+Embeds	90.29*	89.40*	85.64*
+Comb	90.89*	90.33*	86.17*
Ark	90.69	90.00	86.56

Table 6.5: Results on the test data compared to ARK-tagger (Owoputi et al., 2013). \* Significant using a paired t-test at  $p < 0.01$  compared to the previous row.

## 6.5 Evaluation

In this section we report results on the test data, as introduced in Section 6.1.1. Our main aim is to compare different approaches for successfully applying a generic state-of-the-art POS tagger to Twitter data. Therefore we have to assess the contribution of the two methods we explore (normalization and using embeddings) and see how they fare, not only to each other but also in comparison to a state-of-the-art Twitter tagger. We use the ARK tagger (Owoputi et al., 2013) and retrain it on our dataset for direct comparison with our models. The ARK system is a conditional random fields tagger, which exploits brown clusters (Brown et al., 1992), lexical features and gazetteers. We do not compare to other previous work because we use different data splits to avoid tuning on the test data.

Table 6.5 shows the performance of our best models and the ARK tagger on the test datasets. External embeddings perform considerably better than normalization, which confirms what we found on the development data. The combined approach yields performance on par to the Ark tagger for all datasets. The results on TEST\_LEXNORM are consistently lower, because of different tokenization of punctuation compared to the training data. Note that in van der Goot et al. (2017) we manually corrected this, and obtained scores similar to the other datasets.

## 6.6 Conclusion

In this chapter we investigated the impact of normalization on POS tagging for the Twitter domain, presenting the first results using fully automatic normalization and comparing normalization to alternative strategies. We compared a generic tagger to a tagger specifically designed for Twitter data.

Considering the normalization for the vanilla tagger, normalizing the training data led to a small performance improvement. Letting the normalization model choose which words to normalize consistently outperformed normalizing only the words unknown to the tagger. Overall, the best normalization model reached 75% of the performance of the theoretical upper bound of using gold normalization.

Using large amounts of unlabeled data for embedding initialization yields an improvement that is twice as large as the one obtained using normalization. Both methods have shown to be complementary across multiple datasets. Our final model performs on-par with the ARK tagger, which is a carefully domain-tuned tagger.

Normalization could prove to be more useful when training on a canonical domain, where it makes the test data closer to the train data. However, for the datasets used in this chapter, an idiosyncratic tagset is used, so this training data is not available. For the next two chapters on respectively constituency and dependency parsing, this setup is enforced, since only small development and test treebanks exist for the social media domain.

Part III

Constituency Parsing



## Chapter 7

# Integration of Normalization in a PCFG-LA Parser

In this chapter we will evaluate the effect of normalization on constituency parsing. For more details on constituency structures and basic constituency parsers, see Section 3.1. In the last decades, constituency parser have continuously improved, and now reach accuracies well above 90%. However, these accuracies are usually benchmarked on news texts, and do not transfer well to other types of language.

The magnitude of the domain-shift problems for the social media domain becomes clear when training the Berkeley parser on newswire text, and comparing its in-domain performance with performance on the Twitter domain. The Berkeley parser achieves an F1 score above 90 on newswire text (Petrov and Klein, 2007). An empirical experiment that we carried out on a small Twitter treebank revealed that the F1 score drops below 70 for this domain. Part of this performance drop can be explained by the distant training data. But another important factor is the higher amount of variety of language use on social media.

By using normalization, we attempt to make the input data more similar to the training data. Orthogonal approaches focus on making the training data more similar to the input data. However, using normalization has some benefits: grammars do not have to be re-trained and when adapting to another time-span or domain, only the normalization model has to be updated.

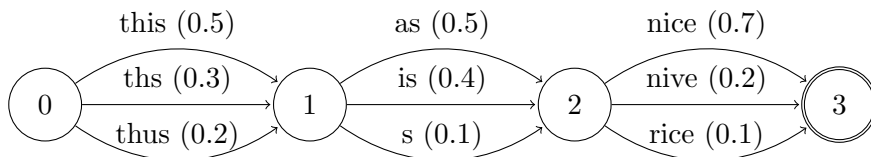


Figure 7.1: A possible output of the normalization model for the sentence ‘ths s nice’.

We compare the traditional approach of only using the best normalization sequence with an integrated approach, in which the parsing model has access to multiple normalization candidates for each word. In this setup, direct propagation of mistakes made by the normalization model can be circumvented. In practice, we will represent the output of the normalization model as a word graph; the parsing as intersection algorithm (Bar-Hillel et al., 1961) can then be used to find the optimal parse tree over this lattice.

An example normalization output of the sentence ‘ths s nice’ is shown in Figure 7.1. In this example output, the probability of ‘as’ is higher than the probability of ‘is’, whereas the correctly normalized sequence would be ‘this is nice’. The parser can disambiguate this word graph because it has access to the syntactic context: ‘is’ is usually tagged as VBZ, while ‘as’ is mostly tagged as IN. This example shows the main motivation for using an integrated approach; this enables the parser to make use of all the information from the normalization.

The two main contributions of this chapter are:

- We show that the use of lexical normalization is useful when parsing social media data
- We show that integrating the normalization into the parsing algorithm leads to an even better parser

Additionally, we test for both of these settings (direct normalization and integrated normalization) whether normalization is only useful for unknown words.

We start this chapter with a brief summary of previous work on using normalization to improve constituency parsers (Section 7.1). In Section 7.2 we will review the data which is used to train and evaluate our models. Next,

we describe the method used to integrate the normalization, first for the basic CYK algorithm, followed by how it can be done in a PCFG-LA parser. In Section 7.4 we evaluate the effect of using normalization directly and integrating normalization. A qualitative analysis of the effect of integrating normalization follows in Section 7.5. Finally, we take a look at the effect of integrating normalization on the efficiency of the parser in Section 7.6

This chapter is based on:

Rob van der Goot and Gertjan van Noord. Parser adaptation for social media by integrating normalization. In *Proceedings of the 55th Annual Meeting of the Association for Computational Linguistics (Volume 2: Short Papers)*, pages 491–497, Vancouver, Canada, July 2017b. Association for Computational Linguistics

The results reported in this chapter are slightly different compared to the paper, because we use a newer version of MoNoise in this chapter.

The code to reproduce the results of this chapter can be found on:  
<https://bitbucket.org/robvandergerberkeleygraph/>

## 7.1 Related Work

The first evaluation of normalization for constituency parsing originates from Foster (2010). She experiments with a rule-based normalization on forum data and reports a performance gain of 2% in F1 score on the task of constituency parsing.

SANCL 2012 hosted a shared task on parsing the English Web Treebank (EWT) (Petrov and McDonald, 2012). A wide variety of different approaches were used: ensemble parsers, product grammars, up-training, word clustering, genre classification, and normalization. The teams that used normalization used simple rule-based systems, and unfortunately did not report the actual performance improvement. Arguably the effect would be rather small, because the EWT contains only a relatively small amount of anomalies (see also Section 5.5).

A theoretical exploration of the effect of normalization on constituency parsing data is done by Kaljahi et al. (2015). They released the Foreebank,

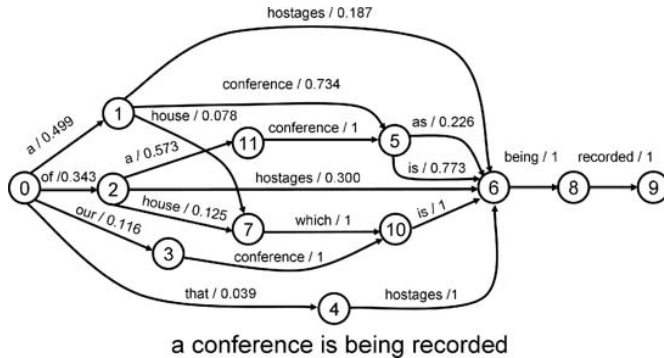


Figure 7.2: Output of a speech recognition system for the spoken phrase “a conference is being recorded”.

a treebank consisting of forum texts, annotated with normalization and constituency trees. They show that parsing manually normalized sentences results in an increase from 77.0 to 78.6 F1 score; still far from performance on clean texts.

To the best of our knowledge, there is no work exploiting the top-N normalization candidates for constituency parsing. However, for the parsing of speech, the top-N candidates has often been used for parsing. In speech recognition, the audio signal is usually converted to a probabilistic word graph by the speech recognition system. Hence, early work on the parsing of speech already used parsing algorithms which accept word graphs as input (Bates, 1975; Lang, 1989). An example of the output of a speech recognizer is shown in Figure 7.2<sup>1</sup>.

In our setup, the observed sequence is already a sequence of words, however, these are not in the form as expected by the parser. So, instead of converting an audio signal to a word graph, we convert a non-standard text to a word graph.

## 7.2 Data

In this section we will describe the data which is used to train and test our parser. The normalization model we use is the same one as in chapter 6,

<sup>1</sup>taken from Gibbon and Liu (2008)

Corpus	Sentences	Words/sentence	Unk%
WSJ (2-21)	39,832	23.9	4.4
EWT	16,520	15.3	3.7
Foster et al. (2011a) <sup>*</sup>	269	11.1	9.3
Li and Liu (2014) <sup>†</sup>	2,577	15.7	14.1

Table 7.1: Some basic statistics for our training and development corpora. Unk%: percentage unknown words calculated against the Aspell dictionary ignoring capitalization. <sup>\*</sup>Only the development part. <sup>†</sup> Only used for training the normalization model

and is trained on the LiLiu corpus (see Section 2.2.1).

We use the treebank from Foster et al. (2011a) as development and test data for our parser. It consists of 519 tweets annotated with constituency trees, split in a development set (269 tweets) and a test set (250 tweets). For training, we use the English Web Treebank (EWT) concatenated with the standard training sections (2-21) of the Wall Street Journal (WSJ) part of the Penn Treebank. For more information on these treebanks, we refer to Section 3.3.

Some basic statistics of our training and development data can be found in Table 7.1. Perhaps surprisingly, the percentage of unknown words (words not present in the Aspell dictionary, ignoring capitalization) in the EWT is lower than in the WSJ. This can be explained by the fact that the WSJ texts contains lots of jargon and named entities which are not present in the Aspell dictionary. The difference in the percentage of unknown words between the normalization training data and the development treebank data is rather large. Derczynski et al. (2013) and Plank (2016) also observed that the performance for POS tagging on this dataset is higher compared to other datasets containing Twitter data. We decided to test how many anomalies the development set contains by annotating this dataset for normalization in a similar style as LiLiu. This resulted in 1.9% of all words normalized, in contrast to 10.5% in LiLiu. At first sight, this might be an issue for our approach. Fortunately, our normalization model has proved to be rather robust (see Section 5.5). Nevertheless, the effect of normalization will be smaller when there are fewer anomalies in the data.

## 7.3 Method

In this section we will first discuss how parsing as intersection can be implemented in the CYK algorithm (Section 3.1.4), after which we explain the concrete setup used in our experiments.

### 7.3.1 Uncertainty in CYK

In this section we will discuss how we can model uncertainty on the word-level in the CYK algorithm. For simplicity, we assume that the word boundaries are given, i.e. normalization is performed on the word level. The main adaptations are done on the POS tag level, as only the word input is different compared to the normal CYK algorithm.

To allow for the insertion of multiple normalization candidates at the same position, we can simply insert the POS tags for multiple words into the chart. If two words on the same position can have the same POS tag, the one with the highest probability will always be used in the final tree. So, a backtracking pointer to this word should be kept, and the pointer to the other words can be removed. Effectively, this can result in the pruning of words as well.

It should be noted that this is not a full implementation of the parsing as intersection algorithm (Bar-Hillel et al., 1961). This implementation relies on the assumption that every word occupies a span of length 1. In other words, the word boundaries are given<sup>2</sup>.

An example of the integration of normalization into the first row in the chart is shown in Figure 7.3. In the first position, both ‘thus’ and ‘ths’ can be assigned the NN tag. However, the probability of ‘thus’ being an NN in position 0,1 is higher compared to the probability of ‘ths’ being a NN in position 0,1. Because of this, ‘ths’ will never be used in a final parse. Similar to ‘ths’, ‘nive’ is also already pruned away from the chart. In both of these cases, the probabilities are low because the words are unlikely normalizations as well as unknown to the POS tagger. In the next section we will motivate how we combine the probability from the POS tagger with the probability from the word graph.

---

<sup>2</sup>Actually, our adaptation to the Berkeley parser also includes a more elaborate graph parsing, which can correctly handle a word graph with differences in word boundaries. However, this is not used in this Thesis.

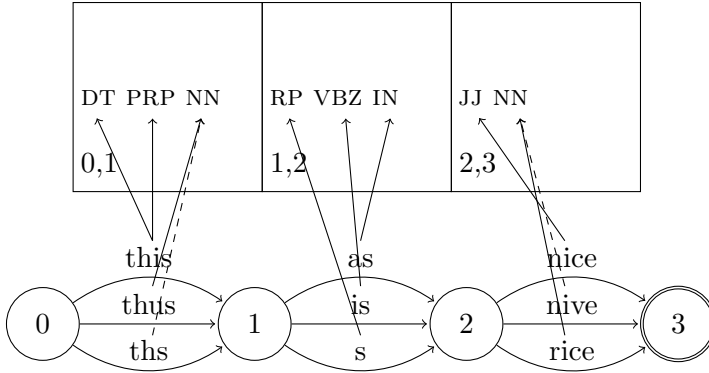


Figure 7.3: The integration of multiple words into the chart for one position. A dashed line represents a word which is pruned from the chart.

### 7.3.2 Concrete Setup

We use the Berkeley parser (Petrov and Klein, 2007) as a starting point for our experiments. This is a PCFG-LA parser, which means that constituents are divided into sub-groups to be able to model more specific language constructions. Parsing is done hierarchically; after each sieve, unlikely constituents are pruned from the chart (see Section 3.1.5 for more detail). The algorithmic changes explained in the previous section can also be used in a hierarchical parsing setup. The only addition is that we have to keep track of which word is used for which POS tag in each parsing sieve, this is necessary to be able to build a complete tree, including the words from the word graph which are used. Note that the integration of a word lattice into a PCFG-LA parser is also done by Goldberg and Elhadad (2011) and Constant et al. (2013) but they do not incorporate probabilities from the word lattice into the parsing algorithm.

To incorporate the probability from the normalization model ( $p_{norm}$ ) into the chart, we combine it with the probability from the POS tag assigned by the built-in tagger of the Berkeley parser ( $p_{pos}$ ) using the weighted harmonic mean (Rijsbergen, 1979):

$$p_{chart} = (1 + \beta^2) * \frac{p_{norm} * p_{pos}}{(\beta^2 * p_{norm}) + p_{pos}} \quad (7.1)$$

Here,  $p_{chart}$  is the probability used in the parsing chart.  $\beta$  can be used to tune the weight between the probability of the POS tagger and the normalization. A  $\beta$  of 1.0 gives equal weight to both probabilities, whereas a lower  $\beta$  gives more importance to  $p_{pos}$ , and a higher  $\beta$  gives more weight to  $p_{norm}$ . We use this formula instead of a simple multiplication for multiple reasons:

1. It allows us to weigh the importance of the normalization and at the same time reward the model if both probabilities are more balanced.
2. The probabilities are not independent, so direct multiplication should not be used.
3. The range of the probabilities is different, the normalization often gives probabilities close to 1.0 and 0.0. The POS tagger usually gives lower scores, especially for words unknown to the parser.

In this chapter we will compare three approaches:

- Vanilla Berkeley parser: baseline
- Direct normalization: the traditional approach of translating a sentence to its normalized equivalence before parsing
- Integrated normalization: exploits the top-N candidates

Within the approaches which make use of normalization, we compare normalizing only the words unknown to the parser against normalizing all words. We refer to these approaches as UNK and ALL, respectively. Figure 7.1 shows a possible output when using ALL. When using UNK, the word ‘nice’ would not have any normalization candidates. For the UNK approach, we also retrain the normalization model to only consider words unknown to the parser for normalization.

## 7.4 Results

The parser is evaluated using the F1 score as implemented by EVALB<sup>3</sup>. All results in this section are the average over 10 runs, using different seeds for the normalization model, unless mentioned otherwise.

---

<sup>3</sup>[nlp.cs.nyu.edu/evalb](http://nlp.cs.nyu.edu/evalb)

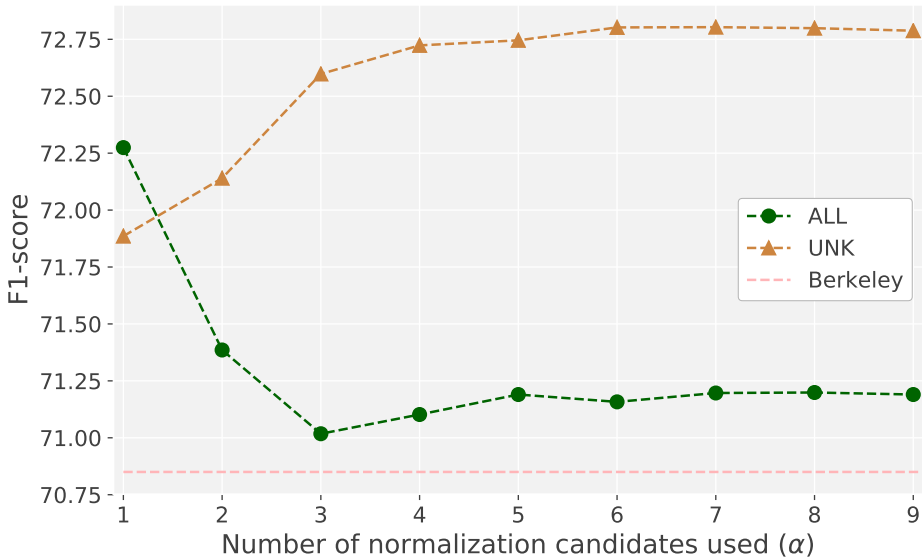


Figure 7.4: F1 scores on the development data when using multiple candidates while normalizing ALL words or only the UNKnown words ( $\beta = 2$ ), compared to the vanilla Berkeley parser. At  $\alpha = 1$ , the normalization is used directly.

The performance of our model depends on two parameters: the number of normalization candidates per word  $\alpha$  and the weight given to the normalization  $\beta$ . We tuned these parameters on the development data using  $\alpha \in [1 - 10]$  and  $\beta \in [0.125, 0.25, 0.5, 1, 2, 4, 8, 16]$  to find the optimal values. The best performance is achieved by UNK,  $\alpha = 6$  and  $\beta = 2$ . From this optimal setting, we will compare the effects of both variables for both the UNK and the ALL normalization strategies.

Figure 7.4 shows the effect of using different numbers of candidates ( $\alpha$ ) and our baseline: the vanilla Berkeley parser. Using the single best normalization sequence directly ( $\alpha = 1$ ) we can obtain an improvement of 1.4 percentage point when normalizing all tokens. If we only normalize the unknown tokens the performance is slightly worse, but it still outperforms the baseline.

If we use more normalization candidates ( $\alpha > 1$ ), performance increases;

Weight( $\beta$ )	ALL ( $\alpha = 1$ )	UNK ( $\alpha = 6$ )
0.125	71.23	71.45
0.25	71.23	71.51
0.5	71.25	71.68
1	71.57	72.10
2	<b>72.27</b>	<b>72.80</b>
4	72.21	72.33
8	71.76	72.48
16	70.55	71.07

Table 7.2: F1 scores on the development data using different weights, comparing only using the best candidate versus using 6 candidates.

it converges around  $\alpha = 6$ . At this optimal setting, the baseline is outperformed by 2.0 percentage point. Interestingly, if more than only the first candidate is used, it is not beneficial to normalize all words anymore. This is probably an effect of creating too much distance between the original sentence and the normalization. The F1 score converges for higher numbers of candidates, because lower ranked candidates have very low normalization probabilities and are thus unlikely to affect the final parse.

Table 7.2 shows the results using different weights. We compare the optimal setting for both ALL ( $\alpha = 1$ ) and UNK ( $\alpha = 6$ ). For both settings,  $\beta = 2$  gives the best results, showing that normalization probability should be given a higher weight compared to the probability from the POS tagger. Increasing the weight even further results in lower performance, indicating over-normalization. Furthermore, the trends look similar for both settings.

For the test data, we use the parameter settings that performed best on the development treebank (UNK,  $\alpha = 6$ ,  $\beta = 2$ ) and the best performing seed for the normalization model. The results on the test data are compared to the traditional approach of only using the best normalization sequence, the vanilla Berkeley parser, and the Stanford PCFG parser (Petrov and Klein, 2007), in Table 7.3. The integrated approach significantly outperforms the Berkeley parser as well as the traditional approach. It becomes apparent that the test part of the treebank is more difficult than the development part. Although the increase is smaller, normalization still improves parser

Parser	dev	test
Stanford parser	66.05	61.95
Berkeley parser	70.85	66.52
Best norm. seq.	72.03	67.06
Integrated norm.	73.14*	67.36*
Gold POS tags	74.98	71.80

Table 7.3: F1 scores of our proposed models and vanilla PCFG-LA parsers on the test set, trained on the EWT and WSJ. \*Statistical significant against Berkeley parser at  $p < 0.01$  and at  $p < 0.05$  against the best normalization sequence using a paired t-test.

performance. On the development set, 55% of the errors which can be accounted to mistakes made by the POS tagger are solved, whereas, on the test set, we only solve 16% of this theoretical upper bound.

## 7.5 Analysis

This section contains some extra analysis to gain a deeper understanding for which cases our approach improves performance. Firstly, we test whether the effect of integrating normalization is similar to lowering the pruning thresholds, which also leads to a slightly larger search space. Secondly, we evaluate performance on newswire texts, which will show if the performance improvement also transfers to domains which do not require normalization. Thirdly, we evaluate the performance of the parser on the normalization task; recall that the parsing as intersection algorithm also disambiguates the word-graph, and yields a syntactically motivated best path.

### 7.5.1 Effect of Lower Pruning Thresholds

The insertion of multiple normalization candidates on one position in the chart leads to higher probabilities for multiple POS tag positions in the chart. This in turn, should lead to less pruning in the Berkeley parser, because more constituents in the tree will have a relatively high probability

(this hypothesis is confirmed in Section 7.6). This could lead to effects very similar to lowering the pruning thresholds. To test if this is the case, we evaluate the vanilla Berkeley parser with lower pruning thresholds<sup>4</sup> on the Twitter development treebank. This resulted in a decrease in F1 score from 70.85 to 70.64, showing that the integration of normalization has a different effect.

### 7.5.2 Performance on Canonical Data

To test performance on canonical data, we ran our proposed parsing model on the standard development part of the WSJ treebank (section 24), which contains well-edited news texts. On this data, the normalization should not lead to a better performance, since there are only very few anomalies. The vanilla Berkeley parser achieves an F1 score of 89.15, whereas the best integrated model (UNK,  $\alpha = 6$ ,  $\beta = 2$ ) scores 89.12 due to minor over-normalization. This shows that the performance improvement does not transfer to this relatively standard data, confirming that the performance improvement on the other corpora is not simply due to increasing probabilities in the chart.

Furthermore, the results show that our model is rather robust. Even on well-edited news texts, the system over-normalizes only a little. A closer inspection revealed that this decrease is mostly an effect of a few very frequent mistakes. The normalization can easily be made more robust by ignoring these cases. These mistakes are mostly made because of differences in tokenization between the normalization training data and the treebanks (see also Section 5.5, in which MoNoise is evaluated on the WSJ).

### 7.5.3 When is Integrating Normalization Beneficial?

The normalization model seldom finds a correct candidate beyond  $\alpha > 2$ , at  $\alpha = 2$  the recall for unknown words is 89.7% on the LexNorm corpus (Han and Baldwin, 2011), whereas the accuracy at  $\alpha = 6$  is 94.5% (See also Section 5.4.2). Perhaps surprisingly, the parser performance still improves when increasing  $\alpha$  even when only 1.9% of the words in the development data are in need of normalization. In this section, we will use the development

---

<sup>4</sup>Tested by running the Berkeley parser with `--accurate`. We also tried to tune the thresholds even further manually, but this had similar effects.

data to investigate in which situations our approach leads to a better parse tree.

In the parsing as intersection algorithm, the parser searches the most probable tree with respect to a word graph. As a result, it does not only yield a tree, but also a path through the word graph. This can be considered a syntactically motivated normalization sequence.

When testing the performance on the normalization task after parsing the LexNorm dataset, the error reduction rate (Section 5.1) drops to 10%, whereas MoNoise itself reached an error reduction rate around 60% (Section 5.2.1). After manually inspecting the sentences for which the integrated parser preferred a different normalization sequence compared to the normalization system, we found that the main reasons for the lower error reduction rate fall in four categories:

- Not normalizing known words: as an effect of using the UNK strategy, many words are not considered for normalization. Because the English Web Treebank is also part of the training data, this excludes a set of anomalies as well.
- Differences in tokenization: because in treebanks tokenization is done differently, some common words are unknown to the parser as well as MoNoise and are thus processed differently. For example, the word “i’m” is unknown to the parser, so it is likely to be replaced.
- Erroneous paraphrasing: if a word suggested by the normalization is more likely in the syntactic context, the parser often chooses to use this word. In most of these cases, the most probable normalization candidate is unknown to the POS tagger, and thus gets a very low POS probability.
- Twitter usernames and hashtags: Because we do not handle twitter usernames differently, these are sometimes replaced by lexically similar words or names. This can be considered a special (domain-specific) case of the previous category.

To better illustrate the type of mistakes which are made by the parser on the normalization task, we show some examples of each of these error categories in Table 7.4. A closer look at these examples reveal that in some

Category	Normalization of parser
Not normalizing known words	thanx $\mapsto$ thanx, lil $\mapsto$ lil, plz $\mapsto$ plz
Differences in tokenization	im $\mapsto$ I, i'm $\mapsto$ in
Erroneous paraphrasing	kerry $\mapsto$ jerry, indicated $\mapsto$ inducted, utube $\mapsto$ tube
Username and hashtags	@chynamonroe $\mapsto$ Monroe, @an2ony $\mapsto$ anyone, #celebrityhour $\mapsto$ celebrity

Table 7.4: Some examples of erroneous normalization replacements made by the parser.

cases a wrong normalization can lead to a better parse, especially for the erroneous paraphrasing and usernames. In these cases, the replacement often shares some syntactic properties with the original word. However, the results with using normalization directly (ALL,  $\alpha = 1$  in Figure 7.4) showed that correct normalizations can already lead to a large proportion of the improved performance.

The lower error reduction rate after parsing can be considered a somewhat disappointing result. However, this can be understood if we think about the task the parser is given; it has to improve upon a classifier which uses a wide variety of features (some of which based on huge amounts of raw texts), which is also optimized directly on the normalization task. It is not surprising that re-ranking using only syntactic information does not improve the normalization. Performance on the normalization task could perhaps be improved by tuning  $\beta$ , however, this would lead to an inferior parser.

### Examples of Parser Improvements

Besides the examples where using the parser lead to a worse normalization, there are also cases where the integration of normalization leads to a better parse as well as a better normalization. Below, we will discuss two of these instances.

The first example is shown in Figure 7.5. In the output of MoNoise, the correct replacement for ‘RIDICULOUS’ was ranked second, because capitalization is not corrected in the training data. However, this lower

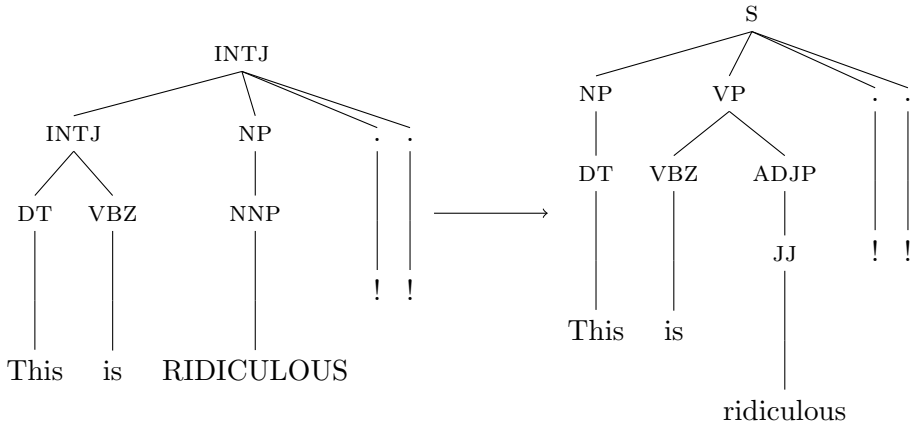


Figure 7.5: An example parse from the development corpus; left is the output of the vanilla Berkeley parser, the right tree is the result after integrating normalization.

ranked candidate is much more likely in the syntactic context, so the parser prefers the correct normalization candidate. Simultaneously, this leads to the correct syntactic tree.

For our second example, shown in Figure 7.6, our parser normalized two words. Without the re-ranking of the parser, MoNoise would keep the original word in both cases. In the first case, ‘except’ is only the fourth normalization candidate. However, the other candidates ‘expert’ and ‘expat’, are not very likely candidates with respect to the constituency tree, hence the parser could correctly find the correct normalization. The second replacement ‘Mets→Dodgers’, is an incorrect replacement, which we would categorize as erroneous paraphrase. In this case, a candidate with a similar word embedding ends up in the candidate list and is also a known word for the parser (the token ‘Mets’ does not occur in the WSJ and EWT). So, despite the low probability from the normalization, the parser prefers ‘Mets’ for this position. The effect on the final parse is probably not that big, since ‘Mets’ was already tagged correctly.

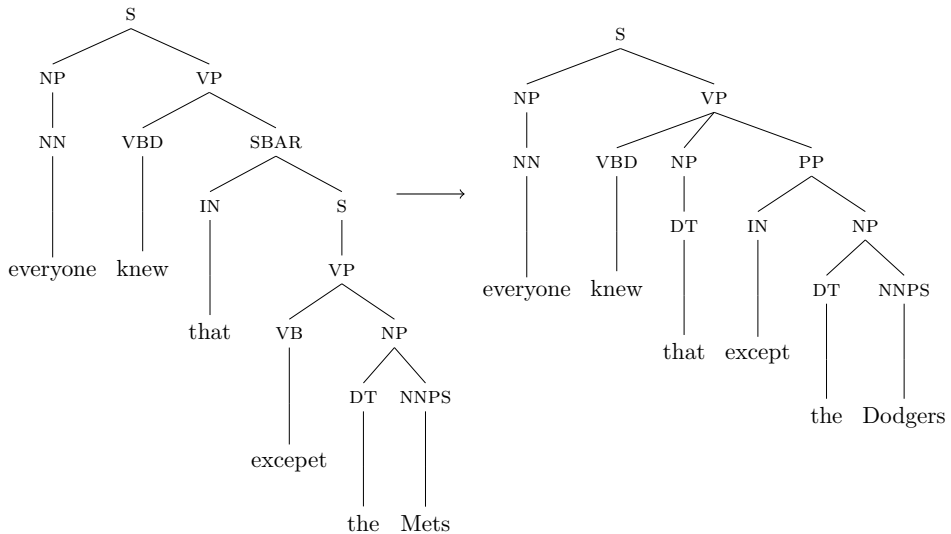


Figure 7.6: An example parse from the development corpus; left is the output of the vanilla Berkeley parser, the right tree is the result after integrating normalization.

## 7.6 Efficiency

To test the effect of the normalization on the search space, we simply count the number of surviving constituents in the chart in the middle and final parse level (see Section 3.1.5 for an explanation on hierarchical parsing). These have a direct impact on the computational cost of the algorithm, since the parser does not have to explore pruned constituents in later levels. Results are shown in Table 7.5. There is a slight increase of approximately 1% in the number of constituents for both parse levels when integrating normalization.

We also tested the effect of integrating normalization on the time the parser takes to parse the development data. These experiments showed a similar effect compared to the number of constituents; averaged over 10 runs, the vanilla Berkeley parser took 24.3 seconds on the development set, whereas our model took 24.5 seconds on the same machine. The main source of extra computation time comes from running the normalization

Parse level	Berkeley	Integrated Norm.
3	756	765
6	3,086	3,115

Table 7.5: The average number of constituents in the chart per sentence for the middle parsing level (3) and the final level (6) of the hierarchical parsing on our development set.

model, which on average takes 30.2 seconds for the development set.

## 7.7 Conclusion

We have shown that we can significantly improve the parsing of out-of-domain data by using normalization. If we use normalization as a simple pre-processing step, we observe a substantial improvement in performance, while higher improvements can be achieved by using an integrated approach. Additionally, we have shown that when using only the best normalization sequence, it is better to normalize all words instead of only the unknown words. However, when using an integrated approach it is better to only consider unknown words for normalization. Further analysis revealed that improvements in parsing performance are not only an effect of using correct normalization candidates, but are also due to wrong normalization candidates which share syntactic properties with the original word.



Part IV

Dependency Parsing



## Chapter 8

# Integration of Normalization in a Neural Network Parser

In the previous chapter we have shown that normalization is beneficial for a PCFG-LA parser when parsing social media data. When exploiting the top-N normalization candidates in an integrated setup, performance increased even further. In this chapter, we will investigate the effect of normalization for another syntactic formalism: dependency grammar. Dependency grammars model the relations between words directly, which has advantages for efficiency and makes extraction of certain relationships between words easier (Covington, 2001).

Previous work has shown that for feature-based dependency parsers, normalization is beneficial for the parsing of social media data (Foster, 2010; Zhang et al., 2013; Baldwin and Li, 2015). However, recently neural network parsers have become prevalent (Chen and Manning, 2014; Dyer et al., 2015; Kiperwasser and Goldberg, 2016), reaching new state-of-the-art performances on standard texts. These neural network parsers allow for new methods to model character level information (de Lhoneux et al., 2017a; Ballesteros et al., 2015; Nguyen et al., 2017) and exploit external raw text in a semi-supervised setup. These new methods are especially beneficial for unknown words, which are the same words as targeted by normalization. This leads to the question whether normalization is indeed no longer required for these modern character-based neural network parsers, or whether normalization is capable of solving problems beyond the scope

of this type of neural network parsers.

Another shortcoming of the previous work on normalization for dependency parsing is that they directly parse the best normalization sequence. This leads to error propagation. Similar to Chapter 7, we attempt to mitigate this error propagation by exploiting the top-N normalization candidates. However, in a neural network parser we can use other integration techniques as words are represented as continuous vectors. Previous work has shown that arithmetics can be used on this vector to combine meanings of words. For example, Mikolov et al. (2013a) showed that  $vector(king) - vector(man) + vector(woman) \approx vector(queen)$ , and Arora et al. (2016) showed that the meaning of a sentence can be successfully captured in a vector by simply averaging the vectors of all words of that sentence. We will exploit this property to merge all the vectors of our normalization candidates for a position into one vector. This leads to the second question: How can we successfully exploit the top-N normalization candidates in a neural network dependency parser?

The main contributions of this chapter are:

- We show that using normalization as pre-processing improves performance on non-standard language for a neural network dependency parser. Even when the parser already exploits pre-trained embeddings and character level information, normalization leads to an increase in performance.
- We propose a novel technique to exploit the top-N candidates provided by the normalization component, and we show that this technique leads to a further increase in parser performance.
- A treebank containing non-standard language is created to evaluate the effect of normalization on parser performance. The treebank consists of 10,005 tokens annotated with lexical normalization and Universal Dependencies (Nivre et al., 2017).

This chapter is based on:

Rob van der Goot and Gertjan van Noord. Modeling input uncertainty in neural network dependency parsing. In *Proceedings of the 2018 Conference on Empirical Methods in Natural Language Processing*, pages 4984–4991, Brussels, Belgium, October 2018. Association for Computational Linguistics

The code and data (including the treebank) to reproduce the results of this chapter can be found on:

<https://bitbucket.org/robvander/normpar/>

## 8.1 Normalization Strategies

In this section, we will first shortly review the two models we will combine: a lexical normalization model and a neural network parser. Then, we describe how they can be combined.

### 8.1.1 Normalization

In this chapter we will use MoNoise (Chapter 4) as normalization model. Similar to the approach in the previous chapter, we convert the confidence score of the classifier of the normalization model into probabilities, so that we can exploit the top-N candidates.

We train MoNoise on the LiLiu dataset (Section 2.2.1), which only contains word-word replacements. In our initial experiments, we noted that the normalization model wrongfully normalized some words due to the different tokenization in the treebank (e.g. “ca n’t”), because these do not occur in the normalization data. We manually created a list of exceptions, which are ignored during normalization.

### 8.1.2 Neural Network Parser

In this chapter, we will use the shift-reduce UUParsers 2.0 (de Lhoneux et al., 2017b; Kiperwasser and Goldberg, 2016), which is a Bidirectional Long-Short Term Memory network (BiLSTM) (Graves and Schmidhuber, 2005) shift-reduce parser. We choose this parser for several reasons. Firstly, it reaches a competitive performance (Zeman et al., 2017). Secondly, it is a neural network parser which can exploit character level embeddings

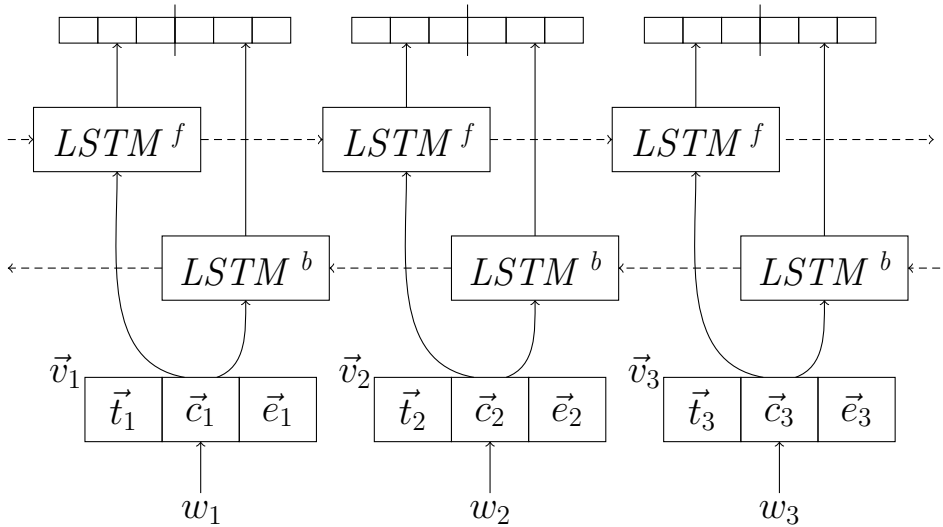


Figure 8.1: Overview of the conversion of input words to vectors which are used in the shift-reduce algorithm.

and external embeddings. Furthermore, it does not rely on a POS tagger, and the code-base is relatively compact and clean, which makes it easy to integrate normalization.

The UUParser 2.0 uses the Arc-Hybrid Transition system (Kuhlmann et al., 2011). Words are first converted to continuous vectors, which are then processed through a BiLSTM before they are used by the parsing algorithm. The decision whether to shift, reduce or swap (the swap action is added by (de Lhoneux et al., 2017b)) is made by a multi-layer perceptron with one hidden layer. The BiLSTM is trained jointly with the parsing objective, so that the vectors are optimized for the parsing task.

Figure 8.1 shows an overview of how the input words are converted to vectors which are used in the shift-reduce algorithm. We denote the vector used as input to the BiLSTM for word  $i$  by  $\vec{v}_i$ . This vector is a concatenation of three vectors which are derived from the input word.  $\vec{t}_i$  is optimized on the training data,  $\vec{c}_i$  is the result of a separate BiLSTM ran

over the characters of word  $i$  and  $\vec{e}_i$  is the external vector; it is obtained from external embeddings which are trained on huge amounts of raw texts. In this chapter we use the same word embeddings as used by the normalization model (Section 4.3.2).

### 8.1.3 Integration Strategy

The most straightforward way to use normalization to improve the parser is to normalize the input sentence before parsing it. However, the normalization model is not perfect, and mistakes are directly propagated to the parser in this setting. To mitigate this error propagation we exploit the top-N normalization candidates. As explained before, words are represented by continuous vectors in neural network parsers. This allows for a different type of integration compared to Chapter 7, since the vectors of all normalization candidates can be merged before parsing. This has some distinct advantages; the parsing algorithm needs no adaptation, and it is efficient, since this integration has no impact on the search space. Compared to the approach taken in Chapter 7, the main downside is that the parser does not yield a normalization sequence as output.

We will compare this integration with a parser which uses the best normalization sequence directly, a baseline which does not use normalization at all and a theoretical upper bound, which uses the manually annotated normalization as input to the parser. Below we will describe the details of each of these settings.

**Notation** We use  $\vec{w}_0 \dots \vec{w}_n$  to represent the vectors of the original words of a sentence. The vectors of the normalization candidates are represented by  $\vec{n}_{ij}$ , where  $i$  is the index of the word in the sentence, and  $j$  is the rank of the normalization candidate (starting from 0). The corresponding probability from the normalization model is  $p_{ij}$ . We use  $\vec{g}_i$  for the vector of the gold normalization of word  $i$ .

Our baseline setup is to simply use the vector of the original word:

$$\text{ORIG: } \vec{v}_i = \vec{w}_i$$

The most straightforward use of normalization is to use the best normalization sequence as input to the parser. In our setup, this means that

we use the vector of the highest ranked normalization candidate for each position:

$$\text{NORM: } \vec{v}_i = \vec{n}_{i0}$$

For our integrated approach, we merge the vectors of all normalization candidates for every position. MoNoise ranks all normalization candidates based on a confidence score, which we normalized to probabilities. We then weigh the vector of each candidate, and use the sum of all these weighted vectors as input for the parser. This is also called linear interpolation, and is calculated as follows:

$$\text{INTEGRATED: } \vec{v}_i = \sum_{j=0}^n p_{ij} * \vec{n}_{ij}$$

Finally, we include a theoretical upper bound of the effect of normalization, which uses manually annotated normalization:

$$\text{GOLD: } \vec{v}_i = \vec{g}_i$$

## 8.2 Data

To test the effect of normalization, we need a treebank containing non-standard language, preferably with a corresponding training treebank from a more standard domain. Since the existing treebanks are not noisy enough (Foster et al., 2011b; Kaljahi et al., 2015)<sup>1</sup> or do not have a corresponding training treebank in the same annotation format (Kong et al., 2014; Daiber and van der Goot, 2016) we annotate a small treebank for development and testing purposes<sup>2</sup>. We choose to use the Universal Dependencies 2.1 annotation format (Nivre et al., 2017), since the annotation efforts on the English Web Treebank (Section 3.3) provide suitable training data. This treebank already contains web specific phenomena like URL’s, E-Mail addresses and emoticons, so we do not have to create special annotation guidelines and the parser can learn these phenomena from the training data.

<sup>1</sup>Kaljahi et al. (2015) only normalize 3.6% of the words, and we manually normalized the development data from Foster et al. (2011b), were even fewer words were in need of normalization.

<sup>2</sup>It should be noted that two other suitable Twitter treebanks in the UD format were created in parallel to our treebank (Liu et al., 2018; Blodgett et al., 2018), which were released shortly after submission of this work.

Our treebank consists of data taken from the dataset by Li and Liu (2015) (also used in Chapter 6). The tweets in this dataset originate from two sources: the LexNorm corpus (Han and Baldwin, 2011), which was originally annotated with normalization, and a corpus originally annotated with POS tags (Owoputi et al., 2013). Li and Liu (2015) complemented this annotation for both datasets, so that they both have a normalization layer and a POS layer. To avoid overfitting on a specific filtering or time-frame we use the data collected by Owoputi et al. (2013) as development data and LexNorm as test data. We only keep the tweets which are still available on Twitter, resulting in a dataset of 305 development and 327 test tweets (10,005 tokens in total). It should be noted that these corpora were filtered to contain domain-specific phenomena and non-standard language, and thus provide an ideal testbed for our experiments but are not representative of the whole Twitter domain.

Tokenization and normalization are first re-annotated, because the Universal Dependencies format requires treebank specific tokenization. To avoid parser bias, dependency relations are annotated from scratch. For more details on annotation decisions for domain-specific structures, we refer to Appendix E.

On our development treebank, MoNoise reaches an error reduction rate of 0.46 for the normalization task. This score is lower compared to the scores on English data reported in Section 5.2, which is due to treebank specific tokenization. The splitting of words like ‘wanna’  $\mapsto$  ‘wan na’ leads to phenomena unknown to MoNoise (in our normalization annotation: ‘na’  $\mapsto$  ‘to’). In these cases, MoNoise tends to keep the original word. These phenomena are not problematic for the parser in most cases, since these also occur in the training treebank.

### 8.3 Evaluation

In this section, we first use the development data to compare the effect of the different normalization settings with the use of character level information and external embeddings. Secondly, we confirm our main results on the test set. Thirdly, we test if our model is sensitive to over-normalization on standard data. Finally, we perform some analysis to examine why normalization is beneficial.

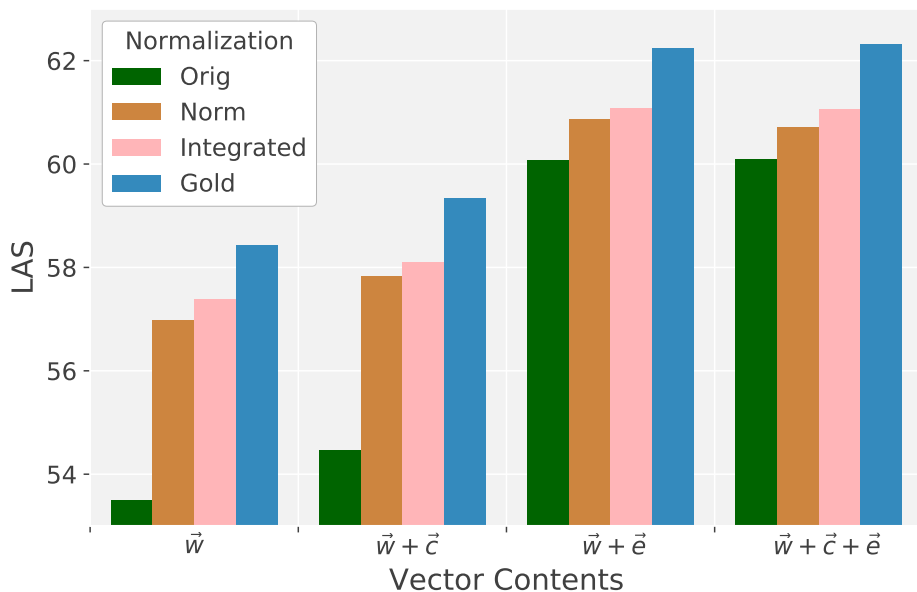


Figure 8.2: The effect of normalization on LAS for the different parsing models on the development data.

All scores reported in this section are obtained using the CoNLL 2017 evaluation script (Zeman et al., 2017). In Section 8.3.1 the results are the average over ten runs, using a different seed for the BiLSTM and the shuffling of the training data. In the remainder of this section, the best model is used to simplify interpretation. The parser is trained using default settings (de Lhoneux et al., 2017b).

In our initial experiments on the development data, it became apparent that the parser often considered a username mention or the retweet token ‘RT’ at the beginning of the tweet as root, resulting in a propagation of errors. The parser does not know how to handle these cases, because they do not occur in the training data. Because we want to exclude any influences from this simple construction, we added a heuristic to our parser which exclude usernames and ‘RT’ at the beginning of a tweet, and connects them to the root after parsing. We use this heuristic in all experiments.

Model	dev		test	
	UAS	LAS	UAS	LAS
ORIG	69.66	60.24	69.63	59.64
NORM	70.55*	61.36*	70.51	61.76*
INTEGRATED	70.60	61.56	70.62	62.30*
GOLD	71.76	63.11*	70.71	62.33

Table 8.1: UAS and LAS for the Twitter development and test treebanks. \*Statistically significant compared to the previous row at  $p < 0.05$  using a paired t-test.

### 8.3.1 Normalization Strategies

The results of the different parser and normalization settings on the development data are plotted in Figure 8.2. Using external embeddings ( $\vec{e}$ ) results in a much bigger performance improvement compared to using character level information ( $\vec{c}$ ). Adding character level embeddings on top of external embeddings only leads to a very minor improvement. This can partly be explained by the coverage of 98.4% of the embeddings on the development data.

In the settings without external embeddings, the direct use of normalization (NORM) results in an improvement of approximately 3 LAS points. However, when external embeddings are included the improvement becomes more than twice as small, indicating that the approaches target some common issues, but are also complementary to each other. When external embeddings and normalization are already used, the character level embeddings do not lead to higher performance. Integration of the normalization (INTEGRATED) consistently results in a slightly higher LAS compared to direct normalization. Interestingly, gold normalization still performs substantially better compared to automatic normalization.

### 8.3.2 Test Data

Table 8.1 shows the results of the parser with external embeddings and character embeddings (using the best seed from the development data), for the different normalization strategies on the development and test treebank.

These results confirm the main observations made before: normalization helps on top of external embeddings, and integrating normalization results in an even higher score. In contrast to the development data, the integrated approach almost reaches the theoretical upper bound of gold normalization on the test data. However, this is only the case on the test data, so this result should be interpreted with caution. The performance difference between the datasets is probably partly due to the differences in data selection. Interestingly, integrating normalization is especially beneficial for the LAS, meaning that it is most useful for choosing the type of relation.

### 8.3.3 Robustness

As stated in Section 8.2, our development and test data is selected to be very non-standard. However, it is undesirable to have a parser that performs bad on more canonical language, because in many real-world situations it is not known beforehand how many anomalies the input data contains. Hence, we also test performance on the English Web Treebank (EWT) development treebank. This dataset also consists of data from the web, however, it contains much less words in need of normalization; MoNoise normalizes less than 0.5% of all words. We compared the performance using no normalization (ORIG) versus our INTEGRATED approach, which showed a very minor performance improvement from 81.42 to 81.43 LAS. This was to be expected, since the EWT contains much less anomalies compared to our development and test treebanks. In these cases, MoNoise often gives high probabilities to the original word, and the parser will not be much affected.

### 8.3.4 Analysis

To gain insights into which constructions are parsed better when using normalization, we compared the predictions of the vanilla parser with our NORM and INTEGRATED methods on the development data. Starting with NORM, the first observation is that the incoming arcs of the words which are normalized are responsible for 44.1% of all improvements, whereas the outgoing arcs are responsible for 17.6% of all improvements. So, the direct context of the normalized words is responsible for only 61.7% of all improvements. We tried to identify trends by manually inspecting the dependency structures that were improved. However, it was hard to

find specific structures which were parsed better. This is because the normalization model normalizes a wide variety of words, which can be part of all types of syntactic structures. One clearly influential effect of using normalization, was that the parser improved on finding the root. When multiple unknown words occurred at the beginning of a sentence, the vanilla parser often failed at identifying the root, which improved considerably after normalizing.

For the INTEGRATED method, almost all the improvements made by NORM remained. On top of these, some additional improvements were made. Manual inspection revealed that these improvements often originated from a non-standard word, for which the correct normalization was ranked high. This then leads to improvements for the non-standard word as well as its context. In some cases, even incorrect normalization candidates lead to performance improvements. For example for ‘Gma’, where the normalization model ranked the original word first, but ‘mom’ second with a higher probability compared to the correct normalization ‘grandma’. Even though ‘grandma’ is the correct normalization, ‘mom’ occurs in similar contexts, and is a known word for the parser, leading to a better parse.

## 8.4 Conclusion

We showed that normalization can improve performance of a neural network parser, even when making use of character level information and external word embeddings. Integrating multiple normalization candidates into the parser led to an even larger performance increase. Normalization has shown to be complementary to external embeddings. However, if we add character embeddings on top of these, they do not add additional information. Our experiments revealed that our approach is robust, and it does not harm performance on more canonical data. On our test treebank, our integrated approach even performs on par with gold normalization. However, on the development treebank, gold normalization results in a much higher score, so we cannot draw strong conclusions about this yet.



**Part V**

**Conclusion**



## Chapter 9

# Summary and Conclusions

In this thesis, we focused on the problem of the parsing of non-standard language. Our approach to tackle this problem was to translate non-standard language into standard language before parsing it. This translation is also referred to as the task of normalization. Because most existing natural language processing systems, including parsers, are designed with standard texts in mind, normalizing a non-standard sentence before processing it can improve performance. We focused mainly on data from Twitter, because most annotated datasets consist of data collected from this microblog service.

We started this thesis by proposing a modular normalization model: MoNoise. This model is based on the observation that the normalization task consists of a variety of different types of replacements. For this reason, a variety of generation modules is designed, each targeting specific sub-set of the normalization problem. MoNoise tackles the problem of normalization in two steps. Firstly, a set of normalization candidates is generated for every word. Secondly, all of these candidates are ranked, and the most probable candidate is considered to be the best normalization. The most important modules for the candidate generation are a traditional spelling correction system, word embeddings and a translation dictionary generated from the training data. Ranking is done with a random forest classifier. This classifier uses features originating from the ranking as well as some additional features. Among the additional features, n-gram probabilities were by far the most predictive.

Intrinsic evaluation (Chapter 5) showed that MoNoise improves upon

---

the state-of-the-art on multiple benchmarks for a variety of Indo-European languages. When evaluating the separate steps of MoNoise separately, it became apparent that most mistakes are made in the ranking step. For most of these cases, the model ranked the original word first and the correct normalization second. This motivates our approaches in the later chapters, where we exploit multiple normalization candidates for each word to avoid error propagation.

As extrinsic evaluation, we tested the effect of MoNoise on a Bi-LSTM POS tagger for the social media domain. (Chapter 6) We compared the effect of normalization with the use of external embeddings. External embeddings showed a larger performance improvement compared to normalization, but using both methods simultaneously led to the best performance. We evaluated the effect of normalization in different settings. Constraining the normalization model to only normalize unknown words lead to a smaller performance gain, showing that normalization solves problems beyond replacing unknown words with known words. Furthermore, it was beneficial to train the tagger on a concatenation of the original training data and the normalized training data (since an idiosyncratic tagset is used, we also trained the POS tagger on tweets).

The first syntactic formalism we investigated was a constituency grammar. More concretely, we used MoNoise to normalize tweets, which are then parsed by the Berkeley parser. Firstly, we showed that the use of normalization as pre-processing is beneficial. Secondly, we introduced a method to integrate multiple normalization candidates into the parser. We do this by transforming the top-N normalization candidates to a word graph, and use a parsing algorithm to parse this word graph. In addition to a parse tree, the parser now also yields a syntactically motivated best path through the word graph. This approach is inspired by previous work on the parsing of speech lattices. The integration led to an even bigger performance improvement compared to the direct use of normalization. However, a qualitative error analysis showed that some of the improvements are due to incorrect normalization candidates which share syntactic properties with the original word. These end up in the final parse because they are known to the POS tagger and thus get a much higher POS probability.

Next to the constituency format, we also investigated the effect of normalization for a dependency grammar. In this setting, we made use of a neural

network parser. Neural network parsers can more naturally exploit character level information and external embeddings trained on large amounts of raw texts. These new approaches are expected to especially improve the processing of unknown words (words not occurring in the training treebank). Since these are also targeted by normalization, we first tested if normalization is still beneficial. Our results showed that normalization increases performance beyond the effect of external embeddings as well as character embeddings. However, as could be expected, the performance gained by normalization is smaller, especially when using external embeddings. Furthermore, we propose a simple method to exploit multiple normalization candidates per input word. This is done by weighting the vectors of each candidate based on the confidence score of MoNoise, and then merging the resulting vectors. This integration of the normalization leads to an even larger increase in performance.

To sum up, we first developed a state-of-the-art normalization model (MoNoise), which is based on a variety of modules, which each target a subset of the normalization problem. MoNoise is then used to improve performance on social media data for a POS tagger, a constituency parser, and a dependency parser. For both the constituency and the dependency parser, we introduced a novel method to exploit multiple normalization candidates per position during parsing, which led to an even higher performance in both cases.

However, for all of the aforementioned tasks, the performance gap compared to news texts is still large. Parser performance on news texts is higher than 90%, on small Twitter development and test treebanks. This performance dropped to 60%-66% (Section 7.4 and Section 8.3.2). Using normalization directly for the constituency parser resulted in an absolute performance improvement of 5%, whereas for the neural network dependency parser this was only 2% (because of external and character level embeddings). Integrating normalization led to a further increase of approximately 1% for our treebanks. Leading to scores between 62% and 74%. Comparison with the effect of manually annotated normalization gave somewhat mixed results. For the neural network dependency parsers, the scores are quite close, whereas for the constituency parsers there seems to be room for improvement for the normalization model.

This performance gap could be reduced further by expanding the nor-

malization task, to make the output even more similar to standard text (i.e. normalization beyond the word level). This has been done in some previous work, however, manually annotated normalization was often used. Normalization beyond the word level is a much harder task, and automatic systems will accordingly make more mistakes. Besides this, evaluation of this task is non-trivial.

Another way to improve parser performance would be to complement the integration of normalization with orthogonal methods. Indeed, normalization is not the only approach to adapt parsers to social media texts, other domain adaptation approaches can also be used, including up-training, transfer learning or domain specific embeddings.

# Appendix A

## Proof of Equivalence for Error Reduction Rate Formulas

The definitions of  $TP$ ,  $TN$ ,  $FP$ , and  $FN$  can be found in Section 5.1. For convenience, we introduce a variable for the total number of words:

$$all = TP + TN + FP + FN \quad (\text{A.1})$$

As discussed in Section 5.1, the accuracy of a normalization system can be calculated with:

$$accuracy_{system} = \frac{TP + TN}{all} \quad (\text{A.2})$$

For a baseline system, which always returns the original sequence, the accuracy can be calculated like this (note that we use the variable values from our system here):

$$accuracy_{base} = \frac{TN + FP}{all} \quad (\text{A.3})$$

This is equal to the percentage of words which are not in need of normalization.

Next, we will show the derivation which shows how to rewrite the error reduction rate formula:

$$ERR = \frac{TP - FP}{TP + FN} \quad (\text{A.4})$$

$$= \frac{TP - FP}{all - (TN + FP)} \quad (\text{A.5})$$

$$= \frac{(TP + TN) - (TN + FP)}{all - (TN + FP)} \quad (\text{A.6})$$

$$= \frac{\frac{TP+TN}{all} - \frac{TN+FP}{all}}{\frac{all}{all} - \frac{TN+FP}{all}} \quad (\text{A.7})$$

$$= \frac{\frac{TP+TN}{all} - \frac{TN+FP}{all}}{1 - \frac{TN+FP}{all}} \quad (\text{A.8})$$

$$= \frac{acc_{system} - acc_{base}}{1 - acc_{base}} \quad (\text{A.9})$$

## Appendix B

# Relation Between Error Reduction Rate and Distance in ROC Space

The distance between a point and a line is give as:

$$distance(ax + by + c = 0, (x_0, y_0)) = \frac{|ax_0 + by_0 + c|}{\sqrt{a^2 + b^2}} \quad (\text{B.1})$$

The formula for the ROC line is then:

$$\begin{aligned} a = +1 \quad b = -1 \quad x = TPR \quad y = FPR \\ ax + by + c = TPR - FPR \end{aligned} \quad (\text{B.2})$$

So the distance to the ROC line (informedness) is:

$$informedness = \frac{TPR - FPR}{\sqrt{2}} = \frac{\frac{TP}{TP+FN} - \frac{FP}{FP+TN}}{1.41} \quad (\text{B.3})$$

We got rid of the absolute in the numerator, since a negative distance means you perform worse than baseline, which is a desirable distinction for our use.

$$ERR = \frac{TP - FP}{TP + FN} \quad (\text{B.4})$$

$$informedness = \frac{\frac{TP}{TP+FN} - \frac{FP}{FP+TN}}{1.41} \quad (\text{B.5})$$

$$(\text{B.6})$$

Actually, we also could ignore the denominator in the ROCdist, as this is a linear transformation, normalizing for the distance between the line and the maximum performance. Without this normalization, the resulting values will be in a more similar range as ERR.

$$ERR = \frac{TP - FP}{TP + FN} \quad (\text{B.7})$$

$$ERR = \frac{TP}{TP + FN} - \frac{FP}{TP + FN} \quad (\text{B.8})$$

$$informedness = \frac{TP}{TP + FN} - \frac{FP}{FP + TN} \quad (\text{B.9})$$

The only difference that remains is the denominator of the second fraction. In the ERR, we divide by the total number of positives, whereas the informedness divides by the total number of negatives. Since these numbers are equal within a dataset, the ERR and ROCdist have a perfect correlation there. Cross-datasets they should have a very high correlation.

Thanks to Hessel and Ahmet for their help with these formulas.

## Appendix C

# Results of MoNoise On Multiple Normalization Evaluation Metrics

The results reported in this appendix are on the test data, and are based on a normalization model trained on a concatenation of the training and the development data. It should be noted that slightly higher scores can be obtained by tuning towards a specific metric. Below, we will first explain metrics which have not been discussed in detail in Section 5.1, the table containing the scores can be found on the next page.

- `accGoldErr`: The accuracy which would be obtained if we had a perfect (gold) error detection.
- `accOracle`: Accuracy which would be obtained with an oracle ranking.
- `precision`: This is the precision calculate in the wrong way (see for more info Section 5.1, page 76).
- `realPrec`: This is the correctly calculated precision.
- `realF1`: F1 score based on `realPrec`.
- `EDprec`: precision of error detection task
- `EDrec`: recall of error detection task
- `EDf1`: f1 of error detection task

Language	GhentNorm	TweetNorm	LexNorm1.2	LexNorm2015	James-Norm	ReLDI-hr	ReLDI-sr
	NL	ES	EN	EN	SL	HR	SR
E <sub>RR</sub>	44.62	35.86	60.61	76.15	67.15	51.73	57.48
accBase	97.55	92.65	88.45	90.56	84.90	85.48	90.08
accuracy	98.64	95.28	95.45	97.75	95.04	92.99	95.78
accGoldErr	99.25	97.73	98.48	98.93	96.99	95.69	97.24
accOracle	99.62	98.34	99.23	99.35	98.32	96.90	98.02
precision	86.84	90.05	78.03	91.98	89.62	92.17	86.43
recall	50.77	37.09	79.12	80.58	70.81	54.23	60.78
F1	64.08	52.54	78.57	85.91	79.11	68.29	71.37
realPrec	89.19	96.79	81.04	94.79	95.09	95.59	94.85
realF1	64.71	53.63	80.07	87.11	81.17	69.20	74.09
WER	1.36	4.72	4.55	2.25	4.96	7.01	4.22
CER	0.56	2.54	2.17	1.12	1.83	3.02	2.08
EDDprec	89.47	97.01	81.74	94.94	95.37	95.75	95.31
EDDrec	52.31	39.96	82.88	83.18	75.35	56.34	67.02
EDF1	66.02	56.60	82.31	88.67	84.19	70.94	78.70
TP	33	181	966	2237	1606	1236	1032
FN	32	307	255	539	662	1043	666
FP	4	6	226	123	83	57	56
FN	32	307	255	539	662	1043	666

## Appendix D

# Overview of Twitter-specific POS tags

Taken from Owoputi et al. (2013). See for more details: Gimpel et al. (2013).

N	common noun
O	pronoun (personal/WH; not possessive)
^	proper noun
S	nominal + possessive
Z	proper noun + possessive
V	verb including copula, auxiliaries
L	nominal + verbal (e.g. im), verbal + nominal (lets)
M	proper noun + verbal
A	adjective
R	adverb
!	interjection
D	determiner
P	pre- or postposition, or subordinating conjunction
&	coordinating conjunction
T	verb particle
X	existential there, predeterminers
Y	X + verbal
#	hashtag (indicates topic/category for tweet)
@	at-mention (indicates a user as a recipient of a tweet)

- ~ discourse marker, indications of continuation across multiple tweets
- U URL or email address
- E emoticon
- \$ numeral
- , punctuation
- G other abbreviations, foreign words, possessive endings, symbols, garbage

## Appendix E

# Annotation Guidelines for Universal Dependencies Annotation for Tweets

In this appendix we will give a short overview of annotation decisions. The Universal Dependencies English Web Treebank 2.1 (Silveira et al., 2014; Nivre et al., 2017) annotations are used as guidance for most annotation decisions. Since the texts in this treebank are sampled from several web domains, it already covers most phenomena occurring in Twitter data.

### E.1 Tokenization

As a starting point, we used the tokenization from the previous annotation (Li and Liu, 2015). On top of this, we ran a simple rule-based tokenizer to make the data better suitable for syntactic annotation. Phrasal abbreviations (e.g. lol, smh) are treated as one token. We also included the normalization from the original corpora in the MISC column, which is manually corrected after tokenization (see Figure E.1).

```
# text = damn im finna roll up again...
1   damn   Norm=damn
2   i      SpaceAfter=No;Norm=I
3   m      Norm=am
4   fin    SpaceAfter=No;Norm=going
5   na     Norm=to
6   roll   Norm=roll
7   up     Norm=up
8   again  Norm=again
9   ...    Norm=...
```

Figure E.1: An example of tokenization in the CoNLL-U format (Only the ‘ID’, ‘FORM’ and ‘MISC’ column are shown here)

No sentence segmentation is performed on the input data because the Tweet-unit is inherent to this domain. Instead, we use the **parataxis** relation to connect different utterances. The head of the first utterance is always the root, and all next utterance are dependents of this node, see Figure E.2 for an example.

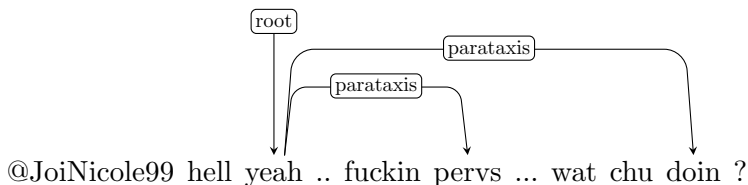


Figure E.2: Annotation of the sentence “@JoiNicole99 hell yeah..fuckin pervs...watchu doin?” , only relevant relations are shown.

## E.2 POS tags

Our parser does not make use of POS tags, but because they were already annotated and are closely related to the choice of dependency relations we corrected them during annotation. POS tags were first automatically mapped to universal tags (Petrov et al., 2012) and then manually corrected.

### E.3 Unknown Words

If the annotator is unsure about the meaning of a word, other tweets containing the same word are searched and where necessary [www.urbandictionary.com](http://www.urbandictionary.com) is consulted. If the annotator still could not understand the word, it is annotated as X with the `dep` relation. This only occurs five times in our data.

### E.4 Emoticons, Emojis, URL's and Phrasal Abbreviations

Since words belonging to this category are often not syntactically connected, we annotate them as dependant of the head of the nearest utterance (see E.1). The relations and POS tags used are similar to the English Web Treebank: emoticons and emojis are a SYMB connected with relation `discourse`, URL's are annotated as X with relation `appos` and phrasal abbreviations like 'lol' and 'smh' are considered to be an INTJ with the `discourse` relation.

### E.5 Domain Specific Tokens

Mentions are used in Twitter to direct tweets towards a specific person or account. They consist of the '@' symbol followed by the targeted username. Because mentions are used to focus a Tweet to a specific user we annotated it as PROP<sub>N</sub> with the relation `vocative`.

Hashtags are used to specify the topic or mood of the Tweet. They are often located at the end of the Tweet. Their usage is similar to interjections in the English Web Treebank, so they are annotated accordingly as INTJ and `discourse`.

A retweet is indicated by the token 'RT', which is usually found at the beginning of the Tweet. We tag it with the X tag and the `discourse` relation.

Because these phenomena are often not syntactically connected to the sentence, we connect them to the root. Note that all of these phenomena can also be used in (syntactic) context, then they are annotated accordingly (see example in FigureE.3).

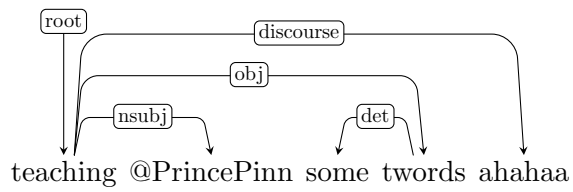


Figure E.3: Annotation of the sentence “teaching @PrincePinn some twords ahahaa”

# Bibliography

- Md Shad Akhtar, Utpal Kumar Sikdar, and Asif Ekbal. IITP: Hybrid approach for text normalization in twitter. In *Proceedings of the Workshop on Noisy User-generated Text*, pages 106–110, Beijing, China, July 2015. Association for Computational Linguistics.
- Inaki Alegria, Nora Aranberri, Víctor Fresno, Pablo Gamallo, Lluís Padró, Inaki San Vicente, Jordi Turmo, and Arkaitz Zubiaga. Introducción a la tarea compartida Tweet-Norm 2013: Normalización léxica de tuits en español. In *Tweet-Norm@ SEPLN*, pages 1–9, 2013.
- Sanjeev Arora, Yingyu Liang, and Tengyu Ma. A simple but tough-to-beat baseline for sentence embeddings. In *5th International Conference on Learning Representations*, 2016.
- AiTi Aw, Min Zhang, Juan Xiao, and Jian Su. A phrase-based statistical model for sms text normalization. In *Proceedings of the COLING/ACL 2006 Main Conference Poster Sessions*, pages 33–40. Association for Computational Linguistics, 2006.
- Timothy Baldwin, Marie-Catherine de Marneffe, Bo Han, Young-Bum Kim, Alan Ritter, and Wei Xu. Shared tasks of the 2015 workshop on noisy user-generated text: Twitter lexical normalization and named entity recognition. In *Proceedings of the Workshop on Noisy User-generated Text*, pages 126–135, Beijing, China, July 2015a. Association for Computational Linguistics.
- Timothy Baldwin, Marie-Catherine de Marneffe, Bo Han, Young-Bum Kim, Alan Ritter, and Wei Xu. Guideline for English lexical normalisation

- shared task. Technical report, Workshop on Noisy User-generated Text, 2015b.
- Tyler Baldwin and Yunyao Li. An in-depth analysis of the effect of text normalization in social media. In *Proceedings of the 2015 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies*, pages 420–429, Denver, Colorado, May 2015. Association for Computational Linguistics.
- Miguel Ballesteros, Chris Dyer, and Noah A. Smith. Improved transition-based parsing by modeling characters instead of words with LSTMs. In *Proceedings of the 2015 Conference on Empirical Methods in Natural Language Processing*, pages 349–359, Lisbon, Portugal, September 2015. Association for Computational Linguistics.
- Yehoshua Bar-Hillel, Micha Perles, and Eliahu Shamir. On formal properties of simple phrase structure grammars. *Sprachtypologie und Universalienforschung*, 14:143–172, 1961.
- Madeleine Bates. The use of syntax in a speech understanding system. *IEEE Transactions on Acoustics, Speech, and Signal Processing*, 23(1): 112–117, February 1975.
- Yevgeni Berzak, Yan Huang, Andrei Barbu, Anna Korhonen, and Boris Katz. Anchoring and agreement in syntactic annotations. In *Proceedings of the 2016 Conference on Empirical Methods in Natural Language Processing*, pages 2215–2224, Austin, Texas, November 2016. Association for Computational Linguistics.
- Steven Bethard. A synchronous context free grammar for time normalization. In *Proceedings of the 2013 Conference on Empirical Methods in Natural Language Processing*, pages 821–826, Seattle, Washington, USA, October 2013. Association for Computational Linguistics.
- Ann Bies, Mark Ferguson, Karen Katz, Robert MacIntyre, Victoria Tredinick, Grace Kim, Mary Ann Marcinkiewicz, and Britta Schasberger. Bracketing guidelines for Treebank II style Penn Treebank project. Technical report, University of Pennsylvania, 1995.

- Ann Bies, Justin Mott, Colin Warner, and Seth Kulick. Bracketing webtext: An addendum to Penn Treebank II guidelines. Technical report, Linguistic Data Consortium, 2012.
- Su Lin Blodgett, Johnny Wei, and Brendan O'Connor. Twitter universal dependency parsing for African-American and mainstream American English. In *Proceedings of the 56th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 1415–1425, Melbourne, Australia, 2018. Association for Computational Linguistics.
- Marcel Bollmann. (semi-)automatic normalization of historical texts using distance measures and the norma tool. In *Proceedings of the Second Workshop on Annotation of Corpora for Research in the Humanities (ACRH-2), Lisbon, Portugal*, 2012.
- Marcel Bollmann, Joachim Bingel, and Anders Søgaard. Learning attention for historical text normalization by learning to pronounce. In *Proceedings of the 55th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 332–344, Vancouver, Canada, July 2017. Association for Computational Linguistics.
- Taylor L. Booth and Richard A. Thompson. Applying probability measures to abstract languages. *IEEE transactions on Computers*, 5:442–450, 1973.
- Leo Breiman. Random forests. *Machine learning*, 45(1):5–32, 2001.
- Peter F. Brown, Peter V. Desouza, Robert L. Mercer, Vincent J. Della Pietra, and Jenifer C. Lai. Class-based n-gram models of natural language. *Computational linguistics*, 18(4):467–479, 1992.
- Eugene Charniak and Mark Johnson. Coarse-to-fine n-best parsing and maxent discriminative reranking. In *Proceedings of the 43rd Annual Meeting of the Association for Computational Linguistics (ACL'05)*, pages 173–180, Ann Arbor, Michigan, June 2005. Association for Computational Linguistics.
- Danqi Chen and Christopher Manning. 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, Doha, Qatar, October 2014. Association for Computational Linguistics.
- Noam Chomsky. Three models for the description of language. *IRE Transactions on information theory*, 2(3):113–124, 1956.
- Noam Chomsky. On certain formal properties of grammars. *Information and control*, 2(2):137–167, 1959.
- Grzegorz Chrupała. Normalizing tweets with edit scripts and recurrent neural embeddings. In *Proceedings of the 52nd Annual Meeting of the Association for Computational Linguistics (Volume 2: Short Papers)*, pages 680–686, Baltimore, Maryland, June 2014. Association for Computational Linguistics.
- John Cocke. *Programming Languages and Their Compilers: Preliminary Notes*. New York University, New York, NY, USA, 1969.
- Jacob Cohen. A coefficient of agreement for nominal scales. *Educational and psychological measurement*, 20(1):37–46, 1960.
- Jacob Cohen. Weighted Kappa: Nominal scale agreement provision for scaled disagreement or partial credit. *Psychological bulletin*, 70(4), 1968.
- Michael Collins. Three generative, lexicalised models for statistical parsing. In *Proceedings of the 35th Annual Meeting of the Association for Computational Linguistics*, pages 16–23, Madrid, Spain, July 1997. Association for Computational Linguistics.
- Michael Collins. Head-driven statistical models for natural language parsing. *Computational linguistics*, 29(4):589–637, 2003.
- Matthieu Constant, Joseph Le Roux, and Anthony Sigogne. Combining compound recognition and PCFG-LA parsing with word lattices and conditional random fields. *ACM Transactions on Speech and Language Processing (TSLP)*, 10(3), 2013.
- Michael A. Covington. A fundamental algorithm for dependency parsing. In *Proceedings of the 39th annual ACM southeast conference*, pages 95–102, 2001.

- Joachim Daiber and Rob van der Goot. The Denoised Web Treebank: Evaluating dependency parsing under noisy input conditions. In *Proceedings of the Tenth International Conference on Language Resources and Evaluation (LREC 2016)*, Portoro, Slovenia, May 2016. European Language Resources Association (ELRA).
- Orphée De Clercq, Bart Desmet, and Véronique Hoste. Guidelines for normalizing Dutch and English user generated content. Technical report, Ghent University, 2014a.
- Orphée De Clercq, Sarah Schulz, Bart Desmet, and Véronique Hoste. Towards shared datasets for normalization research. In *Proceedings of the Ninth International Conference on Language Resources and Evaluation (LREC'14)*, Reykjavik, Iceland, May 2014b. European Language Resources Association (ELRA).
- Miryam de Lhoneux, Yan Shao, Ali Basirat, Eliyahu Kiperwasser, Sara Stymne, Yoav Goldberg, and Joakim Nivre. From raw text to universal dependencies – look, no tags! In *Proceedings of the CoNLL 2017 Shared Task: Multilingual Parsing from Raw Text to Universal Dependencies.*, Vancouver, Canada, 2017a.
- Miryam de Lhoneux, Sara Stymne, and Joakim Nivre. Arc-hybrid non-projective dependency parsing with a static-dynamic oracle. In *Proceedings of the The 15th International Conference on Parsing Technologies (IWPT).*, Pisa, Italy, 2017b.
- Leon Derczynski, Alan Ritter, Sam Clark, and Kalina Bontcheva. Twitter part-of-speech tagging for all: Overcoming sparse and noisy data. In *Proceedings of the International Conference Recent Advances in Natural Language Processing RANLP 2013*, pages 198–206, Hissar, Bulgaria, September 2013. INCOMA Ltd. Shoumen, Bulgaria.
- Chris Dyer, Miguel Ballesteros, Wang Ling, Austin Matthews, and Noah A. Smith. Transition-based dependency parsing with stack long short-term memory. In *Proceedings of the 53rd Annual Meeting of the Association for Computational Linguistics and the 7th International Joint Conference on Natural Language Processing (Volume 1: Long Papers)*, pages 334–343, Beijing, China, July 2015. Association for Computational Linguistics.

- Jacob Eisenstein. What to do about bad language on the internet. In *Proceedings of the 2013 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies*, pages 359–369, Atlanta, Georgia, June 2013. Association for Computational Linguistics.
- Tomaz Erjavec, Darja Fišer, Jaka Čibej, Špela Arhar Holdt, Nikola Ljubešić, and Katja Zupan. CMC training corpus janex-tag 2.0, 2017.
- Mariano Felice and Zheng Yuan. Generating artificial errors for grammatical error correction. In *Proceedings of the Student Research Workshop at the 14th Conference of the European Chapter of the Association for Computational Linguistics*, pages 116–126, Gothenburg, Sweden, April 2014. Association for Computational Linguistics.
- Joseph L Fleiss. Measuring nominal scale agreement among many raters. *Psychological bulletin*, 76(5), 1971.
- Emma Flint, Elliot Ford, Olivia Thomas, Andrew Caines, and Paula Buttery. A text normalisation system for non-standard english words. In *Proceedings of the 3rd Workshop on Noisy User-generated Text*, pages 107–115. Association for Computational Linguistics, 2017.
- Jennifer Foster. “cba to check the spelling”: Investigating parser performance on discussion forum posts. In *Human Language Technologies: The 2010 Annual Conference of the North American Chapter of the Association for Computational Linguistics*, pages 381–384, Los Angeles, California, June 2010. Association for Computational Linguistics.
- Jennifer Foster and Øistein E. Andersen. GenERRate: Generating errors for use in grammatical error detection. In *Proceedings of the Fourth Workshop on Innovative Use of NLP for Building Educational Applications, EdAppsNLP '09*, pages 82–90, Stroudsburg, PA, USA, 2009. Association for Computational Linguistics.
- Jennifer Foster, Özlem Çetinoglu, Joachim Wagner, Joseph Le Roux, Stephen Hogan, Joakim Nivre, Deirdre Hogan, and Josef Van Genabith. #hardtoparse: POS Tagging and parsing the Twitterverse. In *AAAI 2011 Workshop On Analyzing Microtext*, pages 20–25, United States, 2011a.

- Jennifer Foster, Ozlem Cetinoglu, Joachim Wagner, Joseph Le Roux, Joakim Nivre, Deirdre Hogan, and Josef van Genabith. From news to comment: Resources and benchmarks for parsing the language of web 2.0. In *Proceedings of 5th International Joint Conference on Natural Language Processing*, pages 893–901, Chiang Mai, Thailand, November 2011b. Asian Federation of Natural Language Processing.
- David C. Gibbon and Zhu Liu. *Introduction to video search engines*. Springer Science & Business Media, 2008.
- Richard A. Gillman. Automatic verification of hypothesized phonemic strings in continuous speech. Technical report, Advanced Research Projects Agency, May 1974.
- Kevin Gimpel, Nathan Schneider, Brendan O’Connor, Dipanjan Das, Daniel Mills, Jacob Eisenstein, Michael Heilman, Dani Yogatama, Jeffrey Flanigan, and Noah A. Smith. Part-of-speech tagging for Twitter: Annotation, features, and experiments. In *Proceedings of the 49th Annual Meeting of the Association for Computational Linguistics: Human Language Technologies*, pages 42–47, Portland, Oregon, USA, June 2011. Association for Computational Linguistics.
- Kevin Gimpel, Nathan Schneider, and Brendan O’Connor. Annotation guidelines for twitter part-of-speech tagging. Technical report, Carnegie Mellon University, 2013.
- Yoav Goldberg and Michael Elhadad. An efficient algorithm for easy-first non-directional dependency parsing. In *Human Language Technologies: The 2010 Annual Conference of the North American Chapter of the Association for Computational Linguistics*, pages 742–750, Los Angeles, California, June 2010. Association for Computational Linguistics.
- Yoav Goldberg and Michael Elhadad. Joint Hebrew segmentation and parsing using a PCFGLA lattice parser. In *Proceedings of the 49th Annual Meeting of the Association for Computational Linguistics: Human Language Technologies*, pages 704–709, Portland, Oregon, USA, June 2011. Association for Computational Linguistics.

- Kyle Gorman and Richard Sproat. Minimally supervised number normalization. *Transactions of the Association for Computational Linguistics*, 4: 507–519, 2016.
- Alex Graves and Jürgen Schmidhuber. Framewise phoneme classification with bidirectional LSTM and other neural network architectures. *Neural Networks*, 18(5-6):602–610, 2005.
- Bo Han and Timothy Baldwin. Lexical normalisation of short text messages: Mkn sens a #twitter. In *Proceedings of the 49th Annual Meeting of the Association for Computational Linguistics: Human Language Technologies*, pages 368–378, Portland, Oregon, USA, June 2011. Association for Computational Linguistics.
- Zellig S. Harris. Distributional structure. *Word*, 10(2-3):146–162, 1954.
- Hany Hassan and Arul Menezes. Social text normalization using contextual graph random walks. In *Proceedings of the 51st Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 1577–1586, Sofia, Bulgaria, August 2013. Association for Computational Linguistics.
- Kokil Jaidka, Niyati Chhaya, and Lyle Ungar. Diachronic degradation of language models: Insights from social media. In *Proceedings of the 56th Annual Meeting of the Association for Computational Linguistics (Volume 2: Short Papers)*, pages 195–200, Melbourne, Australia, 2018. Association for Computational Linguistics.
- Ning Jin. NCSU-SAS-Ning: Candidate generation and feature engineering for supervised lexical normalization. In *Proceedings of the Workshop on Noisy User-generated Text*, pages 87–92, Beijing, China, July 2015. Association for Computational Linguistics.
- Mark Johnson. PCFG models of linguistic tree representations. *Computational Linguistics*, 24(4):613–632, 1998.
- Lucy Jones. The changing face of spelling on the internet. Technical report, The English Spelling Society, 2010.

- Rasoul Kaljahi, Jennifer Foster, Johann Roturier, Corentin Ribeyre, Teresa Lynn, and Joseph Le Roux. Foreebank: Syntactic analysis of customer support forums. In *Proceedings of the 2015 Conference on Empirical Methods in Natural Language Processing*, pages 1341–1347, Lisbon, Portugal, September 2015. Association for Computational Linguistics.
- Tadao Kasami. An efficient recognition and syntax-analysis algorithm for context-free languages. *Coordinated Science Laboratory Report no. R-257*, 1966.
- Eliyahu Kiperwasser and Yoav Goldberg. Simple and accurate dependency parsing using bidirectional LSTM feature representations. *TACL*, 4: 313–327, 2016.
- Dan Klein and Christopher D. Manning. Accurate unlexicalized parsing. In *Proceedings of the 41st Annual Meeting of the Association for Computational Linguistics*, pages 423–430, Sapporo, Japan, July 2003. Association for Computational Linguistics.
- Lingpeng Kong, Nathan Schneider, Swabha Swayamdipta, Archana Bhatia, Chris Dyer, and Noah A. Smith. A dependency parser for tweets. In *Proceedings of the 2014 Conference on Empirical Methods in Natural Language Processing (EMNLP)*, pages 1001–1012, Doha, Qatar, October 2014. Association for Computational Linguistics.
- Natalia Korchagina. Normalizing medieval german texts: from rules to deep learning. In *Proceedings of the NoDaLiDa 2017 Workshop on Processing Historical Language*, pages 12–17. Linköping University Electronic Press, 2017.
- Marco Kuhlmann, Carlos Gómez-Rodríguez, and Giorgio Satta. Dynamic programming algorithms for transition-based dependency parsers. In *Proceedings of the 49th Annual Meeting of the Association for Computational Linguistics: Human Language Technologies*, pages 673–682, Portland, Oregon, USA, June 2011. Association for Computational Linguistics.
- Karen Kukich. Techniques for automatically correcting words in text. *Acm Computing Surveys (CSUR)*, 24(4):377–439, 1992.

- Bernard Lang. A generative view of ill-formed input processing. *ATR Symposium on Basic Research for Telephone Interpretation (ASTI)*, 1989.
- David Lee. Genres, registers, text types, domains, and styles: Clarifying the concepts and navigating a path through the BNC jungle. *Language Learning and Technology*, 5, January 2002.
- Vladimir I Levenshtein. Binary codes capable of correcting deletions, insertions, and reversals. In *Soviet physics doklady*, volume 10, pages 707–710, 1966.
- Roger Levy. A noisy-channel model of human sentence comprehension under uncertain input. In *Proceedings of the 2008 Conference on Empirical Methods in Natural Language Processing*, pages 234–243, Honolulu, Hawaii, October 2008. Association for Computational Linguistics.
- Chen Li and Yang Liu. Improving text normalization using character-blocks based models and system combination. In *Proceedings of COLING 2012*, pages 1587–1602, Mumbai, India, December 2012.
- Chen Li and Yang Liu. Improving text normalization via unsupervised model and discriminative reranking. In *Proceedings of the ACL 2014 Student Research Workshop*, pages 86–93, Baltimore, Maryland, USA, June 2014. Association for Computational Linguistics.
- Chen Li and Yang Liu. Joint POS tagging and text normalization for informal text. In *Proceedings of the Twenty-Fourth International Joint Conference on Artificial Intelligence, IJCAI 2015, Buenos Aires, Argentina, July 25-31, 2015*, pages 1263–1269, 2015.
- Chu-Cheng Lin, Waleed Ammar, Chris Dyer, and Lori Levin. Unsupervised pos induction with word embeddings. In *Proceedings of the 2015 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies*, pages 1311–1316, Denver, Colorado, May 2015. Association for Computational Linguistics.
- Wang Ling, Chris Dyer, Alan W Black, and Isabel Trancoso. Two/too simple adaptations of word2vec for syntax problems. In *Proceedings of the 2015 Conference of the North American Chapter of the Association*

- for Computational Linguistics: Human Language Technologies*, pages 1299–1304, Denver, Colorado, May 2015. Association for Computational Linguistics.
- Fei Liu, Fuliang Weng, and Xiao Jiang. A broad-coverage normalization system for social media language. In *Proceedings of the 50th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 1035–1044, Jeju Island, Korea, July 2012. Association for Computational Linguistics.
- Yijia Liu, Yi Zhu, Wanxiang Che, Bing Qin, Nathan Schneider, and Noah A. Smith. Parsing tweets into Universal Dependencies. In *Proceedings of the 2018 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, Volume 1 (Long Papers)*, pages 965–975, New Orleans, Louisiana, 2018. Association for Computational Linguistics.
- Nikola Ljubešić, Katja Zupan, Darja Fišer, and Tomaz Erjavec. Normalising Slovene data: historical texts vs. user-generated content. *Bochumer Linguistische Arbeitsberichte*, 2016.
- Nikola Ljubešić, Tomaž Erjavec, Maja Miličević, and Tanja Samardžić. Croatian Twitter training corpus ReLDI-NormTagNER-hr 2.0, 2017a.
- Nikola Ljubešić, Tomaž Erjavec, Maja Miličević, and Tanja Samardžić. Serbian Twitter training corpus ReLDI-NormTagNER-sr 2.0, 2017b.
- Nikola Ljubešić and Filip Klubička. bs,hr,srwac - web corpora of bosnian, croatian and serbian. In *Proceedings of the 9th Web as Corpus Workshop (WaC-9)*, pages 29–35, Gothenburg, Sweden, April 2014. Association for Computational Linguistics.
- Christopher D. Manning. Part-of-speech tagging from 97% to 100%: is it time for some linguistics? In *International conference on intelligent text processing and computational linguistics*, pages 171–189. Springer, 2011.
- Mitchell P. Marcus, Mary Ann Marcinkiewicz, and Beatrice Santorini. Building a large annotated corpus of English: The Penn Treebank. *Computational linguistics*, 19(2):313–330, 1993.

- Valerie Ruth Mariana. The Multidimensional Quality Metric (MQM) framework: A new framework for translation quality assessment. *The Journal of Specialised Translation*, 2014.
- Takuya Matsuzaki, Yusuke Miyao, and Jun'ichi Tsujii. Probabilistic CFG with latent annotations. In *Proceedings of the 43rd Annual Meeting of the Association for Computational Linguistics (ACL'05)*, pages 75–82, Ann Arbor, Michigan, June 2005. Association for Computational Linguistics.
- Tomas Mikolov, Kai Chen, Greg Corrado, and Jeffrey Dean. Efficient estimation of word representations in vector space. *Proceedings of Workshop at ICLR*, 2013a.
- Tomas Mikolov, Ilya Sutskever, Kai Chen, Greg S Corrado, and Jeff Dean. Distributed representations of words and phrases and their compositionality. In *Advances in Neural Information Processing Systems 26*, pages 3111–3119. Curran Associates, Inc., 2013b.
- Alejandro Mosquera, Elena Lloret, and Paloma Moreda. Towards facilitating the accessibility of web 2.0 texts through text normalisation. In *Proceedings of the LREC workshop: Natural Language Processing for Improving Textual Accessibility (NLP4ITA)*, pages 9–14, 2012.
- Hwee Tou Ng, Siew Mei Wu, Ted Briscoe, Christian Hadiwinoto, Raymond Hendy Susanto, and Christopher Bryant. The CoNLL-2014 shared task on grammatical error correction. In *Proceedings of the Eighteenth Conference on Computational Natural Language Learning: Shared Task*, pages 1–14, Baltimore, Maryland, June 2014. Association for Computational Linguistics.
- Dat Quoc Nguyen, Mark Dras, and Mark Johnson. A novel neural network model for joint POS tagging and graph-based dependency parsing. In *Proceedings of the CoNLL 2017 Shared Task: Multilingual Parsing from Raw Text to Universal Dependencies*, pages 134–142, 2017.
- Malvina Nissim, Lasha Abzianidze, Kilian Evang, Rob van der Goot, Hessel Haagsma, Barbara Plank, and Martijn Wieling. Sharing is caring: The future of shared tasks. *Computational Linguistics*, 43(4):897–904, 2017.

- Joakim Nivre. Incrementality in deterministic dependency parsing. In *Proceedings of the ACL Workshop Incremental Parsing: Bringing Engineering and Cognition Together*, pages 50–57, Barcelona, Spain, July 2004. Association for Computational Linguistics.
- Joakim Nivre. Non-projective dependency parsing in expected linear time. In *Proceedings of the Joint Conference of the 47th Annual Meeting of the ACL and the 4th International Joint Conference on Natural Language Processing of the AFNLP*, pages 351–359, Suntec, Singapore, August 2009. Association for Computational Linguistics.
- Joakim Nivre, Zeljko Agić, Lars Ahrenberg, Lene Antonsen, Maria Jesus Aranzabe, Masayuki Asahara, Luma Ateyah, Mohammed Attia, Aitziber Atutxa, Liesbeth Augustinus, Elena Badmaeva, Miguel Ballesteros, Esha Banerjee, Sebastian Bank, Verginica Barbu Mititelu, John Bauer, Kepa Bengoetxea, Riyaz Ahmad Bhat, Eckhard Bick, Victoria Bobicev, Carl Börstell, Cristina Bosco, Gosse Bouma, Sam Bowman, Aljoscha Burchardt, Marie Candito, Gauthier Caron, Gülşen Cebiroğlu Eryiğit, Giuseppe G. A. Celano, Savas Cetin, Fabricio Chalub, Jinho Choi, Silvie Cinková, Çağr Çöltekin, Miriam Connor, Elizabeth Davidson, Marie-Catherine de Marneffe, Valeria de Paiva, Arantza Diaz de Ilarraza, Peter Dirix, Kaja Dobrovoljc, Timothy Dozat, Kira Droганova, Puneet Dwivedi, Marhaba Eli, Ali Elkahky, Tomaz Erjavec, Richárd Farkas, Hector Fernandez Alcalde, Jennifer Foster, Cláudia Freitas, Katarína Gajdošová, Daniel Galbraith, Marcos Garcia, Moa Gärdenfors, Kim Gerdes, Filip Ginter, Iakes Goenaga, Koldo Gojenola, Memduh Gökırmak, Yoav Goldberg, Xavier Gómez Guinovart, Berta Gonzáles Saavedra, Matias Grioni, Normunds Grūzītis, Bruno Guillaume, Nizar Habash, Jan Hajič, Jan Hajič jr., Linh Hà Mỷ, Kim Harris, Dag Haug, Barbora Hladká, Jaroslava Hlaváčová, Florinel Hociung, Petter Hohle, Radu Ion, Elena Irimia, Tomáš Jelínek, Anders Johannsen, Fredrik Jørgensen, Hüner Kaşıkara, Hiroshi Kanayama, Jenna Kanerva, Tolga Kayadelen, Václava Kettnerová, Jesse Kirchner, Natalia Kotsyba, Simon Krek, Veronika Laippala, Lorenzo Lambertino, Tatiana Lando, John Lee, Phuong Lê H`ông, Alessandro Lenci, Saran Lertpradit, Herman Leung, Cheuk Ying Li, Josie Li, Keying Li, Nikola Ljubešić, Olga Loginova, Olga Lyashevskaya, Teresa Lynn, Vivien Mackentanz, Aibek Makazhanov, Michael Mandl, Christopher Manning, Cătălina

Mărănduc, David Mareček, Katrin Marheinecke, Héctor Martínez Alonso, André Martins, Jan Mašek, Yuji Matsumoto, Ryan McDonald, Gustavo Mendonça, Niko Miekka, Anna Missilä, Cătălin Mititelu, Yusuke Miyao, Simonetta Montemagni, Amir More, Laura Moreno Romero, Shin-suke Mori, Bohdan Moskalevskyi, Kadri Muischnek, Kaili Müürisep, Pinkey Nainwani, Anna Nedoluzhko, Gunta Nešpore-Bērzkalne, Luong Nguy`ên Thị, Huy`ên Nguy`ên Thị Minh, Vitaly Nikolaev, Hanna Nurmi, Stina Ojala, Petya Osenova, Robert Ostling, Lilja Ovreid, Elena Pascual, Marco Passarotti, Cenel-Augusto Perez, Guy Perrier, Slav Petrov, Jussi Piitulainen, Emily Pitler, Barbara Plank, Martin Popel, Lauma Pretkalniņa, Prokopis Prokopidis, Tiina Puolakainen, Sampo Pyysalo, Alexandre Rademaker, Loganathan Ramasamy, Taraka Rama, Vinit Ravishankar, Livy Real, Siva Reddy, Georg Rehm, Larissa Rinaldi, Laura Rituma, Mykhailo Romanenko, Rudolf Rosa, Davide Rovati, Benoît Sagot, Shadi Saleh, Tanja Samardžić, Manuela Sanguinetti, Baiba Saulīte, Sebastian Schuster, Djamé Seddah, Wolfgang Seeker, Mojgan Seraji, Mo Shen, Atsuko Shimada, Dmitry Sichinava, Natalia Silveira, Maria Simi, Radu Simionescu, Katalin Simkó, Mária Simková, Kiril Simov, Aaron Smith, Antonio Stella, Milan Straka, Jana Strnadová, Alane Suhr, Umut Sulubacak, Zsolt Szántó, Dima Taji, Takaaki Tanaka, Trond Trosterud, Anna Trukhina, Reut Tsarfaty, Francis Tyers, Sumire Uematsu, Zdeňka Urešová, Larraitz Uria, Hans Uszkoreit, Sowmya Vajjala, Daniel van Niek-erk, Gertjan van Noord, Viktor Varga, Eric Villemonte de la Clergerie, Veronika Vincze, Lars Wallin, Jonathan North Washington, Mats Wirén, Tak-sum Wong, Zhuoran Yu, Zdeněk Zabokrtský, Amir Zeldes, Daniel Zeman, and Hanzhi Zhu. *Universal Dependencies 2.1*, 2017.

Margaret King Odell. The profit in records management. Technical report, *Systems Magazine*, 1956.

Olutobi Owoputi, Brendan O'Connor, Chris Dyer, Kevin Gimpel, Nathan Schneider, and Noah A. Smith. Improved part-of-speech tagging for online conversational text with word clusters. In *Proceedings of the 2013 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies*, pages 380–390, Atlanta, Georgia, June 2013. Association for Computational Linguistics.

Kishore Papineni, Salim Roukos, Todd Ward, and Wei-Jing Zhu. Bleu: a

- method for automatic evaluation of machine translation. In *Proceedings of 40th Annual Meeting of the Association for Computational Linguistics*, pages 311–318, Philadelphia, Pennsylvania, USA, July 2002. Association for Computational Linguistics.
- Deana L. Pennell and Yang Liu. Normalization of informal text. *Computer Speech & Language*, 28(1):256–277, 2014.
- Slav Petrov and Dan Klein. Improved inference for unlexicalized parsing. In *Human Language Technologies 2007: The Conference of the North American Chapter of the Association for Computational Linguistics; Proceedings of the Main Conference*, pages 404–411, Rochester, New York, April 2007. Association for Computational Linguistics.
- Slav Petrov and Ryan McDonald. Overview of the 2012 shared task on parsing the web. In *Notes of the First Workshop on Syntactic Analysis of Non-Canonical Language (SANCL)*, volume 59, 2012.
- Slav Petrov, Leon Barrett, Romain Thibaux, and Dan Klein. Learning accurate, compact, and interpretable tree annotation. In *Proceedings of the 21st International Conference on Computational Linguistics and 44th Annual Meeting of the Association for Computational Linguistics*, pages 433–440, Sydney, Australia, July 2006. Association for Computational Linguistics.
- Slav Petrov, Dipanjan Das, and Ryan McDonald. A universal part-of-speech tagset. In *Proceedings of the Eight International Conference on Language Resources and Evaluation (LREC'12)*, Istanbul, Turkey, May 2012. European Language Resources Association (ELRA).
- Eva Pettersson, Beáta Megyesi, and Jörg Tiedemann. An smt approach to automatic annotation of historical text. In *Proceedings of the workshop on computational historical linguistics at NODALIDA 2013; May 22-24; 2013; Oslo; Norway. NEALT Proceedings Series 18*, pages 54–69, 2013.
- Lawrence Philips. The double metaphone search algorithm. In *C/C++ users journal*, volume 18, pages 38–43, 2000.

- Barbara Plank. What to do about non-standard (or non-canonical) language in NLP. *Proceedings of the 13th Conference on Natural Language Processing (KONVENS)*, 2016.
- Barbara Plank, Anders Søgaard, and Yoav Goldberg. Multilingual part-of-speech tagging with bidirectional long short-term memory models and auxiliary loss. In *Proceedings of the 54th Annual Meeting of the Association for Computational Linguistics (Volume 2: Short Papers)*, pages 412–418, Berlin, Germany, August 2016. Association for Computational Linguistics.
- Jordi Porta and José-Luis Sancho. Word normalization in Twitter using finite-state transducers. *Tweet-Norm@ SEPLN*, 1086:49–53, 2013.
- Vivek Kumar Rangarajan Sridhar. Unsupervised text normalization using distributed representations of words and phrases. In *Proceedings of the 1st Workshop on Vector Space Modeling for Natural Language Processing*, pages 8–16, Denver, Colorado, June 2015. Association for Computational Linguistics.
- Yafeng Ren, Jiayuan Deng, and Donghong Ji. Twitter normalization via 1-to-N recovering. In *International Conference on Web Information Systems Engineering*, pages 19–34. Springer, 2016.
- Martin Reynaert. All, and only, the errors: more complete and consistent spelling and ocr-error correction evaluation. In *Proceedings of the Sixth International Conference on Language Resources and Evaluation (LREC’08)*, Marrakech, Morocco, May 2008. European Language Resources Association (ELRA).
- CJ Rijsbergen. *Information Retrieval*, volume 2. University of Glasgow, 1979.
- Keisuke Sakaguchi, Matt Post, and Benjamin Van Durme. Error-repair dependency parsing for ungrammatical texts. In *Proceedings of the 55th Annual Meeting of the Association for Computational Linguistics (Volume 2: Short Papers)*, pages 189–195, Vancouver, Canada, July 2017. Association for Computational Linguistics.

- Sarah Schulz, Guy De Pauw, Orphée De Clercq, Bart Desmet, Véronique Hoste, Walter Daelemans, and Lieve Macken. Multimodular text normalization of Dutch user-generated content. *ACM Transactions on Intelligent Systems Technology*, 7(4):1–22, July 2016.
- Arnav Sharma, Sakshi Gupta, Raveesh Motlani, Piyush Bansal, Manish Shrivastava, Radhika Mamidi, and Dipti M. Sharma. Shallow parsing pipeline - Hindi-English code-mixed social media text. In *Proceedings of the 2016 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies*, pages 1340–1345, San Diego, California, June 2016. Association for Computational Linguistics.
- Natalia Silveira, Timothy Dozat, Marie-Catherine de Marneffe, Samuel Bowman, Miriam Connor, John Bauer, and Christopher D. Manning. A gold standard dependency corpus for English. In *Proceedings of the Ninth International Conference on Language Resources and Evaluation (LREC-2014)*, 2014.
- Erik Tjong Kim Sang and Antal van den Bosch. Dealing with big data: The case of Twitter. *Computational Linguistics in the Netherlands Journal*, 3: 121–134, December 2013.
- Kristina Toutanova, Dan Klein, Christopher D. Manning, and Yoram Singer. Feature-rich part-of-speech tagging with a cyclic dependency network. In *Proceedings of the 2003 Conference of the North American Chapter of the Association for Computational Linguistics on Human Language Technology*, volume 1, pages 173–180. Association for Computational Linguistics, 2003.
- Rob van der Goot. Normalizing social media texts by combining word embeddings and edit distances in a random forest regressor. In *Normalisation and Analysis of Social Media Texts (NormSoMe)*, 2016.
- Rob van der Goot and Gertjan van Noord. ROB: Using semantic meaning to recognize paraphrases. In *Proceedings of the 9th International Workshop on Semantic Evaluation (SemEval 2015)*, pages 40–44, Denver, Colorado, June 2015. Association for Computational Linguistics.

- Rob van der Goot and Gertjan van Noord. MoNoise: Modeling noise using a modular normalization system. *Computational Linguistics in the Netherlands Journal*, 7:129–144, December 2017a.
- Rob van der Goot and Gertjan van Noord. Parser adaptation for social media by integrating normalization. In *Proceedings of the 55th Annual Meeting of the Association for Computational Linguistics (Volume 2: Short Papers)*, pages 491–497, Vancouver, Canada, July 2017b. Association for Computational Linguistics.
- Rob van der Goot and Gertjan van Noord. Modeling input uncertainty in neural network dependency parsing. In *Proceedings of the 2018 Conference on Empirical Methods in Natural Language Processing*, pages 4984–4991, Brussels, Belgium, October 2018. Association for Computational Linguistics.
- Rob van der Goot, Barbara Plank, and Malvina Nissim. To normalize, or not to normalize: The impact of normalization on part-of-speech tagging. In *Proceedings of the 3rd Workshop on Noisy User-generated Text*, pages 31–39, Copenhagen, Denmark, September 2017. Association for Computational Linguistics.
- Rob van der Goot, Nikola Ljubešić, Ian Matroos, Malvina Nissim, and Barbara Plank. Bleaching text: Abstract features for cross-lingual gender prediction. In *Proceedings of the 56th Annual Meeting of the Association for Computational Linguistics (Volume 2: Short Papers)*, pages 383–389, Melbourne, Australia, 2018a. Association for Computational Linguistics.
- Rob van der Goot, Rik van Noord, and Gertjan van Noord. A taxonomy for in-depth evaluation of normalization for user generated content. In *Proceedings of the Eleventh International Conference on Language Resources and Evaluation (LREC 2018)*, Miyazaki, Japan, May 2018b. European Language Resources Association (ELRA).
- Marvin N. Wright and Andreas Ziegler. Ranger: A fast implementation of Random Forests for high dimensional data in C++ and R. *Journal of Statistical Software*, 77:1–17, 2017.
- Ke Xu, Yunqing Xia, and Chin-Hui Lee. Tweet normalization with syllables. In *Proceedings of the 53rd Annual Meeting of the Association for*

- Computational Linguistics and the 7th International Joint Conference on Natural Language Processing (Volume 1: Long Papers)*, pages 920–928, Beijing, China, July 2015. Association for Computational Linguistics.
- Yi Yang and Jacob Eisenstein. A log-linear model for unsupervised text normalization. In *Proceedings of the 2013 Conference on Empirical Methods in Natural Language Processing*, pages 61–72, Seattle, Washington, USA, October 2013. Association for Computational Linguistics.
- William J Youden. Index for rating diagnostic tests. *Cancer*, 3(1):32–35, 1950.
- Daniel H. Younger. Recognition and parsing of context-free languages in time  $n^3$ . *Information and control*, 10(2):189–208, 1967.
- Daniel Zeman, Martin Popel, Milan Straka, Jan Hajic, Joakim Nivre, Filip Ginter, Juhani Luotolahti, Sampo Pyysalo, Slav Petrov, Martin Potthast, Francis Tyers, Elena Badmaeva, Memduh Gokirmak, Anna Nedoluzhko, Silvie Cinkova, Jan Hajic jr., Jaroslava Hlavacova, Václava Kettnerová, Zdenka Uresova, Jenna Kanerva, Stina Ojala, Anna Missilä, Christopher D. Manning, Sebastian Schuster, Siva Reddy, Dima Taji, Nizar Habash, Herman Leung, Marie-Catherine de Marneffe, Manuela Sanguinetti, Maria Simi, Hiroshi Kanayama, Valeria dePaiva, Kira Droганova, Héctor Martínez Alonso, Çağr Çöltekin, Umut Sulubacak, Hans Uszkoreit, Vivien Macketanz, Aljoscha Burchardt, Kim Harris, Katrin Marheinecke, Georg Rehm, Tolga Kayadelen, Mohammed Attia, Ali Elkahky, Zhuoran Yu, Emily Pitler, Saran Lertpradit, Michael Mandl, Jesse Kirchner, Hector Fernandez Alcalde, Jana Strnadová, Esha Banerjee, Ruli Manurung, Antonio Stella, Atsuko Shimada, Sookyoung Kwak, Gustavo Mendonca, Tatiana Lando, Rattima Nitisaroj, and Josie Li. CoNLL 2017 shared task: Multilingual parsing from raw text to Universal Dependencies. In *Proceedings of the CoNLL 2017 Shared Task: Multilingual Parsing from Raw Text to Universal Dependencies*, pages 1–19, Vancouver, Canada, August 2017. Association for Computational Linguistics.
- Congle Zhang, Tyler Baldwin, Howard Ho, Benny Kimelfeld, and Yunyao Li. Adaptive parser-centric text normalization. In *Proceedings of the 51st Annual Meeting of the Association for Computational Linguistics*

---

(*Volume 1: Long Papers*), pages 1159–1168, Sofia, Bulgaria, August 2013.  
Association for Computational Linguistics.

# Nederlandse Samenvatting

Het syntactisch ontleden van natuurlijke taal is een belangrijke toepassing voor natuurlijke taalverwerking, omdat syntactisch ontleden de eerste stap is voor de interpretatie van taal. Voor teksten geschreven in ‘standaard’ taal werken bestaande automatische ontleders (eng: parsers) erg goed. Voor meer spontane taal, zoals gebruikt wordt op social media, werken deze ontleders veel slechter.

In dit proefschrift proberen we automatische ontleders te verbeteren voor teksten afkomstig van social media. Dit doen we door social media taal te ‘vertalen’ naar standaard taal. Dit wordt ook normalisatie genoemd. Een voorbeeld vertaling is:

kheb da gzien  
ik heb dat gezien

In dit voorbeeld worden alle woorden genormaliseerd. Het eerste woord wordt zelfs gesplitst in twee woorden. Het is duidelijk dat er een verschillende types van vervangingen nodig is, ‘kheb’ moet gesplit worden terwijl bij ‘da’ een medeklinker aan het einde toegevoegd moet worden, en bij ‘gzien’ een klinker ingevoegd wordt.

Omdat normalisatie uit verschillende soorten vervangingen bestaat, hebben we hiervoor een modulair systeem ontworpen; MoNoise. Dit systeem bestaat uit twee onderdelen, namelijk: 1) het genereren van normalisatie kandidaten 2) het ranken van deze kandidaten. Voor beide taken hebben we verschillende modules geëvalueerd. Hiervan zijn de belangrijkste: een ver-taalwoordenboek geleerd van de training data, word embeddings (word2vec), een spelling correctie systeem (Aspell) en n-gram waarschijnlijkheden. We testen MoNoise voor zeven datasets in zes verschillende talen, en voor bijna

alle datasets presteert MoNoise beter dan bestaande systemen.

Als eerste testen we het effect van deze normalisatie voor het toekennen van woordlabels (eng: POS tags) van tweets. Het gebruik van normalisatie leidt tot accuratere labels. Als de data waarop de labeler is getraind ook uit tweets bestaat, kan het bevorderlijk zijn om de training data ook te normaliseren.

Hierna testen we het effect van normalisatie op ontleders voor diepere syntax. In deze experimenten gebruiken we in plaats van een normalisatie per woord, een top-N lijst van normalisatiekandidaten. Dan ziet de normalisatie er als volgt uit:

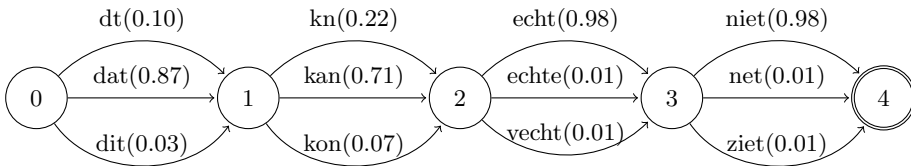


Figure E.4: Output (top-3) van het normalisatiesysteem voor de zin “dt kn echt niet meerr”

Als eerste testen we het effect van deze normalisatie voor een constituenten grammatica. Hiervoor gebruiken we de Berkeley parser (Petrov and Klein, 2007). Door eerst te normaliseren voordat we tweets ontleden, presteert de ontleder significant beter. Nadat we de Berkeley parser aanpassen zodat deze rekening kan houden met meerdere normalisatiekandidaten per positie, nemen de prestaties nog verder toe.

Vervolgens testen we het effect van normalisatie voor een dependentie ontleder. In dit geval maken we gebruik van een neurale netwerkparser, die al een aantal slimmigheidjes heeft om beter om te gaan met onbekende woorden. Onbekende woorden zijn ook de focus van ons normalisatie systeem, maar zelfs in deze setup is het bevorderlijk om normalisatie te gebruiken voor het ontleden. In neurale netwerkparsers worden woorden gerepresenteerd als vectoren met getallen. Hierdoor kunnen we meerdere normalisatie kandidaten per positie integreren door de vectoren van deze kandidaten te wegen aan de hand van de normalisatie waarschijnlijkheden en samen te voegen. Dit leidt wederom tot betere ontledingen.

# Groningen dissertations in linguistics (GRODIL)

1. Henriëtte de Swart (1991). *Adverbs of Quantification: A Generalized Quantifier Approach*.
2. Eric Hoekstra (1991). *Licensing Conditions on Phrase Structure*.
3. Dicky Gilbers (1992). *Phonological Networks. A Theory of Segment Representation*.
4. Helen de Hoop (1992). *Case Configuration and Noun Phrase Interpretation*.
5. Gosse Bouma (1993). *Nonmonotonicity and Categorical Unification Grammar*.
6. Peter I. Blok (1993). *The Interpretation of Focus*.
7. Roelien Bastiaanse (1993). *Studies in Aphasia*.
8. Bert Bos (1993). *Rapid User Interface Development with the Script Language Gist*.
9. Wim Kosmeijer (1993). *Barriers and Licensing*.
10. Jan-Wouter Zwart (1993). *Dutch Syntax: A Minimalist Approach*.
11. Mark Kas (1993). *Essays on Boolean Functions and Negative Polarity*.
12. Ton van der Wouden (1994). *Negative Contexts*.
13. Joop Houtman (1994). *Coordination and Constituency: A Study in Categorical Grammar*.
14. Petra Hendriks (1995). *Comparatives and Categorical Grammar*.
15. Maarten de Wind (1995). *Inversion in French*.
16. Jelly Julia de Jong (1996). *The Case of Bound Pronouns in Peripheral Romance*.
17. Sjoukje van der Wal (1996). *Negative Polarity Items and Negation: Tandem Acquisition*.
18. Anastasia Giannakidou (1997). *The Landscape of Polarity Items*.

19. Karen Lattewitz (1997). *Adjacency in Dutch and German*.
20. Edith Kaan (1997). *Processing Subject-Object Ambiguities in Dutch*.
21. Henny Klein (1997). *Adverbs of Degree in Dutch*.
22. Leonie Bosveld-de Smet (1998). *On Mass and Plural Quantification: The case of French 'des'/'du'-NPs*.
23. Rita Landeweerd (1998). *Discourse semantics of perspective and temporal structure*.
24. Mettina Veenstra (1998). *Formalizing the Minimalist Program*.
25. Roel Jonkers (1998). *Comprehension and Production of Verbs in aphasic Speakers*.
26. Erik F. Tjong Kim Sang (1998). *Machine Learning of Phonotactics*.
27. Paulien Rijkhoek (1998). *On Degree Phrases and Result Clauses*.
28. Jan de Jong (1999). *Specific Language Impairment in Dutch: Inflectional Morphology and Argument Structure*.
29. H. Wee (1999). *Definite Focus*.
30. Eun-Hee Lee (2000). *Dynamic and Stative Information in Temporal Reasoning: Korean tense and aspect in discourse*.
31. Ivilin P. Stoianov (2001). *Connectionist Lexical Processing*.
32. Klarien van der Linde (2001). *Sonority substitutions*.
33. Monique Lamers (2001). *Sentence processing: using syntactic, semantic, and thematic information*.
34. Shalom Zuckerman (2001). *The Acquisition of "Optional" Movement*.
35. Rob Koeling (2001). *Dialogue-Based Disambiguation: Using Dialogue Status to Improve Speech Understanding*.
36. Esther Ruigendijk (2002). *Case assignment in Agrammatism: a cross-linguistic study*.
37. Tony Mullen (2002). *An Investigation into Compositional Features and Feature Merging for Maximum Entropy-Based Parse Selection*.
38. Nanette Bienfait (2002). *Grammatica-onderwijs aan allochtone jongeren*.
39. Dirk-Bart den Ouden (2002). *Phonology in Aphasia: Syllables and segments in level-specific deficits*.
40. Rienk Withaar (2002). *The Role of the Phonological Loop in Sentence Comprehension*.
41. Kim Sauter (2002). *Transfer and Access to Universal Grammar in Adult Second Language Acquisition*.

42. Laura Sabourin (2003). *Grammatical Gender and Second Language Processing: An ERP Study*.
43. Hein van Schie (2003). *Visual Semantics*.
44. Lilia Schürcks-Grozeva (2003). *Binding and Bulgarian*.
45. Stasinou Konstantopoulou (2003). *Using ILP to Learn Local Linguistic Structures*.
46. Wilbert Heeringa (2004). *Measuring Dialect Pronunciation Differences using Levenshtein Distance*.
47. Wouter Jansen (2004). *Laryngeal Contrast and Phonetic Voicing: A Laboratory Phonology*.
48. Judith Rispens (2004). *Syntactic and phonological processing in developmental dyslexia*.
49. Danielle Bougäïré (2004). *L'approche communicative des campagnes de sensibilisation en santé publique au Burkina Faso: Les cas de la planification familiale, du sida et de l'excision*.
50. Tanja Gaustad (2004). *Linguistic Knowledge and Word Sense Disambiguation*.
51. Susanne Schoof (2004). *An HPSG Account of Nonfinite Verbal Complements in Latin*.
52. M. Begoña Villada Moirón (2005). *Data-driven identification of fixed expressions and their modifiability*.
53. Robbert Prins (2005). *Finite-State Pre-Processing for Natural Language Analysis*.
54. Leonoor van der Beek (2005). *Topics in Corpus-Based Dutch Syntax*.
55. Keiko Yoshioka (2005). *Linguistic and gestural introduction and tracking of referents in L1 and L2 discourse*.
56. Sible Andringa (2005). *Form-focused instruction and the development of second language proficiency*.
57. Joanneke Prenger (2005). *Taal telt! Een onderzoek naar de rol van taalvaardigheid en tekstbegrip in het realistisch wiskundeonderwijs*.
58. Neslihan Kansu-Yetkiner (2006). *Blood, Shame and Fear: Self-Presentation Strategies of Turkish Women's Talk about their Health and Sexuality*.
59. Mónika Z. Zemléni (2006). *Functional imaging of the hemispheric contribution to language processing*.
60. Maartje Schreuder (2006). *Prosodic Processes in Language and Music*.
61. Hidetoshi Shiraishi (2006). *Topics in Nivkh Phonology*.

62. Tamás Biró (2006). *Finding the Right Words: Implementing Optimality Theory with Simulated Annealing*.
63. Dieuwke de Goede (2006). *Verbs in Spoken Sentence Processing: Unraveling the Activation Pattern of the Matrix Verb*.
64. Eleonora Rossi (2007). *Clitic production in Italian agrammatism*.
65. Holger Hopp (2007). *Ultimate Attainment at the Interfaces in Second Language Acquisition: Grammar and Processing*.
66. Gerlof Bouma (2008). *Starting a Sentence in Dutch: A corpus study of subject- and object-fronting*.
67. Julia Klitsch (2008). *Open your eyes and listen carefully. Auditory and audiovisual speech perception and the McGurk effect in Dutch speakers with and without aphasia*.
68. Janneke ter Beek (2008). *Restructuring and Infinitival Complements in Dutch*.
69. Jori Mur (2008). *Off-line Answer Extraction for Question Answering*.
70. Lonneke van der Plas (2008). *Automatic Lexico-Semantic Acquisition for Question Answering*.
71. Arjen Versloot (2008). *Mechanisms of Language Change: Vowel reduction in 15th century West Frisian*.
72. Ismail Fahmi (2009). *Automatic term and Relation Extraction for Medical Question Answering System*.
73. Tuba Yarbay Duman (2009). *Turkish Agrammatic Aphasia: Word Order, Time Reference and Case*.
74. Maria Trofimova (2009). *Case Assignment by Prepositions in Russian Aphasia*.
75. Rasmus Steinkrauss (2009). *Frequency and Function in WH Question Acquisition. A Usage-Based Case Study of German L1 Acquisition*.
76. Marjolein Deunk (2009). *Discourse Practices in Preschool. Young Children's Participation in Everyday Classroom Activities*.
77. Sake Jager (2009). *Towards ICT-Integrated Language Learning: Developing an Implementation Framework in terms of Pedagogy, Technology and Environment*.
78. Francisco Dellatorre Borges (2010). *Parse Selection with Support Vector Machines*.
79. Geoffrey Andogah (2010). *Geographically Constrained Information Retrieval*.
80. Jacqueline van Kruiningen (2010). *Onderwijsontwerp als conversatie. Probleemoplossing in interprofessioneel overleg*.

81. Robert G. Shackleton (2010). *Quantitative Assessment of English-American Speech Relationships*.
82. Tim Van de Cruys (2010). *Mining for Meaning: The Extraction of Lexico-semantic Knowledge from Text*.
83. Therese Leinonen (2010). *An Acoustic Analysis of Vowel Pronunciation in Swedish Dialects*.
84. Erik-Jan Smits (2010). *Acquiring Quantification. How Children Use Semantics and Pragmatics to Constrain Meaning*.
85. Tal Caspi (2010). *A Dynamic Perspective on Second Language Development*.
86. Teodora Mehotcheva (2010). *After the fiesta is over. Foreign language attrition of Spanish in Dutch and German Erasmus Student*.
87. Xiaoyan Xu (2010). *English language attrition and retention in Chinese and Dutch university students*.
88. Jelena Prokić (2010). *Families and Resemblances*.
89. Radek Šimík (2011). *Modal existential wh-constructions*.
90. Katrien Colman (2011). *Behavioral and neuroimaging studies on language processing in Dutch speakers with Parkinson's disease*.
91. Siti Mina Tamah (2011). *A Study on Student Interaction in the Implementation of the Jigsaw Technique in Language Teaching*.
92. Aletta Kwant (2011). *Geraakt door prentenboeken. Effecten van het gebruik van prentenboeken op de sociaal-emotionele ontwikkeling van kleuters*.
93. Marlies Kluck (2011). *Sentence amalgamation*.
94. Anja Schüppert (2011). *Origin of asymmetry: Mutual intelligibility of spoken Danish and Swedish*.
95. Peter Nabende (2011). *Applying Dynamic Bayesian Networks in Transliteration Detection and Generation*.
96. Barbara Plank (2011). *Domain Adaptation for Parsing*.
97. Cagri Coltekin (2011). *Catching Words in a Stream of Speech: Computational simulations of segmenting transcribed child-directed speech*.
98. Dörte Hessler (2011). *Audiovisual Processing in Aphasic and Non-Brain-Damaged Listeners: The Whole is More than the Sum of its Parts*.
99. Herman Heringa (2012). *Appositional constructions*.
100. Diana Dimitrova (2012). *Neural Correlates of Prosody and Information Structure*.
101. Harwintha Anjarningsih (2012). *Time Reference in Standard Indonesian Grammatical Aphasia*.

102. Myrte Gosen (2012). *Tracing learning in interaction. An analysis of shared reading of picture books at kindergarten.*
103. Martijn Wieling (2012). *A Quantitative Approach to Social and Geographical Dialect Variation.*
104. Gisi Cannizzaro (2012). *Early word order and animacy.*
105. Kostadin Cholakov (2012). *Lexical Acquisition for Computational Grammars. A Unified Model.*
106. Karin Beijering (2012). *Expressions of epistemic modality in Mainland Scandinavian. A study into the lexicalization-grammaticalization-pragmaticalization interface.*
107. Veerle Baaijen (2012). *The development of understanding through writing.*
108. Jacolien van Rij (2012). *Pronoun processing: Computational, behavioral, and psychophysiological studies in children and adults.*
109. Ankelien Schippers (2012). *Variation and change in Germanic long-distance dependencies.*
110. Hanneke Loerts (2012). *Uncommon gender: Eyes and brains, native and second language learners, & grammatical gender.*
111. Marjoleine Sloos (2013). *Frequency and phonological grammar: An integrated approach. Evidence from German, Indonesian, and Japanese.*
112. Aysa Arylova (2013). *Possession in the Russian clause. Towards dynamicity in syntax.*
113. Daniël de Kok (2013). *Reversible Stochastic Attribute-Value Grammars.*
114. Gideon Kotzé (2013). *Complementary approaches to tree alignment: Combining statistical and rule-based methods.*
115. Fridah Katusemererwe (2013). *Computational Morphology and Bantu Language Learning: an Implementation for Runyakitara.*
116. Ryan C. Taylor (2013). *Tracking Referents: Markedness, World Knowledge and Pronoun Resolution.*
117. Hana Smiskova-Gustafsson (2013). *Chunks in L2 Development: A Usage-based Perspective.*
118. Milada Walková (2013). *The aspectual function of particles in phrasal verbs.*
119. Tom O. Abuom (2013). *Verb and Word Order Deficits in Swahili-English bilingual agrammatic speakers.*
120. Gülsen Yilmaz (2013). *Bilingual Language Development among the First Generation Turkish Immigrants in the Netherlands.*
121. Trevor Benjamin (2013). *Signaling Trouble: On the linguistic design of other-initiation of repair in English conversation.*

122. Nguyen Hong Thi Phuong (2013). *A Dynamic Usage-based Approach to Second Language Teaching*.
123. Harm Brouwer (2014). *The Electrophysiology of Language Comprehension: A Neurocomputational Model*.
124. Kendall Decker (2014). *Orthography Development for Creole Languages*.
125. Laura S. Bos (2015). *The Brain, Verbs, and the Past: Neurolinguistic Studies on Time Reference*.
126. Rimke Groenewold (2015). *Direct and indirect speech in aphasia: Studies of spoken discourse production and comprehension*.
127. Huiping Chan (2015). *A Dynamic Approach to the Development of Lexicon and Syntax in a Second Language*.
128. James Griffiths (2015). *On appositives*.
129. Pavel Rudnev (2015). *Dependency and discourse-configurationality: A study of Avar*.
130. Kirsten Kolstrup (2015). *Opportunities to speak. A qualitative study of a second language in use*.
131. Güliz Güneş (2015). *Deriving Prosodic structures*.
132. Cornelia Lahmann (2015). *Beyond barriers. Complexity, accuracy, and fluency in long-term L2 speakers'speech*.
133. Sri Wachyunni (2015). *Scaffolding and Cooperative Learning: Effects on Reading Comprehension and Vocabulary Knowledge in English as a Foreign Language*.
134. Albert Walsweer (2015). *Ruimte voor leren. Een etnogafisch onderzoek naar het verloop van een interventie gericht op versterking van het taalgebruik in een knowledge building environment op kleine Friese basisscholen*.
135. Aleyda Lizeth Linares Calix (2015). *Raising Metacognitive Genre Awareness in L2 Academic Readers and Writers*.
136. Fathima Mufeeda Irshad (2015). *Second Language Development through the Lens of a Dynamic Usage-Based Approach*.
137. Oscar Strik (2015). *Modelling analogical change. A history of Swedish and Frisian verb inflection*.
138. He Sun (2015). *Predictors and stages of very young child EFL learners'English development in China*.
139. Marieke Haan (2015). *Mode Matters. Effects of survey modes on participation and answering behavior*.
140. Nienke Houtzager (2015). *Bilingual advantages in middle-aged and elderly populations*.

141. Noortje Joost Venhuizen (2015). *Projection in Discourse: A data-driven formal semantic analysis*.
142. Valerio Basile (2015). *From Logic to Language: Natural Language Generation from Logical Forms*.
143. Jinxing Yue (2016). *Tone-word Recognition in Mandarin Chinese: Influences of lexical-level representations*.
144. Seçkin Arslan (2016). *Neurolinguistic and Psycholinguistic Investigations on Evidentiality in Turkish*.
145. Rui Qin (2016). *Neurophysiological Studies of Reading Fluency. Towards Visual and Auditory Markers of Developmental Dyslexia*.
146. Kashmiri Stec (2016). *Visible Quotation: The Multimodal Expression of Viewpoint*.
147. Yinxing Jin (2016). *Foreign language classroom anxiety: A study of Chinese university students of Japanese and English over time*.
148. Joost Hurkmans (2016). *The Treatment of Apraxia of Speech. Speech and Music Therapy, an Innovative Joint Effort*.
149. Franziska Köder (2016). *Between direct and indirect speech: The acquisition of pronouns in reported speech*.
150. Femke Swarte (2016). *Predicting the mutual intelligibility of Germanic languages from linguistic and extra-linguistic factors*.
151. Sanne Kuijper (2016). *Communication abilities of children with ASD and ADHD. Production, comprehension, and cognitive mechanisms*.
152. Jelena Golubović (2016). *Mutual intelligibility in the Slavic language area*.
153. Nynke van der Schaaf (2016). *Kijk eens wat ik kan! Sociale praktijken in de interactie tussen kinderen van 4 tot 8 jaar in de buitenschoolse opvang*.
154. Simon Šuster (2016). *Empirical studies on word representations*.
155. Kilian Evang (2016). *Cross-lingual Semantic Parsing with Categorical Grammars*.
156. Miren Arantzeta Pérez (2017). *Sentence comprehension in monolingual and bilingual aphasia: Evidence from behavioral and eye-tracking methods*.
157. Sana-e-Zehra Haidry (2017). *Assessment of Dyslexia in the Urdu Language*.
158. Srdan Popov (2017). *Auditory and Visual ERP Correlates of Gender Agreement Processing in Dutch and Italian*.
159. Molood Sadat Safavi (2017). *The Competition of Memory and Expectation in Resolving Long-Distance Dependencies: Psycholinguistic Evidence from Persian Complex Predicates*.

160. Christopher Bergmann (2017). *Facets of native-likeness: First-language attrition among German emigrants to Anglophone North America.*
161. Stefanie Keulen (2017). *Foreign Accent Syndrome: A Neurolinguistic Analysis.*
162. Franz Manni (2017). *Linguistic Probes into Human History.*
163. Margreet Vogelzang (2017). *Reference and cognition: Experimental and computational cognitive modeling studies on reference processing in Dutch and Italian.*
164. Johannes Bjerva (2017). *One Model to Rule them all. Multitask and Multilingual Modelling for Lexical Analysis: Multitask and Multilingual Modelling for Lexical Analysis.*
165. Dieke Oele (2018). *Automated translation with interlingual word representations.*
166. Lucas M. Seuren (2018). *The Interactional Accomplishment of Action.*
167. Elisabeth Borleffs (2018). *Cracking the code - Towards understanding, diagnosing and remediating dyslexia in Standard Indonesian.*
168. Mirjam Günther-van der Meij (2018). *The impact of degree of bilingualism on L3 development English language development in early and later bilinguals in the Frisian context.*
169. Ruth Koops van t Jagt (2018). *Show, don't just tell: Photo stories to support people with limited health literacy.*
170. Bernat Bardagil-Mas (2018). *Case and agreement in Panará.*
171. Jessica Overweg (2018). *Taking an alternative perspective on language in autism.*
172. Lennie Donné (2018). *Convincing through conversation: Unraveling the role of interpersonal health communication in health campaign effectiveness.*
173. Toivo Glatz (2018). *Serious games as a level playing field for early literacy: A behavioural and neurophysiological evaluation.*
174. Ellie van Setten (2019). *Neurolinguistic Profiles of Advanced Readers with Developmental Dyslexia.*
175. Anna Pot (2019). *Aging in multilingual Netherlands: Effects on cognition, wellbeing and health.*
176. Audrey Rouse-Malpat (2019). *Effectiveness of explicit vs. implicit L2 instruction: a longitudinal classroom study on oral and written skills.*
177. Rob van der Goot (2019). *Normalization and Parsing Algorithms for Uncertain Input.*

GRODIL  
Center for Language and Cognition Groningen (CLCG)  
P.O. Box 716  
9700 AS Groningen  
The Netherlands