DISSERTATION

Dealing with Spelling Variation in Non-Standard Texts

Fabian Barteld

An der Universität Hamburg eingereichte Dissertation
zur Erlangung des akademischen Grades
Doctor rerum naturalium (Dr.rer.nat.)

Fakultät für Mathematik, Informatik und Naturwissenschaften
Fachbereich Informatik

2021

Prüfungskommission
Prof. Dr. Chris Biemann (Betreuer, Gutachter)
Prof. Dr. Stefanie Dipper (Gutachterin)
Prof. Dr. Simone Frintrop (stellv. Vorsitz)
Prof. Dr. Ricardo Usbeck (Vorsitz)
Prof. Dr. Heike Zinsmeister (Betreuerin, Gutachterin)

Tag der Disputation
07.03.2022

Abstract

In this thesis, we introduce and analyze ways to deal with spelling variation for the automatic processing of texts. Spelling variation is the phenomenon that words are written differently throughout a text, which appears frequently in so-called non-standard texts. We concentrate on dialectal historical German texts from between 1050 and 1650 as an example for non-standard texts.

Spelling variation complicates the automatic processing of texts. Such processing includes the annotation of texts with parts of speech (POS) and lemmas but also simply searching in these texts. Since non-standard texts have received increasing attention in computational linguistics, there has also been a rising interest in automatically dealing with spelling variation in recent years. What sets the approaches presented in this thesis apart from the approaches in most of the literature is that spelling variation is dealt with without resorting to a given standard. This is helpful for non-standard data without closely related standard data.

We look at two approaches to spelling variation: *simplification* and *spelling variant detection*. We evaluate our approaches in two evaluation settings that we have designed to approximate the utility of approaches for searching in non-standard texts and for applying Natural Language Processing (NLP) tools to the texts.

Simplification aims to map different spelling variants to the same word form such that the overall variation in the texts is reduced. Differing from normalization, the results of the mapping do not need to be existing word forms from a standardized language. For simplification, we propose a rule-based approach in which the rules are derived from pairs of equivalent characters or a character and a character bigram. These pairs can either be manually created or learned from known spelling variants.

The goal of spelling variant detection is to find spelling variants for a given word form. For this, we propose a pipeline in which candidates for spelling variants are first generated using string similarity. These candidates are then filtered to remove falsely generated candidates. For this, we train a Machine Learning (ML) algorithm to distinguish spelling variants from pairs of word forms that are not spelling variants. The features used for this are the surface differences between the two word forms as well as the contexts in which they appear.

Regarding NLP, we look into the tasks of POS tagging and lemmatization in more detail. For both tasks, we use statistical tools that have been developed with standard data in mind and adapt them for non-standard data. We show that by slightly adapting the ML approach but also by using the automatic spelling variant detection presented in this thesis, the performance of such tools on non-standard texts can be improved without the need for additional data.

With the presented approaches, we show examples of how dealing with spelling variation is possible without the usage of a defined standard. These techniques

allow to improve the automatic processing for historical but also other kinds of non-standard texts and are helpful when either no closely related standard or no training data for normalization is available.

Zusammenfassung

In dieser Arbeit präsentieren und analysieren wir Möglichkeiten zum Umgang mit Schreibvariation für die automatische Verarbeitung von Texten. Schreibvariation ist das Phänomen, dass Wörter in einem Text unterschiedlich geschrieben werden. Dies tritt häufig in sogenannten Nicht-Standard-Texten auf. Wir konzentrieren uns auf dialektale historische deutsche Texte aus der Zeit zwischen 1050 und 1650 als Beispiel für solche Texte.

Schreibvariation erschwert die automatische Verarbeitung von Texten. Verarbeitung umfasst sowohl die Annotation von Texten mit Wortarten — parts of speech (POS) — und Lemmata, aber auch das einfache Durchsuchen solcher Texte. Da Nicht-Standard-Texte zunehmende Aufmerksamkeit in der Computerlinguistik gefunden haben, hat auch das Interesse an der automatischen Verarbeitung von Schreibvariation in den letzten Jahren zugenommen. Die in dieser Arbeit vorgestellten Ansätze unterscheiden sich von den üblichen Ansätzen dadurch, dass auf die Variation der Schreibweise eingegangen wird ohne Bezug zu einem vorgegebenen Standard. Dies ist hilfreich für Nicht-Standard-Daten ohne engen Bezug zu Standarddaten.

Wir betrachten zwei Ansätze zur Verarbeitung von Schreibvariation: *Vereinfachung* (simplification) und *Erkennung von Schreibvarianten* (spelling variant detection). Wir evaluieren unsere Ansätze auf zwei unterschiedlche Arten, um einerseits den Nutzen der Ansätze für die Suche in Nicht-Standard-Texten und andererseits für die Anwendung von Werkzeugen zur automatischen Sprachverarbeitung — Natural Language Processing (NLP) — abzuschätzen.

Bei der Vereinfachung wird darauf abgezielt, verschiedene Schreibvarianten derselben Wortform zuzuordnen, wodurch die Variation in den Texten reduziert wird. Abweichend von Normalisierung muss bei der Vereinfachung das Ergebnis des Mappings keine existierende Wortform aus einer Standardsprache sein. Für die Vereinfachung schlagen wir einen regelbasierten Ansatz vor, bei dem die Regeln aus Paaren äquivalenter Zeichen oder einem Zeichen und einem Zeichen-Bigramm abgeleitet werden. Solche Paare können entweder manuell erstellt oder aus bekannten Schreibvarianten gelernt werden.

Das Ziel der Erkennung von Schreibvarianten ist es, für eine gegebene Wortform Schreibvarianten zu finden. Dazu schlagen wir eine Pipeline vor, in der Kandidaten für Schreibvarianten zunächst unter Verwendung der Ähnlichkeit von Zeichenketten erzeugt werden. Diese Kandidaten werden dann gefiltert um falsch generierte Kandidaten zu entfernen. Dafür wenden wir Methoden des maschinellen Lernens — Machine Learning (ML) — an. Wir trainieren einen Algorithmus, um Schreibvarianten von anderen Paaren zu unterscheiden. Die hierfür verwendeten Features sind einerseits die Oberflächenunterschiede zwischen den beiden Wortformen sowie die Kontexte, in denen sie auftreten.

Bezüglich NLP betrachten wir POS-Tagging und Lemmatisierung ausführlicher. Für beide Aufgaben verwenden wir statistische Tools, die unter Berücksichtigung von Standarddaten entwickelt wurden und die wir für Nicht-Standard-Daten angepasst haben. Wir zeigen, dass durch eine leichte Anpassung des Machine-Learning-Ansatzes, aber auch durch die Verwendung von der in dieser Arbeit vorgestellten automatischen Erkennung von Schreibvarianten, die Qualität der mit solchen Tools erstellten Annotationen bei Nicht-Standard-Texten verbessert werden kann, ohne dass zusätzliche Daten oder Annotationen nötig sind.

Mit den vorgestellten Ansätzen zeigen wir Beispiele für den Umgang mit Schreibvariation die ohne die Verwendung eines definierten Standards auskommen. Diese Techniken ermöglichen eine Verbesserung der automatischen Verarbeitung von historischen, aber auch andere Arten von Nicht-Standard-Texten, für die entweder kein eng verwandter Standard existiert oder keine Trainingsdaten für die Normalisierung vorhanden sind.

Contents

List of Acronyms

The abbreviations for languages follow ISO 639-3.

CNN	Convolutional Neural Network
CRF	Conditional Random Field
DH	Digital Humanities
ET	Edit Tree
LC	Lexical Correspondence
GMH	Middle High German
GML	Middle Low German
HiTS	Historisches Tagset
HiNTS	Historisches Niederdeutsch Tagset
HMM	Hidden Markov Model
IR	Information Retrieval
ML	Machine Learning
MLE	Maximum Likelihood Estimation
NLP	Natural Language Processing
OOV	out-of-vocabulary
POS	part of speech
PPMI	positive Pointwise Mutual Information
PU learning	learning from positive and unlabeled examples
ReLU	Rectified Linear Unit
ReN	Reference Corpus Middle Low German/Low Rhenish (1200–1650)

ReM	Reference Corpus Middle High German (1050–1350)
RBF	Radial Basis Function
SGNS	Skip-Gram with Negative Sampling
SVD	Singular Value Decomposition
SVM	Support Vector Machine

List of Figures

List of Tables

Chapter 1.

Introduction

Contrary to standardized written language that is found, e. g., in modern newspapers, non-standardized written language, like in historical texts or in computer-mediated communication, is full of spelling variation. Spelling variation is the phenomenon that the same word appears written differently. This thesis is about dealing with spelling variation for the automatic processing of such non-standard texts. More specifically, we conduct experiments with texts from Middle Low German (GML), a group of German dialects from the 13$^{\text{th}}$ to the 17$^{\text{th}}$ century, and Middle High German (GMH), a group of German dialects from the 11$^{\text{th}}$ to the 14$^{\text{th}}$ century, on how spelling variants can be automatically identified and removed or used when further processing the texts. The following examples show spelling variation in GML:[1]

(1) a. **Unde** dachte nycht vp vergenckelyck gud tho krygene
and thought not about ephemeral good to acquire
and did not think about acquiring ephemeral goods
vnde ouede syck in wiltwarck to iagen
and practiced himself in wild game to hunt
and practiced hunting game
(Griseldis)

b. ¶ Wo Johan wedder na **Parysz** reeth to Ambrosio.
how Johann again to Paris rode to Ambosio
How Johann rides again to Paris to Ambrosio
de dar sinre vorbeydende was.
who there he.GEN waiting was
who was waiting there for him.

1. The GML text examples in this thesis are from the Reference Corpus Middle Low German/Low Rhenish (1200–1650) (ReN 1.0). The texts containing the examples are referenced with the sigles given the texts in this corpus (cf. abbr_ddd in the documentation of the metadata, `Metadatendokumentation_2019-08-14.pdf`, which is part of the release of the ReN 1.0, `http://hdl.handle.net/11022/0000-0007-D829-8`). For labeling morphological categories in the examples, we use abbreviations following the Leipzig glossing rules (`https://www.eva.mpg.de/lingua/resources/glossing-rules.php`, last visited October 4, 2021).

DAr na sumede he nicht lange.
After that hesitated he not long
After that he did not hesitate long
sunder in groter vrolicheit reeth he na **Parys**
but with great happiness rode he to Paris
but he happily rode to Paris
(Veer Koeplude)

Spelling variation can be due to dialectal or temporal differences between texts. However, spelling variants even appear close to each other in the same text: Example (1a) shows adjacent lines of the text *Griseldis* in which the word *unde* ('and') appears as *Unde* and *vnde*. And even proper nouns show variation. In Example (1b) the city name *Parys* ('Paris') is written as *Parysz* in a caption and as *Parys* directly in the beginning of the following paragraph. These kinds of spelling variants lead to problems when working with digitized versions of these texts. For example, when extracting all mentions of the city Paris from the text *Veer Koeplude*, one needs to know that the city name appears with at least the two spellings shown above.

In computational linguistics, spelling variation has been examined in historical texts but also in other types of texts like computer-mediated communication as it appears in forums or chats and learner texts. These types of texts can be grouped under the term *non-standard texts* to distinguish them from standard texts used in computational linguistics, i. e., mainly newspaper texts (Plank 2016). When dealing with non-standard texts, common methods from computational linguistics often show shortcomings due to the idiosyncrasies of these texts such as spelling variation. E. g., simply extracting word frequencies becomes problematic (Baron, Rayson, and Archer 2009). Hence, handling spelling variation is important. A common way to do this is by *normalization*. When normalizing a non-standard text, spelling variants are mapped to the variant that appears in closely related standard texts. Thereby spelling variation is removed and the texts are changed in order to become more similar to standard texts.

In this thesis, however, we look at ways to automatically deal with spelling variation without resorting to a related standard language. For this, we exploit the fact that the variation in the spelling is not random but follows certain patterns. The alternation of *u* and *v*, for instance, as shown in Example (1a), is very common in GML texts. So one way to deal with spelling variation is to train a Machine Learning (ML) algorithm on examples of spelling variants from GML in order to distinguish pairs like *unde* and *vnde* from pairs like *unde* and *hunde* ('dogs') using such patterns of variation.

We are mainly interested in using learned patterns of variation to improve the performance of Natural Language Processing (NLP) tools. NLP is a field of computer

science that explores techniques for automatically processing natural language. According to Manning and Schütze (1999) the long term goal of NLP "is to parse and understand language" (p. 341). To achieve this goal in the long run "much research in NLP has focused on intermediate tasks that make sense of some of the structure inherent in language without requiring complete understanding" (Manning and Schütze 1999, p. 341). Two common NLP tasks, which we look at in this thesis, are part-of-speech (POS) tagging and lemmatization.

POS tagging is the task to find the corresponding sequence of POS tags for a sequence of words, often a sentence. Lemmatization is the task to map inflected forms of a word to a common form that one would use as base form in a dictionary, i.e., the lemma. The commonality of both tasks is that both aim to group word forms that are observed in natural language under more abstract categories, the POS and the lemma. This way systematics underlying a sequence of words are revealed. Therefore, POS tagging and lemmatization help in the overall goal of parsing and understanding language. Example (2) shows an English sentence with POS information and lemmas. The sentence contains two determiners (DT), a verb (VBZ) and a noun (NN). The three POS categories in the example are encoded with tags using the Penn tagset as used in the Penn treebank for English (Marcus, Santorini, and Marcinkiewicz 1993).

(2)

token	This	is	an	example
POS	DT	VBZ	DT	NN
lemma	this	be	an	example

In Section 1.1, we further motivate the topic of this thesis. We also motivate the approaches to dealing with spelling variation that are used in this thesis in comparison with other approaches, mainly contrasting dealing with spelling variation without reference to a standard language in contrast to approaches like normalization that use a standard language as reference. Section 1.2 gives an overview of the structure of the thesis.

1.1. Motivation

The rise of Digital Humanities (DH) has increased interest in processing non-standard data to computational linguistics.[2] Non-standard data is as diverse as user-generated content from the internet, learner data or historical texts (Dipper, Lüdeling, and Reznicek 2013). While each of these types of non-standard data poses different challenges to NLP, non-standard data also has commonalities—especially

2. See also the workshop series on Language Technology for Cultural Heritage, Social Sciences, and Humanities (LaTeCH) by ACL SIGHUM (`https://sighum.wordpress.com`, last visited October 4, 2021).

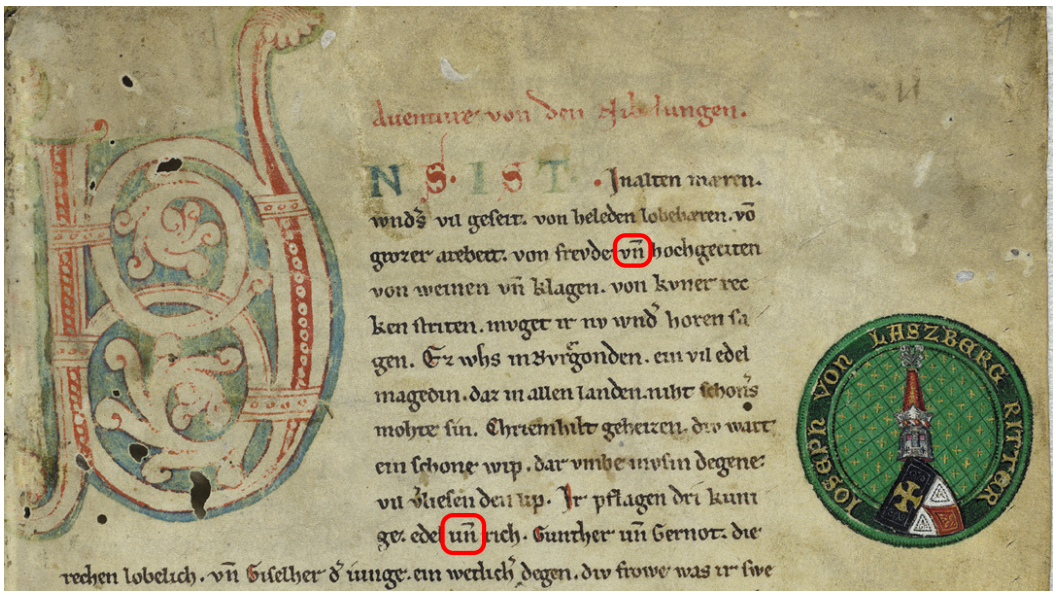


Figure 1.1.: First page of the 'Nibelungenlied' (Manuscript C) – Cod. Donaueschingen 63, Bl. 1r, Karlsruhe: Badische Landesbibliothek, 2003 (URN urn:nbn:de:bsz:31-28918), License: CC BY 4.0

when compared to newswire that "has and actually still *does* dominate [computational linguistics]" (Plank 2016). This thesis focuses on one particular property of non-standard texts, namely spelling variation—the phenomenon in which one and the same word is spelled in different ways. This variation is—for different reasons—characteristic of all the above-mentioned types of non-standard data. Here, we will focus on spelling variation in historical texts, specifically German historical texts.

Elmentaler (2018, p. 14) illustrates spelling variation in historical texts with the beginning of the famous *Nibelungenlied* (*The song of the Nibelungs*). In manuscript C of this German text from the 13$^{\text{th}}$ century, the word *unde*, which is the GMH equivalent of '*and*', appears written as *un̄* and *vn̄*.[3] These two variants are marked with red in Figure 1.1 showing the beginning of this manuscript.

For readers having internalized a modern standardized writing system, this is unfamiliar. Such readers—and writers as well—are used to existing rules for spelling and therefore expect only little variability or even invariability in one text. When they know the rules of a specific writing system, readers can more easily decode a text since there is no need to interpret new spelling variants that they encounter (Elmentaler 2018, p. 15). However, as Elmentaler (2018, pp. 15) points out, fixed rules limit the freedom of writers, e. g., when different pronunciations cannot be

3. The overline signifies an abbreviation.

displayed adequately. Deviations from a spelling norm can also be used without reference to pronunciation, e. g., for emphasis as it appears in communications with smartphones, in advertising or in literature, for instance by capitalizing a whole word.

However, invariability in spelling is not only useful when simply reading texts. It is also useful when using texts as a corpus for linguistic analyses or as source for other disciplines. When working with an unannotated corpus, variation in the spelling decreases the utility of the data, e. g., by reducing the recall for queries and distorting keyword frequencies (Baron, Rayson, and Archer 2009), or by leading to a decreased quality of learned representations for words and sentences (Kumar, Makhija, and Gupta 2020). Furthermore, the variation makes it harder to annotate this data automatically (cf. Bollmann 2018, p. 4 for theoretical considerations and Giesbrecht and Evert 2009 for an example regarding POS tagging). One reason for this is that a word form that does not appear in texts used to train an NLP tool is less likely to be annotated correctly by the tool than a known spelling.[4] In non-standard texts with spelling variation, the number of word forms that do not appear in the training data—out-of-vocabulary (OOV) words in NLP terminology— is higher when compared to the same amount of standard data. In addition, during training time, instances of the same word appear as different types, thereby distributing the information about one word over these types.

Dipper (2011) compares type/token ratios (TTRs) for equally sized parts of modern German and GMH corpora and uses the TTRs as indicators for the quality of automatic POS tagging. The TTR of the normalized GMH texts is lower than the TTR of the modern German texts. This indicates that tagging normalized GMH is easier than the modern texts. Without normalization, however, the TTR of the GMH texts is higher. Therefore, spelling variation as a property of non-standard texts makes processing this kind of data inherently harder than processing standard data. Consequently, researchers in computational linguistics are searching for ways to deal with spelling variation.

Plank (2016) identifies three general strategies to deal with non-standard data for tasks in computational linguistics:

1. Annotate more data, i. e., creating more training data for the non-standard domain to train tools on it.

2. Make training and test data more similar by either normalization, i. e., mapping non-standard data to related standard data, or (less commonly) by introducing properties of the non-standard data like spelling variation to the standard data used to train tools on.

4. See Chapter 4 for more details on ML basics.

3. Domain adaptation, i. e., adapting models trained on standard data to non-standard domains.

Research in computational linguistics has focused mainly on the second and the third way. This can be explained by the fact that these two strategies allow using resources existing for standard data like annotated corpora and existing NLP tools. However, they both rely on the existence of a standard variety that is closely related to the non-standard data. Thus approaches following these two strategies are often congruent with the view of a reader or writer of a standard language where spelling variation can be viewed as deviation from that standard. Such a view can be found, e. g., in the title of a paper on normalizing non-standard computer mediated communication by Eisenstein (2013): 'What to do about bad language on the internet'. This view can also be seen in the analysis of the decreased accuracy of POS taggers in non-newspaper texts by Giesbrecht and Evert (2009). The authors describe some of the datasets they looked at as containing "[w]eb-specific text genres that have not been carefully edited like the newspaper articles in the TIGER treebank. As a result, they contain many typographical and grammatical mistakes" (Giesbrecht and Evert 2009, p. 32).

The deviation between standard and non-standard data can be looked at from either the standard or the non-standard data. *Normalization*, a common approach using the second strategy, is looking from non-standard to standard data (Eisenstein 2013 for user-generated content; Jurish 2010a for historical texts): While there are different ways of approaching normalization (e. g., as spelling correction or as machine translation, cf. Kobus, Yvon, and Damnati 2008), the necessary common point of all approaches is the reference to a target language, i. e., a closely related standard that the non-standard data is normalized to. Thus, for non-standard data for which no closely-related standard exists normalization is problematic:

> In cases where the language under investigation differs considerably from its standard form, text normalization is a problem that is difficult to model. Errors made in the normalization propagate down to subsequent processing steps which influence the results of the final analysis. Moreover, text normalization requires a considerable amount of training data which DH projects are often lacking. Commonly, deviations from the standard form are exactly the focus of the investigation and thus have to be handled with care.
>
> (Schulz 2018, p. 89)

This is also true for *domain adaptation*, the third strategy, which is approaching non-standard and standard data the other way around by changing tools developed for and with standard data so that they work with closely related non-standard

data: This task also becomes more difficult when the non-standard data differs considerably from the standard data.

The first approach—creating more training data—is a viable way to go when working with non-standard data without closely related non-standard data. However, it is also costly and time-consuming. Furthermore, as mentioned above, spelling variation makes the automatic annotation of non-standard texts harder. Hence, to get results of the same quality, more annotated data is needed for non-standard data in comparison to standard data. Therefore, even when using the strategy to annotate more of the non-standard data under consideration, it is important to deal with spelling variation. Another factor is that while creating more training data might be viable for a specific domain of a specific language, "[w]e will never be able to create annotated data that spans all possible combinations" (Plank 2016, p. 14) of domains and languages. This is why it is crucial to deal with spelling variation independently of a standard language. In this thesis, we approach this by identifying spelling variants and/or removing variation. We introduce and analyze respective approaches in Chapter 6 and Chapter 7.

Automatically detected spelling variants can either be directly used for research (see e. g., Andrews 2016; Dipper and Waldenberger 2017; Spadini 2019 for examples of using variants in DH and linguistic research) or for improving the automatic analysis of texts. In this thesis, we look at the utility of detecting spelling variants for further processing of the texts using standard NLP tools in Chapter 8.

For applying NLP tools to non-standard texts, we consider two basic strategies to deal with spelling variation without referring to a standard language, adding to the strategies described by Plank (2016): The first approach is a task-specific adaptation of tools to the needs of non-standard domains, e. g., by including features that help the respective tools to learn variant spellings (e. g., Gimpel et al. 2011; Koleva et al. 2017). Differing from domain adaptation where the goal is to train a model on standard data and adapt the learned model to non-standard data, we train a model directly on the non-standard data.

The second approach is to utilize detected spelling variants in order to remove variation from the data and thus improve the performance of the respective NLP tool. For such processing, detecting spelling variants without referring to a standard language can be useful in two situations: a) when there is missing or sparse training data for normalization, and b) when the non-standard data lacks a closely related standard. Regarding training data, Odebrecht et al. (2017) state for a corpus of Early New High German (1350-1650) texts that they encountered problems with the quality of automatic normalization:

> Statistically learned rules for normalization have not worked well so far either, as the corpus is too small for statistical training as applied e. g., by Jurish (2010b), Bollmann et al. (2011, 2012), or Archer et al. (2015),

> for an overview see Piotrowski (2012). A key problem for a diachronic corpus is that orthography is changing across periods, and each text would require its own normalization rules.
>
> (Odebrecht et al. 2017, p. 715)

Recently, the problem of sparse training data for normalization has been addressed by leveraging resources for the target language to obtain better normalizations (Makarov and Clematide 2020). While approaches like this might improve the quality of the normalization, the problem gets worse for variants of German that differ more from modern Standard German like the older variant GMH or for historical variants that do not come from the High German dialects but from Low German like GML. This is also true in other contexts, e. g., when working with low-resourced endangered languages (Littell, Chelliah, and Levow 2016).

The lack of training data points to another issue relevant in the context of non-standard data: the lack of resources in general. While there are already a lot of resources and tools for standardized modern German, this is not the case for historical German especially when looking at early stages of German like Old High German (750–1050) where even existing text data is rather sparse and highly heterogeneous (cf. Flick 2019). Therefore, we look at different approaches to detecting spelling variants in the context of low-resource scenarios and use them for NLP tasks.

The main dataset that is used for this thesis is a corpus of GML texts. Example (3) shows two sentences from the same text from this corpus. These two sentences contain two spelling variants where i varies with j (*in* and *jn*, 'in') and with y (*steit* and *steyt*, 'is indicated').[5]

(3) a. *in* dem seuenden *steit*
in the seventh is_indicated

b. *JN* dem elenboghen *steyt*
in the elbow is_indicated
(Brem. Ssp.)

Contrary to High German, the Low German dialects have not developed a standardized variant. Hence, the texts differ not only because of language change over time from modern Standard German but also because of the dialectal differences between High and Low German. Therefore, GML is an example for non-standard data for which no closely related standard language exists. Furthermore, Low German bears similarities to German, Dutch and English (Sanders 1982), which makes the choice of a target language less obvious and normalization to any of these languages harder. We discuss GML in more detail in Section 2.1.

In the next section, we give an overview of the parts of this thesis.

5. In all experiments in this thesis, we ignore differences in the capitalization. All types are lowercased before applying and evaluating the presented methods.

1.2. Outline and Main Contributions

As we have argued in the previous section, dealing with spelling variation without reference to a standard language is important. In this thesis, we introduce and evaluate approaches to automatically dealing with spelling variation. For this, we use data from GML and GMH that we introduce in Chapter 2.

Our first contribution is to formally introduce two approaches for dealing with spelling variation without reference to a standard language, namely *spelling variant detection* and *simplification* in Chapter 3. To this end we also define what counts as spelling variant in this thesis and give formal definitions for common ways of dealing with spelling variation using a standard language, i. e., *normalization* and *non-standard variant detection*. These are contrasted with *spelling variant detection* and *simplification*.

Our second contribution is to create and evaluate implementations of spelling variant detection and simplification. For the implementation, we use ML as it is common in modern NLP (Goldberg 2017, p. xvii). In Chapter 4, we present the methods from ML that are used in this thesis, thereby providing a basic understanding of these methods for researchers without a background in ML.

The main focus of the following chapters is to introduce a pipeline for detecting spelling variants of a given type (or token) in a data-driven way by first generating a set of spelling variant candidates and then filtering this candidate set. An implementation of all methods presented in this thesis is made available at `https://github.com/fab-bar/SpellvarDetection` making the approaches readily usable (cf. Appendix B). This part consists of three chapters:

In Chapter 5, two settings for evaluating spelling variant detection are presented. These two settings are used to compare different approaches for simplification and spelling variant detection. We have designed these settings in order to approximate the utility of the approaches for searching in texts and for improving the performance of NLP tools.

In Chapter 6, different simplification approaches are introduced and compared.

Chapter 7 presents experiments with our pipeline for spelling variant detection.

Our third contribution is to evaluate methods on adapting machine learning models for NLP tasks to non-standard data in Chapter 8. There, we look at POS tagging and lemmatization, two basic NLP applications, in the context of low-resource non-standard text scenarios, i. e., the situation where texts should be annotated automatically with POS tags or lemmas with only a small amount of training data for the task available. We look at adapting tools to the non-standard nature of the data, i. e., allow them to handle spelling variation. Furthermore, we look at ways to

use the pipeline for spelling variant detection presented in Chapter 7 in order to improve the results of POS tagging and lemmatization. With our experiments we show that spelling variant detection and simplification are useful for improving the performance of a POS tagger and a lemmatizer in this context. The experiments with POS tagging are found in Section 8.3. In Section 8.4, we present the experiments with lemmatization. The scripts used to run the POS and lemmatization experiments are available at `https://github.com/fab-bar/thesis-POS_lemma_experiments`. Our additions to the tool for lemmatization Lemming are available at `https://github.com/fab-bar/cistern/`.

Chapter 9 concludes this thesis and points to possible directions for future research.

1.3. Publication Record

In this section, we list the parts of this thesis that have been previously published. A list of our related publications from the time working on this thesis is given in Appendix A.

Section 1.1 contains parts of Section 1 from Barteld, Biemann, and Zinsmeister (2019) and Section 1 from Barteld (2017).

Section 2.2 contains parts of Section 3 from Barteld, Biemann, and Zinsmeister (2018).

Section 2.3 contains parts of Section 4 from Barteld, Biemann, and Zinsmeister (2019). Section 2.3 contains parts of Section 3 from Barteld, Biemann, and Zinsmeister (2018).

Chapter 3 is based on Section 3 of Barteld (2017) and Section 2 of Barteld, Biemann, and Zinsmeister (2019).

Chapter 5 is a revised version of Section 5 from Barteld, Biemann, and Zinsmeister (2019).

Chapter 6 is based on Section 8.2 from Barteld, Biemann, and Zinsmeister (2019), the contained experiments and results have been updated in the context of this thesis and with an updated version of the code.

Chapter 7 is an updated and expanded version of Section 6 to Section 10 from Barteld, Biemann, and Zinsmeister (2019). Section 7.1.1 and Section 7.1.3 present new experiments with n-gram similarities and a filter based on Brown clusters presented first in Barteld, Schröder, and Zinsmeister (2015) and Barteld, Schröder, and Zinsmeister (2016) and contain parts from these papers. Section 7.2.2 adds new experiments with the surface-based filters presented in Barteld, Schröder, and Zinsmeister (2016) and Barteld (2017) as well as experiments with the binary-classification filter from Barteld (2017). It also contains parts from these papers.

Section 8.2 contains parts of Section 2 from Barteld, Biemann, and Zinsmeister (2018). Section 8.3 is based on Section 4, Section 5, Section 6 and Section 7 from Barteld, Biemann, and Zinsmeister (2018). Section 8.5 contains parts of Section 8 from Barteld, Biemann, and Zinsmeister (2018).

Chapter 2.

The Data

In Section 1.1, we have argued that normalization or similar approaches to spelling variation that use a standard language get harder with increasing differences between the non-standard and the standard data. For our experiments with approaches to spelling variation without the reference to a standard, we use texts from Middle Low German (GML) and Middle High German (GMH). Schulz (2018) uses GMH as an example for non-standard data where normalization "is a problem that is difficult to model" (p. 89) as it differs considerably from modern Standard German. GML even differs more from modern Standard German as it comprises the Low German dialects in contrast to the High German dialects, from which modern Standard German developed.

Recently, four projects have created and released annotated corpora for different variants of historical German (Dipper 2015, pp. 522–523) that allow for experiments on spelling variant detection. For the experiments in this thesis, we use two of these datasets: the Reference Corpus Middle Low German/Low Rhenish (1200–1650) (ReN) (Schröder 2014) containing GML texts and the Reference Corpus Middle High German (1050–1350) (ReM) (Petran et al. 2016) containing GMH texts. In this chapter, we give a short overview of the history of the German language and its different dialects and describe the two corpora.

2.1. A Short Introduction to Middle High and Middle Low German

As has been pointed out in Section 1.1, modern written German is standardized by rules, the orthography. The current set of rules is given by Rat für deutsche Rechtschreibung (2018). However, an orthographic convention for the German language was first established only in 1901/02 (Wegera, Waldenberger, and Lemke 2018, p. 80). Before that, there was merely a discourse about the implementation of a correct spelling but not a standardization as there is today. This discourse reached a greater public through the rise of the printing press in the second half of the 15^{th} century. Rule sets for orthography were developed and finally culminated in the

orthographic convention mentioned above that was established in the beginning of the 20th century and was largely based on the rule set developed by Konrad Duden in 1872 and 1880 (Wegera, Waldenberger, and Lemke 2018, p. 81).

Hence, before the orthographic convention, writers had to decide how to spell words. Doing that they had to choose between different alternatives, e. g., by following the examples of other writers, local rule sets, e. g., those by the printing office, or trying to write as they would speak (Wegera, Waldenberger, and Lemke 2018, p. 78). This choice could be different for different words in a text—it could also be different for appearances of the same word in a text. Recent studies on spelling variation in historical texts even suggest the deliberate use of variants by a writer assuming the aesthetic idea of variation as a guiding principle (*variatio delectat*) (Wegera, Waldenberger, and Lemke 2018, p. 80). All this led to the spelling variation that is observable in historical German texts.

When looking at the corpora as a whole instead of single texts, the variation that appears is amplified by dialectal differences between writers. At every point of its history, the German language has been a conglomerate of different dialects that can be subsumed under two broad dialect groups (Wegera, Waldenberger, and Lemke 2018, p. 18): the High and the Low German dialects that are represented in the ReM and the ReN. The Low German dialects are the dialects in the northern part of the German-speaking area and the High German dialects are the dialects in the southern part. Dialects belonging to the same of the two groups share many commonalities. Figure 2.1 on the facing page gives a schematic overview of dialect areas and their location in the German-speaking area.

The main commonality for the dialects in the Low German group—and at the same time the main distinction from the High German dialect group—is the absence of the second Germanic (or High German) consonant shift. This consonant shift is a collection of pronunciation changes. The main changes affect the three consonants /p/, /t/ and /k/ that shift to corresponding fricatives or affricates like /ff/ and /t͡s/ (cf. Nübling et al. 2017, pp. 42). The High German consonant shift supposedly took place between the 5th and 7th/8th century (Wegera, Waldenberger, and Lemke 2018, p. 134). It is one of the main features that distinguishes (High) German from other Germanic languages like English. This can be seen when looking at word pairs from modern English and German like *ape* – *Affe* and *apple* – *Apfel*. Since the Low German dialects did not undergo the consonant shift, they share features with English as well as with modern Standard German.

The border between a dialect in which a shift has happened and a dialect in which the same shift has not happened for certain words is called isoglosse. The High German and the Low German dialects are separated by the Benrath isogloss, named after Benrath, a part of Düsseldorf. This isogloss is also called the *maken-machen* isogloss after the forms of the German word *machen* ('to make'). In dialects that are spoken to the north of the Benrath isogloss, the Low German form *maken*

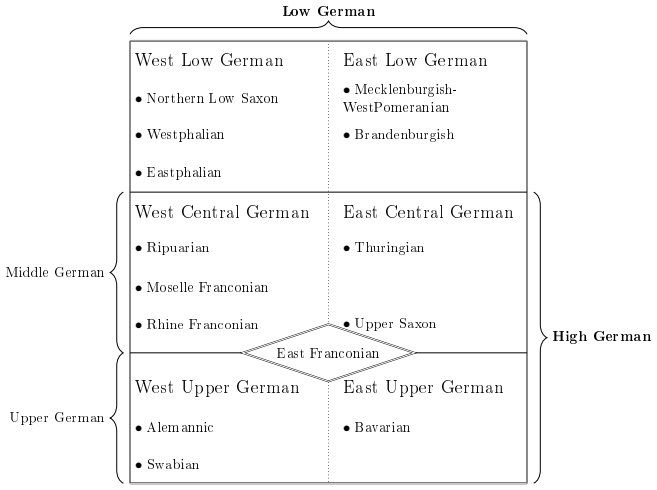


Figure 2.1.: Schematic overview of the dialects of German (translated version of Figure 1.5 from Wegera, Waldenberger, and Lemke 2018, p. 19)

is used. In dialects to the south of this isogloss, the form *machen* is found (Wegera, Waldenberger, and Lemke 2018, p. 135).

Despite this commonality between the dialects in the High and the Low German dialect group respectively—which allows to group these dialects into the two broad dialect groups in the first place—one has to keep in mind that these groups are not homogeneous and there are clearly regional spelling variants (cf. XVII. Regionalsprachgeschichte in Besch et al. 2003).

So the ReM and the ReN both collect texts from different dialects leading to spelling variation between the different texts—alongside the variation in texts that exists due to the lack of an orthography. However, the lack of a norm and the different dialects are not the only reasons for spelling variation in the corpora: both the ReM and the ReN also contain texts from a period of over 200 years. When considering that language change is not a sudden and abrupt change but a gradual and constant process (Wegera, Waldenberger, and Lemke 2018, pp. 24), the periods covered in the corpora include a lot of changes that are also visible as spelling differences.

The history of the High German dialect group is usually divided into four periods: the Old High German, the Middle High German (GMH), the Early New High German and New or Modern High German periods. The GMH period, from which the texts in the ReM are, is between 1050 and 1350. So, while the GMH dialects are predecessors of modern Standard German, they differ considerably from modern Standard German due to their diachronic distance. This makes normalization to

standard German possible but hard (Schulz 2018, p. 89). Research about normalization often considers more recent historical predecessors. One example are the datasets compiled for the large scale study by Bollmann (2018; 2019) where the oldest texts (German and overall) are from the 14th century (the Early New High German period). The datasets provided by Bollmann (2018) have also been used in other recent normalization studies (e. g., Makarov and Clematide 2020). Therefore, due to their diachronic distance to modern Standard German, the texts in the ReM are good candidates for using alternative ways of dealing with spelling variation.

However, regarding normalization of GMH, there is an alternative to normalizing to modern Standard German: 19th century philologists edited GMH texts in order to reconstruct a standard GMH supposedly used by poets (Kragl 2015, p. 3). While using such normalized editions as basis of linguistic studies has been criticized since it leads to the description of an artificial GMH (Wegera 2000), the language of these editions is a standardized GMH—whether or not it has been used by poets or is artificial. The ReM includes such a normalized version of the texts. This version has been used to train a POS tagger by Schulz and Ketschik (2019, pp. 849). We compare normalizing the texts to that artificial standard GMH before training and applying a tool for automatic annotation with using our approaches that do not use a standardized target language in order to improve a POS tagger in Section 8.3 and a lemmatizer in Section 8.4.

For Low German, three periods are distinguished, which are similar to the periods for High German: Old Low German, Middle Low German (GML) and New Low German. However, the periods are dated differently: The GML period overlaps with but is not the same as the GMH period. It ranges from 1200 to 1650.

GML as the written language used by the Hanse was important as a cross-regional commercial language for northern Europe, especially in the 14th and 15th century (Sanders 1982, p. 145). Due to this role, there were standardization tendencies. However, there were still regional variations that are characteristic for different dialects (Niebaum 2000, p. 1422). With the rise of the printing press in the 15th century, the Low German dialects lost their relevance (see König, Elspaß, and Möller 2005, p. 103 for a short overview of the development of the Low German dialects). It was finally replaced by High German in the 16th century (Sanders 1982, p. 161). Consequently, no standardized version of the Low German dialects exists today. This leaves modern Standard German, Dutch or probably English as the closest related standard language. Furthermore, the modern Low German dialects are low-resourced with only recently started efforts to create annotated corpora (Siewert, Scherrer, and Tiedemann 2021). Combined with the diachronic distance this makes the texts in ReN good candidates for experiments as described in this thesis.

In the following sections, we describe how the texts are presented in the corpora and which parts of the corpora are used in the different experiments.

2.2. The Reference Corpora of Historical German

As described above, the corpora used in this thesis have been created as part of an initiative to create reference corpora of historical German. This initiative comprises the four corpora Reference Corpus Old German (ReA, 750–1050)[6], Reference Corpus Early New High German (ReF, 1350–1650)[7], and the aforementioned ReM and ReN. The texts for the corpora have been carefully selected in order to represent the respective historical variant of German (cf. Lemnitzer and Zinsmeister 2015, Section 7.2 for a definition of reference corpus). The selection of texts has been described by Petran et al. (2016) for the ReM and by Barteld et al. (2017) for the ReN.

For both the ReM and the ReN, the selected texts have been manually transcribed from the original manuscripts or prints, tokenized and split into sentences. Following Dipper et al. (2013, p. 3) both corpora distinguish two tokenizations of the underlying text: a *diplomatic* tokenization that follows the usage of whitespace in the text and a so-called *modernized* tokenization. While the diplomatic tokenization is useful to recreate the original form of the text including the usage of whitespace as well as line and page breaks, the *modernized* tokenization is more similar to what is usually meant by tokenization: a segmentation of the text into words (following a mostly syntactic view).

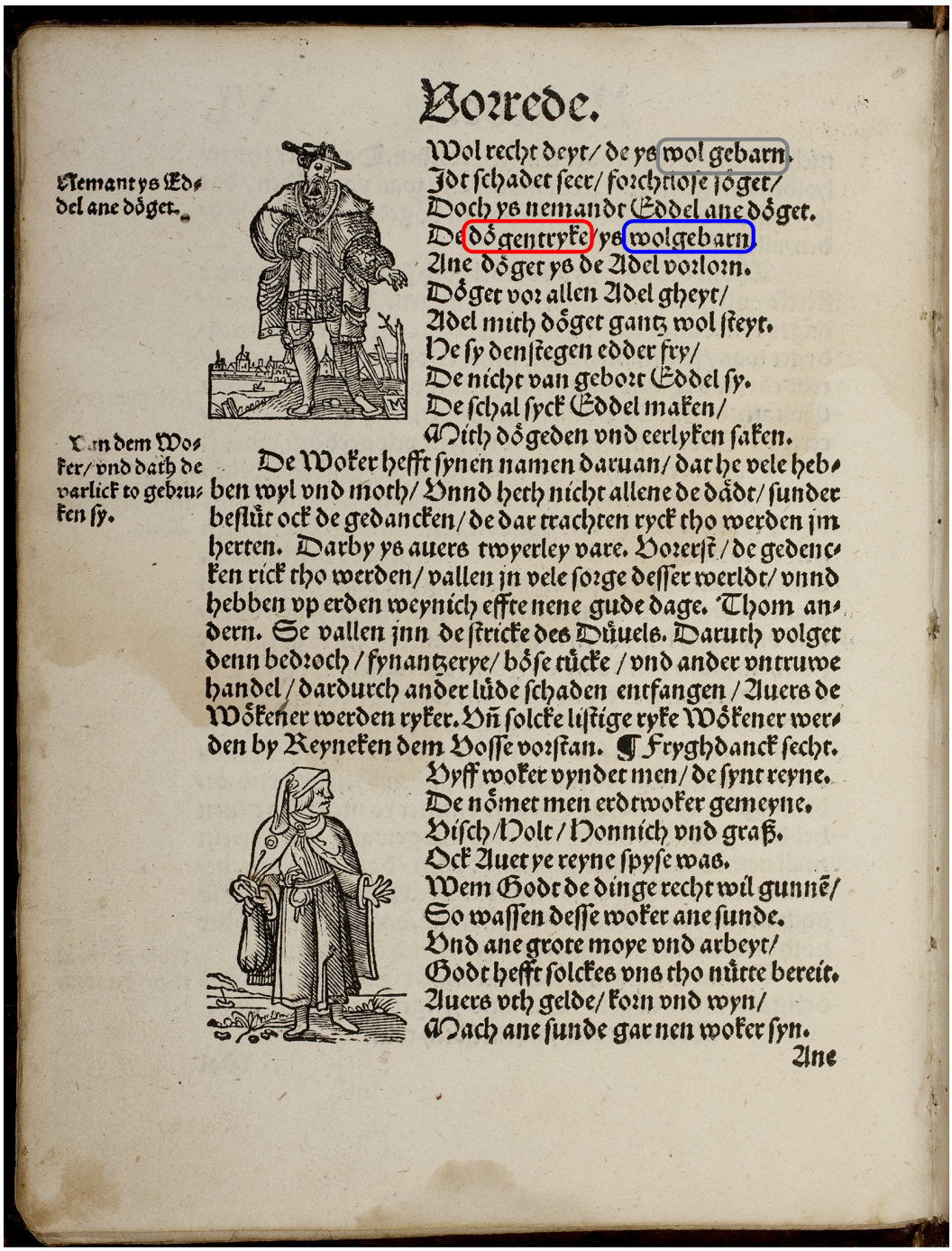
The difference between these tokenizations can be seen with two examples from *Reinke de Vos 1539* in Figure 2.2 on the next page. In the ReN, the part marked in red corresponds to the two tokens *dō̆* and *gentryke* in the diplomatic tokenization because of the whitespace between *ō̆* and *g*. In the modernized tokenization it corresponds to one token, the word *dō̆gentryke* ('full of virtue'). The other example for differing *diplomatic* and *modernized* tokenizations is given by the part marked with blue. Due to the absence of whitespace it corresponds to the diplomatic token *wolgebarn* but to the two tokens *wol* ('good') and *gebarn* ('born') in the modernized tokenization. The two words appear with whitespace between them earlier on the page as well, marked with gray in Figure 2.2. Figure 2.3 on page 20 shows the transcription of these two examples in the ReN. The diplomatic tokenization is labeled with *tok_ dipl*, the modernized tokenization with *tok_ anno*. Please note that the two tokenizations only differ in the segmentation of the transcription. The tokens from the modernized tokenization are then annotated. We use this segmentation for all of our experiments and refer to it simply as the tokenization of the texts.

The corpora present the modernized tokens in three different versions, which we exemplify with the word *bift* ('(you) are'):[8]

6. `http://www.deutschdiachrondigital.de/`, last visited October 4, 2021.

7. `https://www.linguistics.rub.de/ref/`, last visited October 4, 2021.

8. They are called *trans*, *utf* and *ascii* in the CorA-XML format (`https://cora.readthedocs.io/en/latest/document-model/#token-representations`, last visited October 4, 2021), which is the main file format in which the corpora are published.



Vorrede.

Nemant ys Edel ane döget.

Wol recht deyt/ de ys wol gebarn.
Idt ſchadet ſeer/ forchtloſe jöget/
Doch ys nemandt Eddel ane dögeth.
De dögentryke/ ys wolgebarn
Ane dögeth ys de Adel vorlorn.
Dögeth vor allen Adel gheyt/
Adel mith dögeth gantz wol ſteyt.
He ſy denſtegen edder fry/
De nicht van gebort Eddel ſy.
De ſchal ſyck Eddel maken/
Mith dögeden vnd eerlyken ſaken.

Van dem Woker/ vnd dath de darlick to gebruken ſy.

De Woker hefft ſynen namen daruan/ dat he vele hebben wyl vnd moth/ Vnnd heth nicht allene de dädt/ ſunder beſlůt ock de gedancken/ de dar trachten ryck tho werden jm herten. Darby ys auers twyerley vare. Vorerſt/ de gedencken rick tho werden/ vallen jn vele ſorge deſſer werldt/ vnnd hebben vp erden weynich effte nene gude dage. Thom andern. Se vallen jnn de ſtricke des Důuels. Daruth volget denn bedroch/ fynantzerye/ böſe tůcke/ vnd ander vntruwe handel/ dardurch ander lůde ſchaden entfangen/ Auers de Wökener werden ryker. Vñ ſolcke liſtige ryke Wökener werden by Reyneken dem Voſſe vorſtan. ¶ Fryghdanck ſecht.

Vyff woker vyndet men/ de ſynt reyne.
De nömet men erdtwoker gemeyne.
Viſch/Holt/Vonnich vnd graß.
Ock Auer ye reyne ſpyſe was.
Wem Godt de dinge recht wil gunnē/
So waſſen deſſe woker ane ſunde.
Vnd ane grote moye vnd arbeyt/
Godt hefft ſolckes vns tho nůtte bereit.
Auers vth gelde/ korn vnd wyn/
Mach ane ſunde gar nen woker ſyn.

Ane

Figure 2.2.: Page 6^{v} from 'Reynke Vosz de olde' (Rozstock: Dyetz, 1539; Borchling & Claussen 1312) – Library: Niedersächsische Staats- und Universitätsbibliothek Göttingen, Signature: 8 P GERM II, 1413 RARA, `http://resolver.sub.uni-goettingen.de/purl?PPN633656895`

1. **transcription**: This version uses specific markup developed in the projects and encodes a lot of the peculiarities of the script, e. g., **(b*)i$t/*, where **(X*)* indicates the usage of an initial, *$* encodes a long s (ſ), which was used nearly interchangeably with the graph *s*, and / encodes that there is no space after this word.

2. **strict**: This version encodes many of the peculiarities using Unicode, but does abstract away from features of the script such as initials, e. g., *biſt*.

3. **simple**: This version removes variation mainly by mapping non-ASCII characters to their ASCII counterparts, e. g., the long s (ſ) to s, as in *bist*. This version has been created with rule-based mappings. Basically, it is a version of the texts with removed spelling variation by applying a simplification as defined in Chapter 3.

For our experiments, we use the *strict* and the *simple* version. The *strict* version captures a lot of the spelling variation in the texts while not using project-specific markup. Models trained on this version will be more useful with other resources. The *simple* version is used as a reference for experiments on reducing spelling variation with rule-based simplifications. With the ReN, *simple* does not deviate much from *strict*. For this corpus, we use the rules presented by Koleva et al. (2017) to prepare an alternative simplified version. This is described in more detail in Chapter 6. Capitalization is ignored in all experiments.

These two corpora are large resources of historical German and allow for training NLP tools for GMH and GML with a good amount of training data. Yet, for researchers working with historical texts from other periods or from specific genres, training data is sparse. This is why in this thesis we are interested in low-resource settings, using only subsets of the corpora. For the POS tagging and lemmatization experiments in Chapter 8, for example, we limit ourselves to a selection of six texts from each of the corpora and train the taggers on only about 12,000 tokens.

In the next two sections we describe the ReN and the ReM in more detail.

2.2.1. The Reference Corpus Middle Low German/Low Rhenish

The ReN (`http://referenzkorpus-mnd-nrh.de`, last visited October 4, 2021), the main dataset used in this thesis, has been created in a project funded by the German Research Foundation between 2013 and 2019. This project was a cooperation between the universities of Hamburg (led by Ingrid Schröder) and Münster (led by Robert Peters). The corpus design and its motivation has been described in Peters and Nagel (2014) and Schröder (2014).

Version 1.0 of the dataset (`http://hdl.handle.net/11022/0000-0007-D829-8`) contains 146 texts with 1,415,362 tokens annotated with sentence boundaries, lem-

⊟ All

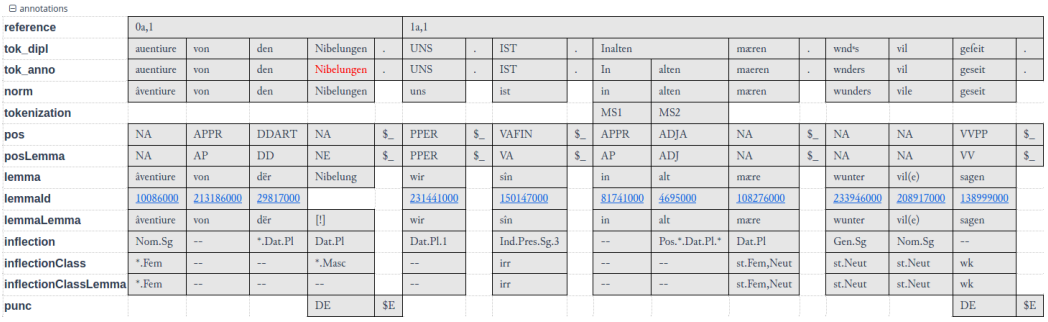
token	De	dǫ̂#gentryke		/	ys	wol§gebarn		.
tok_dipl	De	dǫ̂	gentryke	/	ys	wolgebarn		.
reference	06v,05							
tok_anno	De	dǫ̂gentryke		/	ys	wol	gebarn	.
lemma_wsd	dê¹,dê¹,dat²a	dǫ̂̄gentrīk		--	wēsen²	wol²	(ge)bēren¹	--
lemma	dê,dê,dat	dǫ̂̄gentrīk		--	wēsen	wol	(ge)bēren	--
lemma_simple	de,de,dat	dogentrik		--	wesen	wol	(ge)beren	--
lemma_var	de,de,dat	dogentrik		--	wesen	wol	beren,geberen	--
pos	DDARTA	ADJS		$;	VAFIN	AVD	VVPP	$;
posLemma	DD	ADJ		$;	VA	AVD	VV	$;
morph	Masc-Fem.Nom.Sg	Pos.Masc-Fem.Nom.Sg.*		--	Irr.3.Sg.Pres.Ind	--	--	--
comment		Pos.Masc.Nom.Sg.Sw, Pos.Fem.Nom.Sg.*						
bound_sent	Satz							

Figure 2.3.: Example from 'Reinke de Vos 1539' with annotations from the Reference Corpus Middle Low German/Low Rhenish (1200–1650)

mas and POS tags including morphological information using the Historisches Niederdeutsch Tagset (HiNTS) (Barteld et al. 2018), a tagset specifically developed for GML based on the Historisches Tagset (HiTS), which has been used for the ReM (Dipper et al. 2013). Figure 2.3 shows an example from 'Reinke de Vos 1539' (compare Figure 2.2 on page 18) with all annotations from the ReN as they are visualized in ANNIS (Krause and Zeldes 2016), the query and visualization system that can be used to work with the reference corpora. In addition to the fully annotated texts, the ReN 1.0 contains additional 89 texts annotated only with sentence boundaries consisting of 908,682 tokens.

One of the main aims of the corpus design is to create a resource for grammatical analyses of GML. Examples of how to use the corpus for analyses of syntactic phenomena can be found in Barteld et al. (2019) and Ihden (2020). Furthermore, there is an ongoing project to create a grammar of GML based on the ReN. In the context of this project new annotated texts are added to the corpus as well (cf. Ihden and Schröder 2021 and the already published version 1.1 of the dataset, `http://doi.org/10.25592/uhhfdm.9195`).

During the development of the corpus finished texts have been released on a regular basis to allow the usage of these texts for research before the final release. All in all there have been 9 releases before version 1.0. Note that the experiments presented in this thesis have been conducted with different pre-release versions of the ReN, since this thesis was created during the process of the corpus development. We point to the specific versions when describing the datasets in Section 2.3 and Section 2.4.



annotations																	
reference	0a,1					1a,1											
tok_dipl	auentiure	von	den	Nibelungen	.	UNS	.	IST	.	Inalten		mæren	.	wnd^s	vil	gefeit	.
tok_anno	auentiure	von	den	Nibelungen	.	UNS	.	IST	.	In	alten	maeren	.	wnders	vil	geseit	.
norm	âventiure	von	den	Nibelungen		uns		ist		in	alten	mæren		wunders	vile	geseit	
tokenization										MS1	MS2						
pos	NA	APPR	DDART	NA	$_	PPER	$_	VAFIN	$_	APPR	ADJA	NA	$_	NA	NA	VVPP	$_
posLemma	NA	AP	DD	NE	$_	PPER	$_	VA	$_	AP	ADJ	NA	$_	NA	NA	VV	$_
lemma	âventiure	von	dër	Nibelung		wir		sîn		in	alt	mære		wunter	vil(e)	sagen	
lemmaId	10086000	213186000	29817000			231441000		150147000		81741000	4695000	108276000		233946000	208917000	138999000	
lemmaLemma	âventiure	von	dër	[!]		wir		sîn		in	alt	mære		wunter	vil(e)	sagen	
inflection	Nom.Sg	--	*.Dat.Pl	Dat.Pl		Dat.Pl.1		Ind.Pres.Sg.3		--	Pos.*.Dat.Pl.*	Dat.Pl		Gen.Sg	Nom.Sg	--	
inflectionClass	*.Fem	--	--	*.Masc		--		irr		--	--	st.Fem,Neut		st.Neut	st.Neut	wk	
inflectionClassLemma	*.Fem	--	--	--		--		irr		--	--	st.Fem,Neut		st.Neut	st.Neut	wk	
punc				DE	$E											DE	$E

Figure 2.4.: Example from the 'Nibelungenlied' with annotations from the Reference Corpus Middle High German (1050–1350)

2.2.2. The Reference Corpus Middle High German

We use the ReM as additional dataset in the experiments with POS tagging (Section 8.3) and lemmatization (Section 8.4). The ReM consists of 394 texts with 2,448,379 tokens annotated with POS tags, morphology and lemma. The tagset used for the POS annotation is the HiTS, which has been specifically developed for the reference corpus projects (Dipper et al. 2013). The HiTS has been derived from the STTS (Schiller et al. 1999), a tagset for modern Standard German. Figure 2.4 shows the beginning of the 'Nibelungenlied' from the ReM as visualized in ANNIS (Krause and Zeldes 2016) (see Figure 1.1 on page 4 for the corresponding manuscript page).

In addition to the annotation that can be found in the ReN as well, the ReM also contains a normalized GMH form. Importantly—as has been pointed out above—this normalization is not modern Standard German but this annotation layer "contains automatically-created word forms that closely correspond to word forms as used in traditional editions of historical manuscripts in German" (Petran et al. 2016, p. 4). The script that creates the automatic normalization uses the text as well as lemma, POS tag and morphology annotation (cf. Klein and Dipper 2016, p. 8). We use this normalization to compare reduction of spelling variation using a normalization approach with an approach using spelling variant detection as presented in Chapter 7 in order to improve POS tagging and lemmatization.

2.3. Datasets Used for Simplification and Spelling Variant Detection

For the experiments with simplification and spelling variant detection in Chapter 6 and Chapter 7 we have used the ReN. Most of the experiments have been con-

ducted with Version 0.3 (http://hdl.handle.net/11022/0000-0006-473B-9) of the corpus. The one exception are the experiments with different hyperparameters for the Support Vector Machine (SVM)-based filter in Section 7.2.2. They have been conducted with Version 0.1 of the corpus (http://hdl.handle.net/11022/0000-0001-B002-5).

Version 0.3 consists of 32 texts with 200,664 annotated tokens. We split the data into training (24 texts, 160,240 tokens), development (6 texts, 20,736 tokens) and test (2 texts, 19,688 tokens) sets.[9] Texts are not split between training, development and test set, but included as whole into one of the three sets. This simulates a situation, where the spelling variants are known for a given set of texts—the training set—and spelling variant detection is to be applied to new texts. While the texts in the training set are from different language areas with 11 texts from the North Low Saxon area, the texts in the development and test sets are all from the North Low Saxon area.

The development set contains 3,991 types. 1,959 of them do not appear in the training set but almost half of them (841) have spelling variants in the training set. The test set has a slightly less diverse vocabulary. It contains only 3,063 types, 1,681 of them do not appear in the training set, 623 of them have spelling variants in the training set. This means that by mapping all unknown or OOV types from the development and test data to spelling variants in the training data could reduce the OOV rates of 0.49 (development) and 0.55 (test) to 0.28 (development) and 0.35 (test), respectively.

As a background corpus (cf. Section 4.1.4), which we use for the computation of Brown clusters and word embeddings (cf. Section 4.8.2) for the experiments in Chapter 7, we have used transcripts of 69 other texts from the ReN. Since the transcription and annotation of these texts had not been finished when we conducted the experiments, these texts have not been released in that version. Furthermore, these texts where not manually tokenized when we were compiling the background corpus. The texts therefore have been split at whitespace and special characters leading to 1,730,614 tokens. This however overestimates the size of the background corpus since the whole corpus with 146 texts consists of only 1,414,362 tokens when manually tokenized. Using this tokenization will lead to the situation that some types might appear in the background corpus but are split by the simple automatic tokenization and are therefore not recognized. But this is a realistic setting as a background corpus will often not be manually tokenized.

9. See Section 4.2 for how training, development and test sets are used for evaluating different approaches.

2.4. Datasets Used for Part-of-Speech Tagging and Lemmatization

For the experiments with POS tagging and lemmatization in Chapter 8 we have used texts from the ReM 1.0 (`http://islrn.org/resources/332-536-136-099-5`) and the ReN 0.6 (`http://hdl.handle.net/11022/0000-0007-C64C-5`) for training, development and test as well as background data.

We limit ourselves to a selection of the data in these releases to emulate a low-resource scenario. Schulz (2018) shows for GMH that the learning curve of a statistical POS tagger flattens after 12,000 tokens. After that, adding 2,000 tokens more as training data only leads to small improvements below 1%. This suggests that 12,000 tokens would be a good start in a setting where training data for a tagger needs to be generated.

For our selection, we have picked texts that come from similar points in time and dialects to minimize the amount of spelling variation that is due to temporal and dialectal differences. For the ReM, we limit our selection to texts from its MiGraKo subcorpus (Klein and Dipper 2016, p. 3) and use only prose texts from the upper German dialect area from the first half of the 13$^{\text{th}}$ century. This selection leads to six texts. For the ReN, the selected texts have been limited to texts from the 14$^{\text{th}}$ century. We have taken four texts from Northern Low Saxon and two texts from Eastphalian.

As training data, we use roughly the first 2,000 tokens (always using complete sentences) from each text of both datasets. This simulates the approach where a POS tagger for a low-resourced language is created by annotating the beginnings of texts used for training a model that automatically annotates the remaining parts of the texts. For development, we use the following 1,000 tokens and for testing the next 1,000 tokens (again, complete sentences).

Table 2.1 on the following page shows statistics on the datasets regarding POS tags and lemmas. The tagsets used for the POS annotation are fine-grained and therefore include many tags. One specificity introduced in HiTS—which has also been adopted for HiNTS—is the usage of two types of POS tags: a context-specific tag and a lexeme-specific tag. For our experiments, we use the concatenation of both tags as POS tag. For lemmatization, we use only the context-specific lemma for the ReM, for the ReN, we use the lemma without word sense disambiguation. Spelling variation is measured by giving the proportion of morphological words that are realized by more than one type in the data (cf. Section 3.1). *Strict* and *simple* refer to the different versions of the tokens described in Section 2.2. *Norm* refers to the normalization contained in the ReM described in Section 2.2.2.

	ReM	ReN
Number of tokens		
training set	12,108	12,025
development set	6,064	6,024
test set	6,062	5,667
Spelling variation in training set		
strict	22.81%	18.11%
simple	18.12%	16.81%
norm	5.56%	–
Number of POS tags in training set	79	70
Number of lemmas in training set	1778	1346

Table 2.1.: Statistics of the datasets used for the Natural Language Processing experiments

Chapter 3.

Spelling Variation

The predominant approach for dealing with spelling variation in non-standard data is normalization, i. e., transforming the non-standard data in order to make it more similar to standard data. For historical variants of modern standard languages, normalization can be defined as the task of mapping words from the historical text to their "*canonical cognates*, preserving both the root(s) and morphosyntactic features of the associated historical form(s), which should suffice (modulo major grammatical and/or lexical semantic shifts) for most natural language processing tasks" (Jurish 2010a, p. 72). This approach is often coupled with the aim to use resources that exist for the standard language, e. g., a POS tagger (Bollmann 2013b). Conversely, spelling variation can be approached by what we call non-standard variant detection (Pilz et al. 2006; Ernst-Gerlach and Fuhr 2006; Hauser and Schulz 2007), i. e., by finding the set of historical (or non-standard) words corresponding to a given contemporary or canonical query term. This approach can be found in the context of searching in historical texts (as e. g., Pilz et al. 2006).

However, both approaches use a given standard language as basis. If no closely related standard language is available—as it is the case with GML since modern Low German does not have a standardized orthography either (cf. Section 2.1)—these approaches cannot be applied in a straightforward manner or are hard to model. Furthermore, historical languages are often analyzed using differing categories from their modern counterparts. One example is the usage of specialized POS tagsets like HiTS (Dipper et al. 2013) and HiNTS (Barteld et al. 2018) that differ from the tagsets used for contemporary language data. In these cases, applying a POS tagger trained on modern data on a normalized version of historical texts would not lead to a text annotated with the POS tags as used by historical linguists complicating the adaptation of tools to historical texts. We look more closely at POS tagging for historical texts in Section 8.3.

In contrast to normalization and non-standard variant detection, we approach spelling variation without the reference to a standard language such that the presented methods are readily usable in the situations described above. The two alternatives to normalization and non-standard variant detection pursued in this thesis are simplification and spelling variant detection. When using simplification, spelling

variation is reduced by applying rewrite rules to the data. For spelling variant detection, the set of variants for a given token from the non-standard data is generated. Implementations and evaluations of these two approaches are presented in Chapter 6 and Chapter 7. In order to clarify the similarities and differences between normalization and non-standard variant detection on the one hand and simplification and spelling variant detection on the other hand, we give formal definitions for spelling variation and the related tasks in this chapter.

3.1. Defining and Quantifying Spelling Variation

According to Elmentaler (2018), the difference between historical and modern writing systems can be seen in the existence of variation: while readers and writers of historical texts accept variability, the orthography of modern texts is based on invariability (p. 14).[10] Spelling variation in historical texts has been illustrated with the two variants *unde* and *vnde* for the GML word *unde* ('and') in Example (1) on page 1.

Spelling variation is studied as part of graphematics, a branch of linguistics that explores spelling regularities in language systems (cf. Elmentaler 2018). The basic unit within the field of graphematics is the *grapheme*. Graphematics often relates such graphemes to *phonemes*, the smallest functional units of sound (Dürscheid 2016, pp. 134). Accordingly, Niebaum (2000), who presents an inventory of graphemes for GML, defines grapheme as "the smallest distinctive unit of written language that represents a phoneme (or sometimes a sequence of phonemes)" (pp. 1422, own translation[11]). As an example for a grapheme, Niebaum (2000) lists <*u*>. According to this study, this grapheme has, among others, the variants *u* and *v*. We have already seen this variation in Example (1a) on page 1 and, for GMH, in the Nibelungenlied (cf. Figure 1.1 on page 4).

Such variation can have various reasons. E. g., for GML vowel graphemes, Niebaum (2000) distinguishes four main types (p. 1424). For the purposes of this thesis, it suffices to give an overview of the types of phenomena that have to be covered for spelling variant detection or simplification. In order to do this, the following constructed pair of GML sentences illustrates three different types of spelling variation.[12]

10. In this thesis, we are only concerned with spelling variation and therefore ignore other aspects of standard and non-standard texts.

11. [...] wird im Rahmen dieses Beitrags das *Graphem* als kleinste distinktive Einheit geschriebener Sprache aufgefaßt, die ein *Phonem* (bzw. gelegentlich auch eine Phonemfolge) repräsentiert.

12. For a description of the abbreviations see Footnote 1 on page 1.

(4) Do he komen was van deme kloster
DO he ghecomen was uan dem kloster
when he come.PTC was from the cloister
'when he had been coming from the cloister'

The three types of spelling variation are: a) spelling variation in a narrow sense, i. e., two types for which the same pronunciation is assumed, b) spelling variation that corresponds to differences in the pronunciation, and c) spelling variation due to the presence or absence of a morphological marker. {*Do*, *DO*} and {*van*, *uan*} from Example (4) illustrate spelling variation in the narrow sense. {*deme*, *dem*} and {*komeen*, *gehcomen*} are variants, where the final <e> and the initial *ge* can be assumed to correspond to a difference in the pronunciation (cf. Lasch 2011, § 221 VI for a description of the variation with *ge*).

A clear-cut distinction between those two types of variation is not always possible: {*deme*, *dem*} could also be a spelling variant in the narrow sense, as it cannot be decided for a single instance if there actually was a difference in the pronunciation. Furthermore, the difference between a missing and a realized final <e> can also be seen as morphological variation, treating the <e> as the overt dative marker (cf. Lasch 2011, § 381 for feminine *i*-stems without *e* in the nominative).

Another source for variation that we also cover under the term spelling variation are spelling errors. Errors are usually defined as a deviation from a norm (Brill and Moore 2000). In the case of the lack of a norm as with GML such a definition is not applicable. Therefore, we define errors as a type of variant that is unlikely to appear, it may even appear only once. The GML corpus, for instance, contains one instance of *gesprok* as the past participle of *speak*, whereas other instances of the past participle are realized with the suffix *-en*. For this example, it can be assumed that this suffix was to be realized as an abbreviation, a dash over the *k*, and was simply forgotten. In practice, however, the distinction between spelling variation and spelling errors is often not possible in the context of a missing standard.

Regarding corpora containing historical texts, spelling variation can also result from errors in the transcription (done manually or with optical character recognition, cf. Amrhein and Clematide 2018). While these are definitely not variations that are of interest in historical graphematics, they lead to spelling variants in the data that should be discovered by an algorithm for spelling variant detection as well. Consequently, we cover not only variation that is strict spelling variation as it would be analyzed in historical graphematics but also variation that stems from potential errors—either in the manuscript or in the transcription.

In order to formalize what we cover under the term spelling variant, we use the definition of language as a system of binary signs that has been introduced into language science by Saussure ([1931] 2001). In general, a binary sign consists of a

signified and a signifying part. For Saussure ([1931] 2001), a language sign consists of the combination between a concept and a sound-image (pp. 76). For our purposes—since we are interested in spelling—we take possible spellings, *types*, as signifying parts. Furthermore, we are not interested in the actual concept signified by a type. Instead, we are concerned with the possibility of a given type to denote different concepts or not. Therefore, we do not aim to precisely represent the signified part of a sign, but we resort to *morphological words*. Morphological words are inflected word forms of a language. For defining spelling variation, we consider the signs making up a language to be the possible pairings of a *type* or *word (form)* and a *morphological word*. Please note that we use the terms *type* and *word (form)* interchangeably in this thesis due to the established usage of word in NLP, e. g., in word embedding (see Section 4.8) or OOV word.

Spelling variation as described above leads to the situation that in non-standard texts the same morphological word is often realized by multiple types, i. e., there are signs with the same *morphological word* signified by different *types*. For instance, in the ReN 1.0, the first person personal pronoun is realized by 38 different types.[13] A selection of frequent variants is given in Example (5).

(5) ic, ick, ik, jc, jck, jk, yck, yk

This example shows variation between *i*, *j* and *y* that appear in general mostly interchangeable in GML manuscripts and prints. Only bi-graphs like *ei* are an exception to this rule.

For our purposes, the notion morphological word can be operationalized by combining POS, morphological information and lemma including a word sense disambiguation if necessary: Each combination of these attributes is an abstract representation of a morphological word. For example *{Personal Pronoun, Nominative Sg., ik}* is the morphological word representing the nominative singular of the first person personal pronoun in GML. As *ik* is not ambiguous, no word sense disambiguation is shown. An example for an ambiguous lemma is the modern German type *Bank* meaning either 'bench' or 'bank'. Therefore, the lemma *Bank* should include word sense disambiguation.

The tendency to invariability that Elmentaler (2018) mentions for modern writing systems or standard texts can now be modeled by a mapping between *morphological words* and *types*: For a strictly standardized language, a unique mapping from a morphological word to a type would be expected. For instance, *{Personal Pronoun, Nominative Sg., I}* is usually realized as 'I' in Modern English. On the other hand, the GML morphological word *{Personal Pronoun, Nominative Sg., ik}* is realized

13. We ignore differences in the capitalization. Including capitalization there are 56 different types.

with—among others—the types given in Example (5), exhibiting the variation typically found in historical writing systems or non-standard texts.

More formally, given the types of a language $L \subseteq \Sigma^*$, where Σ denotes the alphabet, and a set of morphological words L_{morph}, we define the left- and right-total binary relation S, the spelling relation:

$$S \subseteq L_{\text{morph}} \times L \tag{3.1}$$

S relates all *morphological words* with their potential spellings and all *types* with the *morphological words* they can realize. We represent a morphological word as a tuple of POS, morphological information and a lemma (p, m, l). If S is functional, i. e., for each *morphological word* there is only one *type* as potential spelling, we say that the language represented by L is a strictly standardized language, otherwise we call it a non-standard language. In general S is not injective, i. e., there will be *types* that realize multiple morphological words, e. g., due to ambiguities as shown above with the German example *Bank* ('bench' or 'bank').

Example (6) shows GML sentences with *yk* and *ik* (variants of the lexeme *ik* shown in Example (5)) from the ReN 1.0 with the POS and morphological annotations for *yk* and *ik* using the HiNTS (Barteld et al. 2018) (cf. Section 2.2.1). These examples illustrate the variation between *yk* and *ik*. Example (6c) where *ik* is labeled as accusative is actually an instance where *ik* is falsely labeled as being accusative case. This shows that, when relying on the annotation in a corpus to identify spelling variants, annotation errors can be a reason for spelling variation.

(6) a. Vorwar segghe **yk** iuw .
PPER.1.Sg._.Nom
in truth tell I you.PL
I tell you in truth

b. vorwar segghe **ik** di .
PPER.1.Sg._.Nom
in truth tell I you.SG
I tell you in truth
(Lüb. Bibel 1494)

c. Dat wolde **ik** alle na halen
PPER.1.Sg._.Akk
that wanted I all make up for
I wanted to make up for that all
(Lüb. Dod. Dantz 1489)

Figure 3.1 on the next page shows the two GML types *ik* and *yk* and their relations to the two morphological words for the nominative and the accusative singular of

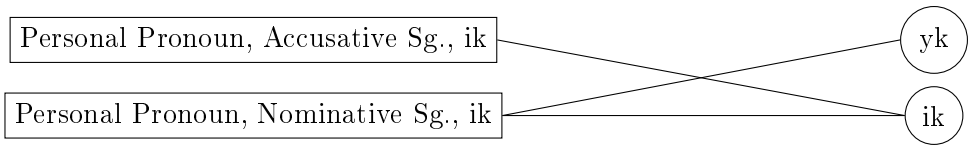


Figure 3.1.: Examples for Middle Low German morphological words and their spelling. Morphological words are shown in rectangles, types are shown in circles.

the first person personal pronoun *ik* according to the annotations in Example (6). The figure illustrates that the type *ik* is ambiguous since it can be used to realize both morphological words, while *yk* can only be used to realize the nominative singular. Looking at the morphological words, there is spelling variation regarding the nominative singular as both *yk* and *ik* can be used to realize the morphological word *{Personal Pronoun, Nominative Sg., ik}*. The accusative singular, in contrast, is only realized by *ik*.

In theory, we have a strict distinction between standard and non-standard languages: either there is a unique mapping from morphological words to types or there is not. In practice, however, there are hardly any languages that would be categorized as *standard language* according to this strict definition. Modern German, for instance, has certain words for which the writing norm allows different spellings, e. g., *Joghurt* 'yogurt', which can also be spelled *Jogurt*.[14]

Another example for spelling variation in modern Standard German is the usage of *ß* and *ss*. While there are rules governing their distribution (cf. Rat für deutsche Rechtschreibung 2018, § 25), there still is variation. The official norm even mentions the substitution of *ß* with *ss* when *ß* is not available, e. g., when it is missing on a keyboard (cf. Rat für deutsche Rechtschreibung 2018, § 25, E2).

Hence, in practice, there is no strict separation between standard and non-standard languages, but only a continuum of more or less variation in spellings. While there are examples like *Joghurt*/*Jogurt* and *ss*/*ß* in modern Standard German, most morphological words will be realized by only one type. In GML, however, there is substantially more variation. Even the first person personal pronoun shows variant spelling as shown in Example (6). So, it suffices for a language to be considered a standard language to have only a small amount of variation.

For our experiments, we approximate the spelling relation S (cf. Equation (3.1) on the preceding page) from an annotated corpus as follows: Given a corpus $C = (t_i)_1^n$ with n tokens t_i in which each token consists of a type $w \in L$, its associated POS (p), morphological information (m) and lemma (l), i. e., each token t_i is a tuple

14. See 'Joghurt' at *Duden online* (`https://www.duden.de/node/73447/revision/73483`, last visited October 4, 2021).

(w, p, m, l), we define the relation $S' \subseteq L_{\text{morph}} \times L$ on the basis of the corpus such that $((p, m, l), w) \in S'$ if $(w, p, m, l) \in C$. S' will only approximate S since a corpus might not contain all types that can potentially realize a morphological word. Furthermore, S' will often contain more variation than S either due to spelling errors or annotation errors (as already illustrated by Example (6)). We discuss further implications of the effect of annotation errors for the evaluation of spelling variant detection in Chapter 5.

The relation S can be used to define a spelling variant relation $SV \subseteq L \times L$ as follows:

$$(w, v) \in SV \Leftrightarrow \exists (p, m, l) \in L_{\text{morph}} : ((p, m, l), w) \in S \wedge ((p, m, l), v) \in S \quad (3.2)$$

That is, two types $w \in L$ and $v \in L$ are defined as spelling variants if they are spellings of the same *morphological word*, i. e., they can be used to represent the same *morphological word* in a text.

Compatible with the view of relating spelling variation in non-standard data to the corresponding standard data (cf. Section 1.1), quantification of spelling variation often quantifies the deviation from standard data. For example Baron, Rayson, and Archer (2009) mark all word forms that are not contained in a modern word list for English as spelling variants in order to quantify the extent of spelling variation in a corpus of Early Modern English. However, this quantification of spelling variation is not informative about the internal consistency of the data, it only quantifies the difference from the standard language. For NLP purposes this quantification informs us if a tool for the standard language might be applied to the data with some success but it is not informative about the inherent spelling variation in the data that makes training a tool on the non-standard data harder than on the standard data.

In contrast, we use S to quantify the amount of spelling variation in a language as the number of *morphological words* that have more than one spelling:

$$v := |\{m \in L_{\text{morph}} | \exists t_1, t_2 \in L : t_1 \neq t_2 \wedge (m, t_1) \in S \wedge (m, t_2) \in S\}| \quad (3.3)$$

When L, L_{morph}, and S are induced from a corpus, v can be given as a ratio since L_{morph} is necessarily finite. In this case, v quantifies the proportion of morphological words that are realized by more than one type. For this, we exclude morphological words that are instantiated only once in the corpus as they cannot exhibit possible variance.

Given these definitions of L, S and SV, we now discuss the different approaches to spelling variation in more detail.

3.2. Defining How to Deal with Spelling Variation

The task of finding spelling variants is similar to grouping *morphological words* with *lexemes*: A lexeme like the English verb *(to) be* appears in the form of different morphological words in texts, e. g., the present tense first person singular and the past tense 3rd person singular, realized by the types *am* and *was*. In lemmatization, the task is to assign the same lemma to the types realizing different morphological words of the same lexeme, thereby abstracting over inflectional differences. Similarly, the aim of spelling variant detection is to detect the types that belong to the same morphological word. However, while the morphological words belonging to the same lexeme differ with respect to their morphology, the types that belong to the same morphological word only differ in their spelling.

Given a set of morphological words L_{morph}, types of a non-standard language L and types of a standard language L_s with the corresponding spelling relations S_L and S_{L_s} (cf. Equation (3.1) on page 29)[15] and the power sets $\mathcal{P}(L)$ and $\mathcal{P}(L_s)$, we can define (type-based) normalization, non-standard variant and spelling variant detection more precisely:

- Normalization is the function $n : L \rightarrow \mathcal{P}(L_s)$ with

  $n(t) := \{t_s \in L_s | \exists m \in S_{L_s}^{-1}(t_s) : (m, t) \in S_L\}$

- Non-standard variant detection is the function $e : L_s \rightarrow \mathcal{P}(L)$ with:

  $e(t_s) := \{v \in L | \exists m \in S_{L_s}^{-1}(t_s) : (m, v) \in S_L\}$

- Spelling variant detection is the function $s : L \rightarrow \mathcal{P}(L)$ with:

  $s(t) := \{v \in L | \exists m \in L_{\text{morph}} : (m, t) \in S_L \wedge (m, v) \in S_L\}$

$S_L^{-1}(t)$ denotes the morphological words that can be realized by the type t, i. e., the set $\{m \in L_{\text{morph}} | (m, t) \in S_L\}$.

Normalization and non-standard variant detection are closely related, as the following definitions demonstrate. They show how a non-standard variant detection $e(t_s)$ can be induced given a normalization $n(t)$ and the other way around:

$$e(t_s) = \{v \in L | t_s \in n(v)\}$$
$$n(t) = \{t_s \in L_s | t \in e(t_s)\}$$

15. Please note that we assume for the set of morphological words L_{morph} that it is identical for the non-standard and the standard language. While this is not always the case, at least a substantial overlap between the sets of morphological words is needed when using normalization in order to use tools developed for the standard data on the non-standard data as described in Bollmann (2013b). To obtain a full set of morphological words, the union of the sets of morphological words for both languages can be used as L_{morph}.

Furthermore, a normalization induces a spelling variant detection $s_n(t)$:

$$s_n(t) = \{v \in L | n(t) \cap n(v) \neq \emptyset\}$$

However, while normalization and non-standard variant detection need to refer to the types of a language L_s, spelling variant detection does not need such a reference as can be seen by the presence and absence of L_s in the definitions given above.

So far, we have only defined type-based variants of these three approaches. The token-based variants operate on tokens, which we represent as types with a given left and right context. In the case of normalization the token-based version will return only one element from L_s. Usually, models for normalization are token-based in that sense, i. e., they output a single normalization for a given token. However, most of these models do not use the context to find the normalization and therefore normalize the same type to the same standard type irrespective of its context (cf. Bollmann 2019). For non-standard variant and spelling variant detection, the token-based versions return subsets of the type-based version, because some of the potential variants of the given type can be excluded given a specific context. We discuss token-based spelling variant detection in more detail in Chapter 5.

We want to use the quantification of spelling variance (Equation (3.3) on page 31) to introduce simplification more formally and to bridge the gap from the theoretical definitions given above and existing normalization approaches. For many applications, it is not necessary that the target language is an existing standard language. It suffices if the language resulting from the mapping exhibits less variation than the non-standard language. Such an operation is commonplace for Twitter data, i. e., the removal of repeated characters as a preprocessing step (Han, Cook, and Baldwin 2013). This removal reduces variation, which can be easily seen by looking at variations that users create for expressive reasons, e. g., *loool* and *loooool* instead of *lol*. This, however, is not a normalization in the sense of mapping to an existing standard since *lol* is not a standard word, a corresponding normalization would map *loool*, *loooool*, and *lol* to *laughing out loud*.

Bowers (1989) discusses this distinction in the context of creating critical editions of texts (compare also the discussion of normalization in Bollmann 2018), distinguishing *normalization* from *regularization*:

> The two means by which a critical editor creates some order from anomalous irregularities I label regularization and normalization. [...] I construe regularization as the bringing of inconsistent elements in a text into conformity by the adjustment of variants to some one regular form already present and assumed to be authorial. Normalization I conceive as imposing an external standard of regularity without the evidence of some specific precedent in the text being edited, but one that is guided by evidence derived from similar authorial documents.
>
> (p. 82)

While following this distinction, we use the term *simplification* instead of *regularization* due to the usage of *regularization* in ML (Hastie, Tibshirani, and Friedman 2009, Chapter 18).

More formally, a mapping r from the types of a language L to the types of a language L^* is a simplification if it reduces variation.[16] In this sense, normalization is a special kind of simplification where L^* contains the types of an existing standard language.[17] As mentioned above, when L^* is an existing standard language, this allows the usage of tools and resources that are available for L^* for L. While this is not the case for a general simplification, simplification still is beneficial: From the viewpoint of Information Retrieval (IR), a search term s given by the user can be mapped to $r(s)$ and the user can be presented with results for all types from L that are mapped to $r(s)$ as well, i. e., $r^{-1}(r(s))$ improving the recall of the search. From the viewpoint of NLP, L^* will have less variation, which results in less data sparsity and OOV words. Similar to normalization simplification also induces a spelling variant detection. Since simplification does not need the reference to a standard language, this is one way to achieve spelling variant detection in the absence of a standard language. Please note that we only consider token-based simplification, i. e., a simplification approach that maps a type $t \in L$ to one type $t_s \in L^*$. This does, however, not mean that the context in which t appears is used for the simplification.

In this section, we have defined the methods that we look at in the following sections of this thesis: simplification and spelling variant detection. As we have shown in this chapter, both do not refer to a standard language to deal with spelling variation. In the next section, we present the ML methods that we use to learn simplification and spelling variant detection from a given dataset.

16. Simplification does not mean that the resulting types are simpler in the sense that they are shorter. For a language with a variation between *g* and *gh* as in the GML words *gecomen* and *ghecomen*—the past participle of *kōmen* ('(to) come')—, adding an *h* after every *g* that is not already followed by one could be an example of a simplification as this change removes the variation between *g* and *gh*.

17. Compare also the remark by Jurish (2011) that "[t]he range of a canonicalization function need not be restricted to extant forms; in particular a phonetization function mapping arbitrary input strings to unique phonetic forms can be considered a canonicalization function in this sense" (p. 115). With our definitions, a phonetization function would be a simplification. However, we do not restrict (type-based) normalizations and simplifications to map to unique elements but allow for them to map to multiple elements.

Chapter 4.

Machine Learning Methods

In the previous chapter, we have defined spelling variant detection and similar methods to deal with spelling variation. We implement these approaches using techniques from ML and data mining, i. e., "techniques for finding patterns in data, patterns that provide insight or enable fast and accurate decision making" (Witten et al. 2017, p. 9). While knowledge about spelling variation extracted from a collection of texts either in the form of simplification rules (cf. Chapter 6) or in the form of a spelling variant relation can be used to understand the spelling variation, our focus lies in using the structures either for predicting spelling variants for given types, i. e., spelling variant detection (Chapter 7), or for improving the prediction of other properties of a token such as POS and lemma (Chapter 8). In this chapter, we give an overview of the methods and relevant concepts that we use for the automatic prediction of spelling variants as well as POS tagging and lemmatization and for evaluating the results.

Mitchell (1997, p. 2) defines learning in the context of computer programs by stating that the program or algorithm—the learner—improves its *performance* for a given *task* with *experience*. In the experiments for this thesis, the tasks encompass spelling variant detection or simplifications, POS tagging and lemmatization. The experience comes from training data containing spelling variation. Based on what information the training data contains, different kinds of learning are distinguished in the ML literature. We describe the different kinds that are relevant in the context of this thesis in Section 4.1. Performance is measured on development and test data that is different from the training data. Depending on the task at hand, we use different measures, which are described in Section 4.2.

From an ML perspective, we approach the different tasks tackled in this thesis with the following techniques: learning of rules for simplification (cf. Witten et al. 2017, Section 3.4), binary classification for spelling variant detection (cf. Goldberg 2017, Section 2.3.1), multiclass labeling for lemmatization (cf. Goldberg 2017, Section 2.3.3) and sequence labeling for POS tagging (cf. Goldberg 2017, Chapter 19).

We have defined simplification as the task to learn a mapping from a type to a different type with the objective that the amount of spelling variation in the lan-

guage is reduced (cf. Section 3.2). For the implementation, we treat simplification as the task of applying a set of rules that transform a given type into its simplified version. For learning such rules, we extract possible transformation rules for individual characters from the training data. These rules are then filtered using the *support*, i. e., how many instances of the training data the rule covers, and the *confidence*, i. e., how often the application of the rule actually leads to a spelling variant (Witten et al. 2017, Section 3.4). The approach is described in more detail in Chapter 6.

For spelling variant detection, we apply a filter that decides for a pair of words if they are actually spelling variants or not. For this, we use algorithms for binary classification that we describe in Section 4.3. Since the training data for this task has more negative than positive examples, we present approaches for dealing with such imbalanced training data in Section 4.4.

In contrast to the binary decision used in spelling variant detection, both POS tagging and lemmatization are multiclass classification problems: For each token, the best of multiple classes (the POS tags or the lemmas) is chosen. We describe multiclass classification in Section 4.5. However, the number of labels for lemmatization is huge (potentially infinite). This makes a naive approach hard to compute. Ways to find good labels for lemmatization are described in Section 4.6.

While it is possible to simply apply multiclass labeling approaches for POS tagging and lemmatization that output the label for an individual token, especially POS tagging approaches like those that we use below use methods that directly find an optimal sequence of labels for a sequence of words. We introduce sequence tagging or sequence labeling approaches in Section 4.7. For lemmatization, it is not as relevant to take the whole sequence into account when POS tags are used as features. This is because POS tags already provide a contextualized clue for the lemmatization. The tool that we use for lemmatization, therefore, uses a simple multiclass labeling approach. The specific tools that we use for POS tagging and lemmatization in the experiments with spelling variation are described in Section 8.1.

As pointed out above, ML algorithms take instances from the training data in order to improve their performance on the given task. One important aspect of learning from training instances is how the instances or examples are presented to the algorithms. Usually, this happens in the form of "values on a fixed, predefined set of features" (Witten et al. 2017, p. 53). We discuss features for spelling variant detection in Section 4.8.

4.1. Different Kinds of Learning from Data

The data that we use for learning to deal with spelling variants are historical texts. The specific datasets are described in Chapter 2. For all the experiments, we assume

that the texts are tokenized, i. e., segmented into a list of tokens. Depending on additional information available when training the ML algorithm, different types of learning from data are distinguished. In this section, we discuss learning types that are relevant in the context of this thesis.

4.1.1. Supervised Learning

Learning is called *supervised* when the training data contains information that includes the information to be learned. In this case, the knowledge about correct outcomes can be used to guide the learning process (Witten et al. 2017, p. 44). For spelling variant detection, it is the spelling variant relation for a given dictionary that should be learned. In our experiments with spelling variant detection using supervised learning, we assume that a collection of texts is available for which a spelling variant relation is known. The task is then to train an ML algorithm to identify spelling variants in these texts in order to generalize to other texts for which the spelling variant relation is not known.

In our specific case, we use annotations with lemma, POS and morphology to extract a spelling variant relation from the training texts (cf. Chapter 3). However, this spelling variant relation that we use to guide the learning only approximates the real (unknown) spelling variant relation: we can say that spelling variants that are in the relation also exist in the unknown spelling variant relation unless there are errors in the annotation, whereas pairs that are not in the spelling variant relation might actually be spelling variants according to the unknown true spelling variant relation.

Therefore, it is not a fully supervised learning scenario but more like a scenario where we are learning from positive and unlabeled examples (PU learning) (Li and Liu 2005). The learner acts under the assumption that the unlabeled examples (pairs that are not part of the spelling variant relation) are often actually negative examples, i. e., they are not spelling variants, but some of them are positive examples. We elaborate on this in Section 5.3.

4.1.2. Learning with Distant/ Weak Supervision

Sometimes neither the spelling variant relation for a given set of texts is known, nor are the texts annotated in a way that the relation can be extracted. However, there might still be information available that helps us to deduce or approximate the spelling variant relation. One example that we use is a text annotated only with lemmas. Two types that appear with the same lemma are either different morphological words of the same lexeme or they are spelling variants of the same morphological word. When using lemmas to deduce a spelling variant relation, this spelling variant relation will contain pairs of types that are not actually spelling

variants. Hence, we can apply supervised learning but have to be aware that our training data contains errors. This learning scenario has been called weak supervision since the training data is only "weakly labelled" (Craven and Kumlien 1999). In the context of relation extraction, Mintz et al. (2009) coined the alternative term distant supervision for this kind of learning. This term emphasizes that the supervision does not come directly from the data used to learn but leverages other (distant) resources. This concept has also been used for other tasks, e. g., POS tagging (Plank et al. 2014). Distant supervision is similar to PU learning but in distant supervision, there is *noise* in both sets not only in the unlabeled part, i. e., the positive examples may contain instances that are actually negative and the negative examples may contain instances that are positive.

We apply this learning scenario in Chapter 8 where the overall goal is not to learn a spelling variant relation but to learn POS tags and lemmas. We assume that for these tasks training data for supervised training is available (i. e., a collection of texts annotated with POS tags or lemmas) and evaluate the idea that the learning of POS tags or lemmas can be improved by providing information about spelling variants. While the explicit knowledge about the spelling variant relation for the training data might not be available, the training data for POS tagging or lemmatization can be leveraged to approximate a spelling variant relation leading to a weakly supervised learning scenario.

4.1.3. Unsupervised Learning

In supervised and weakly supervised learning the spelling variant relation is—at least approximately—known for the training data and used to guide the learner. In unsupervised learning, however, the spelling variant relation is not known. The learner can only use the tokenized text for learning how to identify spelling variants. This type of learning has also been called *structure discovery*, "the research line of algorithmic descriptions that find and employ structural regularities in natural language data" (Biemann 2012, p. 3). Structural regularities that are relevant for spelling variation are, for instance, their surface similarity or the context in which they appear in texts. We present approaches for unsupervised learning to identify spelling variants in Section 7.1. This type of learning is especially relevant for searching in a collection of texts when the spelling variant relation is not known and no training data that contains spelling variation which is similar to the variation in the target texts is available.

4.1.4. Semi-Supervised Learning

Semi-supervised learning is the combination of supervised and unsupervised learning. In this setting, the training data for supervised learning is augmented by

an unlabeled dataset, a background corpus. One type of semi-supervised learning is using structures discovered in the unlabeled data as features in the supervised learning. Thereby the performance of the learner can often be improved (cf. Miller, Guinness, and Zamanian 2004). This is especially helpful in the case where only a small amount of training data is available. We apply this approach when using word embeddings as features (see Section 4.8) for representing the context in which types appear. In order to compute the embeddings, we rely on a background corpus of unlabeled text.

4.2. Evaluating the Performance

In order to guide the learning in supervised settings and to compare different approaches, the performance of the learners has to be measured. For this, we apply standard metrics depending on the task. In this section, we describe the metrics for the performance of a learner that are used in this thesis.

In Chapter 6 and Chapter 7, we present experiments where for a given set of types their spelling variants are predicted and compared with known spelling variants as described in detail in Chapter 5. For this comparison, we use the standard metrics *precision* (p), *recall* (r) and F_1-*score* also called F_1-*value*, F_1 or sometimes simply *F-score*. They are defined as follows:

$$p = \begin{cases} \frac{tp}{tp+fp} & tp + fp > 0 \\ 1 & tp + fp = 0 \end{cases} \tag{4.1}$$

$$r = \frac{tp}{tp + fn} \tag{4.2}$$

$$F_1 = 2\frac{p \times r}{p + r} \tag{4.3}$$

Every type that is predicted to be a spelling variant for a given input can either actually be a spelling variant—a true positive—or not—a false positive. In Equation (4.1) and Equation (4.2) tp is the number of true positives and fp is the number of false positives. fn is the number of spelling variants that have not been predicted to be spelling variants, i. e., have not been discovered by the algorithm, the so-called false negatives. Precision (Equation (4.1)) measures the proportion of actual spelling variants among the predicted variants, while recall (Equation (4.2)) measures the proportion of predicted variants among the actual spelling variants. These two measures focus on different aspects. For recall, it is not relevant how many potential

spelling variants a measure predicts, but only how many of the existing spelling variants are found. Therefore, it is easy to get a good recall by simply predicting every type to be a spelling variant. For precision, on the other hand, it is not relevant how many of the actual spelling variants are found, but only how many of the predicted variants are correct. To get a high precision, it is a good strategy to only predict a few variants for which the certainty of being an actual spelling variant is high. For extreme cases, where nothing is predicted, i. e., tp and fp are both zero, p cannot be calculated. We set the precision to be 1 for such cases.

Because of the opposite effects that precision and recall measure, it is often their harmonic mean that is used to measure the overall performance of a learner. The harmonic mean measures the compromise between identifying spelling variants (recall) and at the same time not reporting too many false positives (precision). This mean is called F_1-score if both precision and recall are weighted equally.

In the following experiments, the spelling variants for multiple types or tokens are predicted. The different precision, recall and F_1 values for each of the types have to be combined to an overall average. There are different ways to do this. In this thesis, we use the so-called micro-average, i. e., we sum the number of true and false positives respectively negatives for all the types/ tokens before calculating the precision, recall and F_1-score. This way, we get an overall measure to compare different methods for spelling variant detection on the data.

When looking at applications for spelling variant detection in Chapter 8, we look at POS tagging and lemmatization. These tasks differ from spelling variant detection regarding that for each token exactly one label (POS tag or lemma) has to be found while in spelling variant detection for each token an unknown number (including 0) of spelling variants have to be identified. We use *accuracy*, i. e., the proportion of tokens where the predicted label is correct, to compare different approaches of handling spelling variation for POS tagging and lemmatization.

The performance of an approach is measured on data that is different from the training data and the potentially used background corpus. It is a common practice to use—in addition to the training data—two separate parts of the datasets for comparing and evaluating the different approaches: the development set (sometimes also called *validation* set) and the test set (Witten et al. 2017, Section 5.1). Different settings for the approaches are compared on the development set in order to select the best settings for an approach. The performance of an approach that is an estimate for the performance on similar but unseen data is then measured on the test set. In Chapter 2 we have described how the ReM and the ReN are divided into training development and test sets for the different experiments.

For POS tagging and lemmatization, we use statistical testing in order to test the null hypothesis that an observed difference in the accuracy for two approaches is only due to chance and would not be observable on a similar but alternative test set. As the approaches are applied to the same data, a statistical test that works with

paired data is needed. We use McNemar's test (McNemar 1947) with continuity correction (Edwards 1948) to test the null hypothesis that the probability of the first approach predicting the correct label in cases where the second approach does not predict the correct label is the same as the probability of the second approach predicting the correct label while the first approach does not. If one of the two approaches is indeed better than the other, we would expect this hypothesis to be false. Consequently, McNemar's test should result in a low p value and therefore allow us to reject the null hypothesis. The test statistic is:

$$\chi_1^2 = \frac{(|a_1 - a_2| - 1)^2}{a_1 + a_2}$$

a_1 is the number of tokens for which only the first approach predicts the correct tag. a_2 is the number of tokens for which only the second approach predicts the correct tag. Under the null hypothesis, this statistic follows a Chi-square distribution with one degree of freedom. We use this distribution to calculate the p-value and reject the null hypothesis if $p < 0.05$.

4.3. Binary Classification

For spelling variant detection, we apply a binary classification approach: for a given pair of two word forms, we train an ML algorithm to decide whether the given pair is a pair of spelling variants or not. The classification either simply uses two types as input for type-based spelling variant detection or one type with its context and the potential spelling variant for token-based spelling variant detection. As algorithms, we apply a Support Vector Machine (SVM) for the type-based and a Convolutional Neural Network (CNN) for the token-based classification. We describe SVMs in Section 4.3.1. The CNN uses a logistic regression layer for the binary classification. We describe logistic regression in Section 4.3.2. The convolutional layers, which give the CNN its name, are described in Section 4.8.3.

A naive approach for using binary classification in spelling variant detection would be to apply these algorithms to every possible pair of types in the dictionary. However, for a dictionary with size n this would lead to assessing $\binom{n}{2} = \frac{n*(n-1)}{2}$ pairs, which leads to a quadratic runtime regarding the number of types in the dictionary. To improve the runtime, we use a generator to get probable spelling variants for a given type. Such a generator should have a high recall. While the precision does not have to be very high, it should be high enough to substantially reduce the number of pairs that have to be evaluated compared to simply evaluating every possible pair. The general pipeline approach for spelling variant detection using a generator and filters is described in Chapter 7.

To get an idea of the binary classification task, we use the following pairs of types from GML:

(7) Pairs of spelling variants of GML *unde* ('and')
   a. un (22), vnnd (350)
   b. und (498), vn (229)
   c. vnnd (350), vnnde (177)

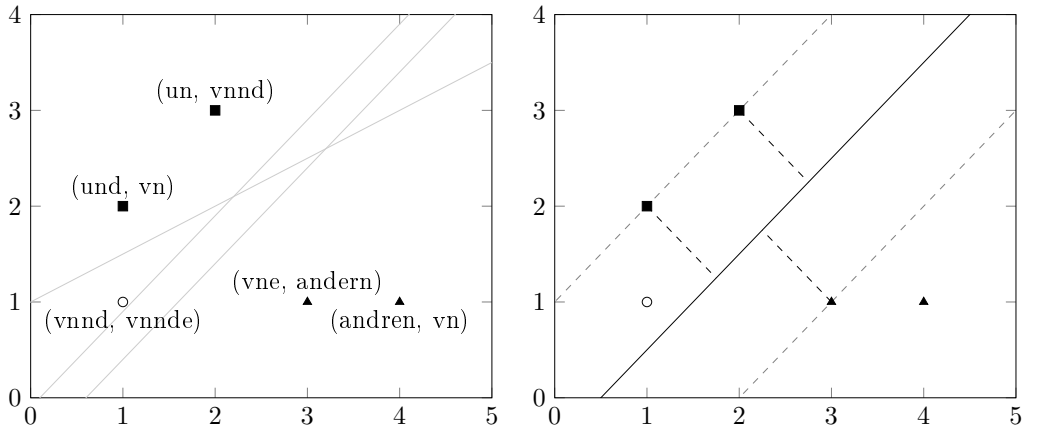
(8) Pairs of a spelling variant of *unde* ('and') and a spelling variant of *ander* ('second/other')
   a. andren (36), vn (229)
   b. vne (6), andern (189)

The examples are taken from the ReN 1.0 (cf. Section 2.2.1). The numbers in the parentheses are the frequencies for these types appearing written exactly as this in the corpus respecting capitalization. While the pairs in Example (7) are spelling variants, the pairs in Example (8) are not.[18]

For our binary classification example, we use two numeric properties to represent the pairs of types for the algorithm in order to decide if the given pair is a pair of spelling variants or not. We use the difference in length of the types and the difference in frequency, leading to a two-dimensional numeric representation for each pair. Regarding frequency differences, we put these into classes by calculating $\lfloor \frac{f}{100} \rfloor$ from the difference in the frequencies f. As it should not matter in which order the pair is represented, we use the absolute values of the differences. Please note that these two features of the word pairs serve only for a simple presentation of the used ML approaches. In our experiments, we use the features presented in Section 4.8. Figure 4.1a shows the data points for the five pairs from Example (7) and (8) in the two-dimensional space. The frequency difference is shown on the y-axis, the length difference on the x-axis. The squares represent known spelling variant pairs, the triangles represent pairs that are known not to be spelling variants. This is the training data for the binary classification. Additionally, the figure shows a potential new data point—the circle—that should be labeled as either positive or negative.

The basic approach in binary classification is to find a separation between the points (or vectors) that represent spelling variants and that do not represent spelling variants in the training data. Once learned on this data, this separation can be used to decide for a new pair of types whether they are spelling variants. In our two-dimensional example, this separation, the so-called decision boundary, can be realized by a straight line—called a hyperplane when generalized to an arbitrary number

18. The pairs have been chosen simply for illustration and are not complete. The corpus contains other spelling variants of *unde*. In the pipeline for spelling variant detection presented in Chapter 7, the word-pairs to which the binary classification is applied are selected by surface similarity.



(a) Examples for different decision boundaries (b) Maximum margin hyperplane as decision boundary

Figure 4.1.: Different decision boundaries for a binary classification problem in a 2-dimensional space.
The classes are represented with squares and triangles. The circle represents a data point that needs to be labeled by the algorithm.

of dimensions. The three lines in Figure 4.1a show potential decision boundaries for the given training data. While all shown decision boundaries perfectly separate the training data, they differ with respect to the additional data point. The difference between learning algorithms for binary classification can be conceptualized regarding how they find and model the decision boundary. In the following two sections, we describe how this is achieved with SVMs and with logistic regression.

4.3.1. Support Vector Machines

An SVM is an algorithm for binary classification that uses the maximum margin hyperplane as decision boundary. The maximum margin hyperplane is the hyperplane with the maximum distance to the closest data point. This decision boundary is shown in Figure 4.1b for our example. The dashed lines perpendicular to the maximum margin hyperplane show the distance to the closest data points in the training set. This distance is maximized by the shown decision boundary.

Formally, a hyperplane is given by the following set of points:

$$\{x : f(x) = x^T\beta + \beta_0 = 0\} \tag{4.4}$$

A hyperplane separates a given space into two parts and $f(x)$ gives the signed distance (weighted by the length of β) between the hyperplane and the given point

x. Furthermore, the sign of $f(x)$ denotes on which side of the hyperplane x lies. If we denote the two classes in the binary classification problem with 1 and -1, each point can be assigned a respective class with respect to the hyperplane defined by $f(x) = 0$ by the decision function $G(x)$ (Hastie, Tibshirani, and Friedman 2009, p. 418):

$$G(x) = \text{sgn}(f(x)) \tag{4.5}$$

sgn denotes the sign function which returns either 1 or -1.

The absolute normalized distance of a hyperplane to any point in the training data can be obtained by multiplying $f(x)$ with the class label t (to remove the sign) and dividing the result by the length of the parameters of $f(x)$, i. e., $\|\beta\|$. The margin of a hyperplane with respect to the training data is defined as the smallest distance between the hyperplane and any point in the training data. Now, the maximum margin hyperplane defined by $f_{max}(x)$ can be obtained by finding β and β_0 such that the margin is maximized:

$$\arg\max_{\beta,\beta_0}\{\frac{1}{\|\beta\|}\min_n[t_n f(x_n)]\} \tag{4.6}$$

n iterates over the training set. Finding β and β_0 that maximize Equation (4.6) can be shown to be equivalent with the optimization problem

$$\arg\min_{\beta,\beta_0}\frac{1}{2}\|\beta\|^2 \tag{4.7}$$

given the constraint that each data point from the training set has a distance of at least 1 to the hyperplane:

$$t_n f(x_n) \geq 1 \tag{4.8}$$

See Bishop (2006, pp. 327) for more details.

As can be seen from Figure 4.1b, not all vectors are relevant regarding the maximum margin hyperplane. Only the vectors that are closest to the hyperplane are relevant. If, for example, the data point $(4, 1)$ in Figure 4.1b changed to $(5, 1)$ it would not affect the decision boundary of the SVM. The relevant vectors—$(1, 2)$, $(2, 3)$ and $(3, 1)$ in Figure 4.1b—are called support vectors and give the SVM its name. These are the vectors x_n for which the equality $t_n f(x_n) = 1$ holds in Equation (4.8).

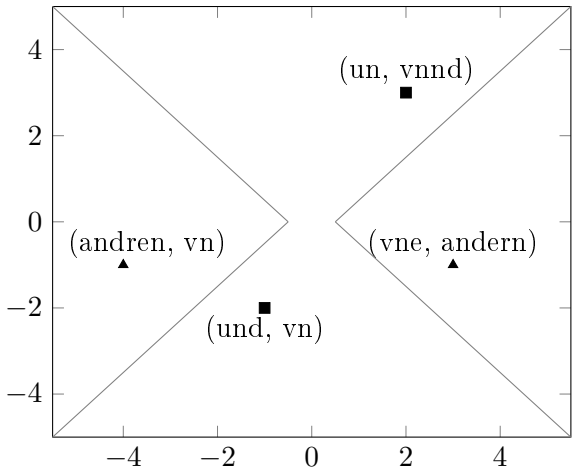


Figure 4.2.: Non-linear decision boundary.
The decision boundary is the decision boundary from Figure 4.1b projected into the space of the examples where the sign of the differences is kept.

As the data points in the training data are often not linearly separable, two extensions of this basic idea are usually applied with SVMs. The first of these two extensions is the usage of a function that maps the original data points into a different, maybe higher dimensional, space that we refer to as feature space. The linear separation learned for the feature space can be non-linear in the original space, which allows using an SVM for non-linear separation problems. One example for such a mapping into a feature space is using the absolute values for the elements of the data points as described above. While this was justified with the commutativity of the pairs, this is also a way to obtain a non-linear decision boundary in the original space where the sign of the numbers is kept as shown in Figure 4.2. In this space, the two classes are not linearly separable. However, they are separated by the non-linear decision boundary obtained by projecting the linear decision boundary back from the feature space into the original space.

So instead of calculating the maximum margin hyperplane in the original space, it is calculated in the feature space. For this, Equation (4.4) has to be changed to the following equation:

$$\{x : f(x) = h(x)^T\beta + \beta_0 = 0\} \tag{4.9}$$

$h(x)$ is a function that maps the original data points to the feature space.

As an alternative to the decision function from Equation (4.5) its dual representation can be used (Boser, Guyon, and Vapnik 1992):

$$G(x) = \operatorname{sgn}(\sum_{i=1}^{m} \alpha_i t_i h(x)^T h(x_i) + b) \tag{4.10}$$

The x_i are the support vectors of the maximum margin hyperplane. This dual representation allows substituting the usage of the feature map $h(x)$ with a suitable kernel function (Schölkopf and Smola 2001, pp. 201–204). A kernel function $k(x, x')$ is a function that returns the dot product for two vectors with respect to some function h, i. e.,

$$k(x, x') = h(x)^T h(x') \tag{4.11}$$

Therefore, in order to use an SVM, the feature map does not need to be given explicitly but can be defined by using a kernel. In our experiments we use a (Gaussian) Radial Basis Function (RBF) kernel, which has the form

$$k(x, x') = \frac{1}{\exp(\gamma \|x - x'\|^2)} \tag{4.12}$$

$\gamma > 0$ is not a parameter that is learned together with the other parameters β and β_0. It is a so-called hyperparameter that has to be chosen before calculating the hyperplane. As can be seen from Equation (4.12), the RBF kernel is a decaying function of the distance between two points in the original space. γ influences how fast this function decays, i. e., the width of the bell shaped (Gaussian) curve. With higher values of γ, $k(x, x')$ will decrease faster for increasing distances between x and x'.

The second approach to deal with data that is not linearly separable is the soft-margin SVM (Cortes and Vapnik 1995) that we use in the experiments of this thesis. So far we have assumed that a maximum margin hyperplane exists. But as we have seen above this is not always the case, cf. Figure 4.2. And even when using a feature map it is not guaranteed that the data points are linearly separable in the feature space. In these cases, it is not possible to fulfill the constraints given in Equation (4.8). This is due to the fact that for data points on the wrong side of the hyperplane $t_n f(h(x_n))$ will result in a negative value. In order to allow for misclassified data points in the training set, the soft-margin SVM algorithm introduces variables $\xi_n \geq 0$ for each data point in the training set. With these slack variables the constraints from Equation (4.8) are relaxed:

$$t_n f(h(x_n)) \geq 1 - \xi_n \tag{4.13}$$

Now it is possible to fulfill the constraint for data points on the wrong side of the hyperplane by choosing a respective value for ξ_n. In order to find a balance between a large margin and allowing for misclassifications with high values of ξ_n, the ξ_n are included into the optimization as well:

$$\arg\min_{\beta,\beta_0} \frac{1}{2}\|\beta\|^2 + C\sum_n \xi_n \tag{4.14}$$

The constant C can be interpreted as a cost parameter. With a high value for C, classification errors lead to a higher value of Equation (4.14) and should be avoided to find the optimal solution. When C approaches ∞, all ξ_n need to be zero for the optimal solution, leading to the strict maximum margin hyperplane. Looking at the linearly separable case, the interaction between the two terms and the role of C in Equation (4.14) can be illustrated as follows: Since the data is linearly separable there is a maximum margin hyperplane and the ξ_n are not needed, i. e., the constraints in Equation (4.13) can be fulfilled with all $\xi_n = 0$. For a solution where at least one $\xi_n > 0$ to lead to a lower value in Equation (4.14), $\|\beta\|^2$ needs to be smaller in order for the sum to be smaller. This is equivalent to a larger margin which now includes the data points for which $\xi_n > 0$. So the soft-margin constraints allow for ignoring some data points with respect to the margin of the chosen hyperplane. The optimal solution depends on C: For small values of C, the margin of the hyperplane gets larger with possibly many data points lying inside the margin or even on the wrong side of the hyperplane. Increasing values of C lead to smaller margins with less data points inside.

Instead of solving Equation (4.14) directly, the equivalent dual problem in Equation (4.15) can be maximized in order to find a solution using a kernel function as described above (Hastie, Tibshirani, and Friedman 2009, pp. 420–421):

$$\arg\max_{\alpha_n} \sum_n \alpha_n - \frac{1}{2}\sum_n\sum_m \alpha_m\alpha_n t_m t_n k(x_m, x_n) \tag{4.15}$$

with the constraints

$$0 \leq \alpha_n \leq C \tag{4.16}$$

$$\sum_n \alpha_n t_n = 0 \tag{4.17}$$

When employing a (soft-margin) SVM for classification, the user needs to choose the cost parameter C and a feature map $h(x)$ before training the SVM. Since we use an RBF kernel instead of $h(x)$ the hyperparameter γ needs to be chosen as well.

An alternative to an SVM for binary classification that is often found in conjunction with neural networks is logistic regression. We describe logistic regression in the following section.

4.3.2. Logistic Regression and Neural Networks

Logistic regression uses a linear model to find the decision boundary. A generalized linear model is given by

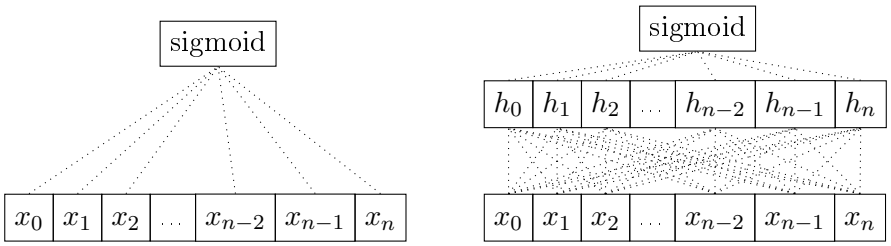
$$p(x) = g(w^T h(x) + w_b) \tag{4.18}$$

$g(x)$ is a nonlinear function called activation function in the context of neural networks and h is a feature map as described for SVMs. w is a vector containing the parameters that are learned during training. The parameter w_b is called bias. Notationally, the bias can be omitted by using the convention that $x_0 = 1$ and $w_0 = w_b$ (Bishop 2006, p. 229).

In the case of binary classification, the basic generalized linear model is called logistic regression. A logistic regression function models the probability of the data points to belong to one of the two classes. In our case, this is the probability of a data point representing spelling variants. When classifying an unseen pair of types, the model says that it is a spelling variant pair if this probability is at least 0.5. In order to model such probabilities, for logistic regression, the sigmoid function σ is used as activation function:

$$\sigma(a) = \frac{1}{1 + exp(-a)} \tag{4.19}$$

The sigmoid function results in a value between 0 and 1. When looking at points for which the probability is exactly 0.5, one can show that they form a hyperplane (or a line in the two-dimensional case), a linear decision boundary as presented above for SVMs. As with the SVM, it is desirable to introduce a mapping into a feature space for cases where the original data points are not linearly separable. In contrast to using a fixed feature mapping as in the case of SVMs where a kernel with a fixed-parametrization is chosen before searching the maximum margin hyperplane for the training data, another way to do this is simply to use another linear model with a non-linear activation as a feature map. The parameters of this linear model are learned when training the whole model.



(a) Logistic regression

(b) Feed-forward network with one hidden layer

Figure 4.3.: Neural Network diagrams

This is the basic idea behind (feed-forward) neural networks: stacking linear models (with non-linear activation functions) as for example the logistic regression model onto each other (Bishop 2006, p. 227). Mathematically a neural network is simply a "nonlinear statistical model" (Hastie, Tibshirani, and Friedman 2009, p. 392). Neural networks are often depicted in a diagram to give an overview of the function. Figure 4.3a shows a visualization of the simple logistic regression model with $h(x) = id(x)$. The dotted lines represent the multiplication with the respective parameter. In theory, such functions can be stacked arbitrarily deep, hence the term *deep learning* that is often found in conjunction with neural networks (Goodfellow, Bengio, and Courville 2016, pp. 164-165).

Above we have assumed that w is a vector. In general, w can be a matrix, leading to a vector as the result of the linear model. A neural network with $h(x) = a(w_h x)$ where w_h is a quadratic matrix with the same length as x is shown in Figure 4.3b. Layers like $h(x)$ that are neither the input nor the output of the network are called hidden layers. In the diagrams, the activation function a is omitted. While we use sigmoid as activation function for the output layer, we apply a rectified linear function r or Rectified Linear Unit (ReLU) activation for hidden layers, which is given in the following equation:

$$r(a) = \max(0, a) \tag{4.20}$$

In Section 4.8.3 we describe using convolutions for mapping the data points to a feature space for token-based spelling variant detection.

4.4. Imbalanced Training Data

When training a filter for spelling variant detection, the number of positive examples will be substantially smaller than the number of negative examples. This is due to the fact that most types only have a few spelling variants but at the same time many similar types that are not spelling variants, e. g., inflectional variants. This leads to the class imbalance problem:

> In these cases, standard classifier learning algorithms have a bias toward the classes with greater number of instances, since rules that correctly predict those instances are positively weighted in favor of the accuracy metric, whereas specific rules that predict examples from the minority class are usually ignored (treating them as noise), because more general rules are preferred. In such a way, minority class instances are more often misclassified than those from the other classes.
>
> (Galar et al. 2012, p. 465)

As the instances of the minority class are the spelling variants, that would mean that the algorithm would often discard spelling variants. In order to tackle class imbalance, different approaches have been suggested. We apply two of these in our experiments: random undersampling and bagging.

Random undersampling can be used to create balanced training data by taking a random sample that contains an equal amount of positive and negative data points. While this is a simple and effective approach, the learner does not use the whole training data but only a subset. Therefore, information from the removed data points is lost.

Bagging is not only an approach for imbalanced training data but it is a meta-algorithm that can be used with different learning approaches in order to improve their performance. Bagging exploits the fact that multiple decision boundaries exist as shown in Figure 4.1a on page 43 43. In bagging, multiple decision boundaries are combined with equal vote. The category with the most votes is chosen. A classifier combining the decision boundaries shown in Figure 4.1a would group the new data point, the circle, with the squares following two of the three decision boundaries. One way to apply bagging is by training learners on random samples of the training dataset, omitting and duplicating some instances. Due to variation in the training data, different learners lead to different decision boundaries (Witten et al. 2017, Section 12.2).

However, bagging can also be used to handle the imbalanced data problem (Galar et al. 2012). In this thesis, we apply UnderBagging (Galar et al. 2012, p. 472), where each sampled training set contains all the positive data points and equally many randomly sampled negative data points. Using bagging with SVMs has been

proposed for PU learning by Mordelet and Vert (2014). So we use bagging with SVMs to obtain a better decision boundary given the imbalanced and noisy training set.

4.5. Multiclass Labeling

ML approaches for binary classification as described in Section 4.3 can be easily adapted to the multiclass problem. Simple adaptations are the one-versus-the-rest or one-vs-one approaches (Bishop 2006, p. Section 7.1.3). An example for a POS tagger using the one-versus-the-rest approach with SVMs is SVMTool (Giménez and Màrquez 2004). In this approach, a separate SVM is trained for each POS tag. These SVMs learn to discriminate between the respective POS tag and all other tags. So for a given word, each of the trained SVMs decides whether the word should be labeled with the respective tag or any of the other tags (one-vs-the-rest).

However, there are also learning approaches that directly allow multiple classes. One of these approaches is a special class of log-linear models which is called multiclass logistic regression or softmax regression. We use such a log-linear model for lemmatization. Multiclass logistic regression is also the basis for sequence labeling using Conditional Random Fields (CRFs) that we introduce in Section 4.7.2.

Multiclass logistic regression models use the basic generalized linear model for binary classification from Equation (4.18) on page 48. But instead of using the sigmoid function from Equation (4.19) as activation function g, for multiclass classification the softmax function is used (Goldberg 2017, pp. 24):

$$softmax(a)_c = \frac{exp(a_c)}{\sum_j exp(a_j)} \tag{4.21}$$

$c \in C$ indicate one of the possible classes, a is a vector containing one number for each class. j iterates over the classes. By combining Equation (4.18) with Equation (4.21), one gets a generalized linear model for multiclass classification:

$$p(x)_c = softmax(w^T h(x) + w_b)_c = \frac{exp(w_c^T h(x) + w_{b_c})}{\sum_{j \in C} exp(w_j^T h(x) + w_{b_j})} \tag{4.22}$$

The model returns a value for each of the labels c. Due to the normalizing denominator in the softmax function, the values are all positive numbers and sum to 1. Therefore, $p(x)_c$ can be seen as a probability distribution over the different classes c given the observed value x. Note that w and w_b are matrices which contain the different parameter vectors w_c and w_{b_c} for each of the labels c. In order to avoid having multiple parameter vectors w_c, an alternative notation can be used

using feature functions f_k and one parameter vector w (Sutton and McCallum 2012, pp. 279):

$$p(x)_c = softmax(\sum_k w_k f_k(c,x))_c = \frac{exp(\sum_k w_k f_k(c,x))}{\sum_j exp(\sum_k w_k f_k(j,x))} \tag{4.23}$$

$f_k(c,x) \in \{f_{c',j}(c,x) = 1_{\{c'=c\}} h(x)_j\} \cup \{f_{c'}(c,x) = 1_{\{c'=c\}}\}$ and w_k is the respective parameter from w_c or bias parameter from w_{b_c} in Equation (4.22).

4.6. Learning Labels for Lemmatization

Lemmatization is the task of labeling a sequence of tokens with the corresponding lemmas or base forms as shown in Example (2) on page 3. As pointed out above, approaching this task as a labeling task is not trivial since the number of labels is large, even potentially infinite. A large number of labels has consequences for training and applying statistical models. For example, when estimating probabilities for the lemmas from an annotated corpus, there will be many lemmas with only a few appearances in the corpus and even many lemmas that do not appear in the corpus at all. Furthermore, the excessive number of labels will make it impractically slow to find the best label when applying the trained model.

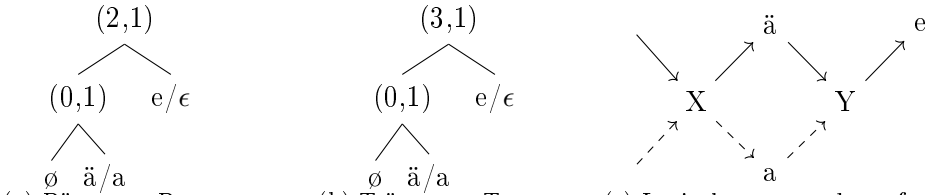
To solve this issue and apply (sequence-)labeling to lemmatization, Chrupała (2006) introduces the idea to use transformation rules as labels. The rules that are used as labels can then be used to transform a token into the corresponding lemma. As the rules generalize over multiple token-lemma pairs, the number of labels are reduced. Chrupała (2006) uses the shortest edit script between the reversed type and the reversed lemma for this, which

> is a set of *instructions* (insertions and deletions) which, when applied to sequence a, transform it into sequence b. An instruction specifies whether an insertion or a deletion should be performed, at which position in sequence a, and which element is to be inserted or deleted.
>
> (Chrupała 2006, p. 122).

Type and lemma are reversed in order for the edit scripts to better generalize over suffixes. The shortest edit scripts are automatically obtained from the training data.

This approach has been improved in Chrupała (2008) by using an Edit Tree (ET) instead of the shortest edit script:

> The idea is to find the longest common substring (LCS) between the form w and the lemma w'. We know that the portions of the string in



(a) Bäume → Baum "trees - (the) tree"

(b) Träume → Traum "dreams - (the) dream"

(c) Lexical correspondence for Träume → Traum, Bäume → Baum

Figure 4.4.: Edit Trees and Lexical Correspondence for the a-umlaut + -e inflection pattern from Barteld, Schröder, and Zinsmeister (2016)

> the lemma before (prefix) and after (suffix) the LCS need to be modified in some way, while the LCS (stem) stays the same. If there is no LCS, then we simply record that we need to replace w with w'.
>
> (p. 130)

To obtain the full ET, the procedure of finding the LCS is applied recursively to the found prefix and suffix. This kind of transformation rule encodes changes in the beginning and the end of the word form in the same way irrespective of its length.

While Chrupała, Dinu, and van Genabith (2008) use these transformation rules as labels, Müller et al. (2015) use the lemmas directly as labels but include a support function $h_w(l)$ into a log-linear model $p(l|w,m) \propto h_w(l) \times \exp(f(l,w,m)^T\theta)$ which is 1 for possible lemmas and 0 for all the other lemmas. h_w is 1 for all lemmas that can be obtained by applying a fixed set of ETs to the word form w. The set of ETs is extracted from the training set.

In Barteld, Schröder, and Zinsmeister (2016), we have introduced the usage of *Lexical Correspondences (LCs)* as an alternative to ETs in order to better generalize over word-internal modification as it appears in the morphology of languages like German and Dutch (Jongejan and Dalianis 2009). One example is the umlaut in *B**a**um* ('tree') – *B**ä**ume* ('trees') and in *Tr**a**um* ('dream') – *Tr**ä**me* ('dreams'). The corresponding ETs are shown in Figure 4.4a and Figure 4.4b. The numbers at the nodes of the ETs denote the start and the end of the LCS, measured from the beginning and the end of the type respectively. Since *um*—the LCS of both pairs *Bäume* – *Baum* and *Träume* – *Traum*—starts at a different position, the corresponding ETs differ.

LCs have been introduced formally by Fulop and Neuvel (2013) and have been used in morphological learning (Neuvel and Fulop 2002). As they do not encode the position of the common substrings, they do not differ for the example *Bäume* – *Baum* and *Träume* – *Traum* as can be seen in Figure (4.4c). Hence, when the training data only contains examples like *Bäume*, the usage of LCs allows to predict

the correct lemma for *Träume*. This is not possible with ETs. However, LCs are also ambiguous in certain cases, which leads to the creation of many lemma candidates. Therefore, we have tested variants of lexical correspondences for the support function in Lemming. In this thesis, we use LCs with anchored insertions and refer to them simply as LC. See Barteld, Schröder, and Zinsmeister (2016) for a more detailed discussion.

In Barteld, Schröder, and Zinsmeister (2016) we have used LCs as a means to deal with morphological word-internal modification. However, spelling variation also leads to word-internal differences. Hence, LCs can also be seen as a way to deal with word-internal modifications that are due to spelling variation. So in Section 8.4, we test using LCs instead of ETs for adapting Lemming to historical texts with spelling variation.

4.7. Sequence Tagging

When predicting POS tags for a sentence, the choices for individual labels influence choices for other surrounding labels. When applying a multiclass classifier to each token in a sequence individually, this dependency is lost.

In SVMTool (Giménez and Màrquez 2004), which is the POS tagger mentioned in Section 4.5, SVM classifiers are applied to each of the tokens in a sentence to find the best POS tag. To include the dependency between the tokens, the features for an individual target token include tokens and their POS tags before and after the target token. While the tags for the left context are the tags predicted by the tagger, the tags for the right context are not known when tagging the tokens sequentially. Therefore, in SVMTool ambiguity tags are used, i. e., concatenations of all possible tags for the given type that are found in the training data (Giménez and Marquez 2012, Section 3.3.2). This way, the SVMs can learn dependencies between the tags, e. g., if the previous tag is a determiner the current tag is not likely to be a preposition but might be a noun or an adjective. This is a simple approach to include the sequence information. However, many tools for POS tagging use approaches that model the dependencies between the tags in the sequence more explicitly. The tools that we use for the POS tagging experiments each use either a Hidden Markov Model (HMM) or a CRF, which we introduce in the following sections.

4.7.1. Hidden Markov Model

A Hidden Markov Model (HMM) models the probabilities of tag sequences. To do this, two assumptions are used: first, the Markov assumption and second, stationarity (Manning and Schütze 1999, p. 345). The Markov assumption states that the probability of a tag only depends on the previous tag instead of all previous tags,

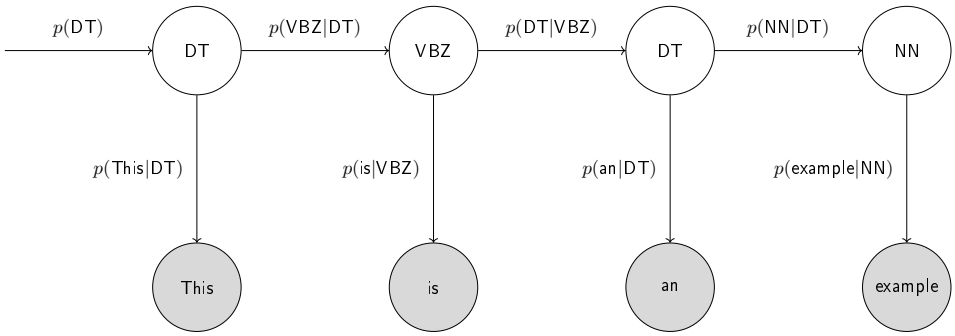


Figure 4.5.: Graphical representation of a Hidden Markov Model for the part-of-speech sequence of an example sentence. Observed states, i. e., the words, are shown in gray.

so $p(t_i|t_{1:i-1}) = p(t_i|t_{i-1})$. Stationarity means that $p(t_i|t_{i-1}) = p(t_j|t_{j-1})$ for any $i \neq j$ where $t_i = t_j$ and $t_{i-1} = t_{j-1}$. I. e., the probability of a given tag sequence does not depend on its position in the whole sequence, e. g., the probability that a noun follows a preposition is the same in the beginning, the middle and the end of a sentence.

Given these two simplifying assumptions, the probability of a tag sequence $t_{1:n}$ can be calculated as follows:

$$p(t_{1:n}) = p(t_1)p(t_2|t_1)p(t_3|t_2)\dots p(t_n|t_{n-1}) \tag{4.24}$$

Yet, this does only model the probability of tag sequences. But we want to model the probability of a tag sequence together with the corresponding sequence of types. In an HMM the tags are modeled as hidden states that emit types w_i with probabilities $p(w_i|t_i)$. Figure 4.5 shows a graphical representation of an HMM indicating the transition and emission probabilities for the sentence from Example (2) on page 3.

Given such an HMM, the probability that a sequence of tags $t_{1:n}$ emits an observed sequence of words $w_{1:n}$ can be calculated simply as the product of the emission probabilities:

$$p(w_{1:n}|t_{1:n}) = \prod_{i=1}^{n} p(w_i|t_i) \tag{4.25}$$

For training an HMM, the transition and emission probabilities have to be found. This is usually done with Maximum Likelihood Estimation (MLE) using an anno-

tated corpus. To account for rare and unseen words, smoothing can be applied (cf. Manning and Schütze 1999, pp. 346).

For tagging, we are looking for the optimal sequence of tags $t_{1:n}$ for a given sequence of words $w_{1:n}$, i. e., $\arg\max_{t_{1:n}} p(t_{1:n}|w_{1:n})$. While this cannot be calculated directly with an HMM, it is equivalent to finding $\arg\max_{t_{1:n}} p(w_{1:n}|t_{1:n})p(t_{1:n})$ (cf. Manning and Schütze 1999, p. 347), which can be calculated with Equation (4.24) and Equation (4.25). To find the optimal sequence, the Viterbi algorithm is usually applied. This algorithm uses a trellis to store the possible tag sequences and dynamic programming to find the optimal sequence (Manning and Schütze 1999, pp. 331–333).

A common variant of the simple HMM as described above is to take more context into account for the transitions. Instead of using the described bigram model that uses only the previous state ($p(t_i|t_{i-1})$), trigram models use the previous two states for the transition probabilities ($p(t_i|t_{i-2}, t_{i-1})$) and so on. Furthermore, the emission probabilities can also use more context, e. g., the surrounding tags ($p(w_i|t_{i-1}, t_i, t_{i+1})$).

One shortcoming with HMMs that is relevant in the context of spelling variation stems from the fact that they are so-called *generative models*, i. e., they model the sequence of observations, the tokens, as being generated (emitted) by the sequence of labels. Therefore, it is not straightforward to incorporate probabilities for unseen events (e. g., words that do not appear in the training data) into the model (cf. Manning and Schütze 1999, pp. 351–353). The HMM-based taggers that we use in the POS tagging experiments both use the tag distribution over word suffixes in the training data to estimate emission probabilities for OOV words.

An alternative to a *generative model* is to model the sequence of labels as being based on the observations, a so-called *discriminative model*. An example for a discriminative model is (multiclass) logistic regression. Since a discriminative model models the probability of the (sequence of) labels given an observation—or more precisely some features of the observation—, unseen events in the observation are easily handled by the model as long as the unseen events comprise known features. This seems favorable for texts with spelling variation. The discriminative counterpart to the generative HMMs is a subclass of CRFs which we describe in the next section.

4.7.2. Conditional Random Field

Conditional Random Fields (CRFs) are generalizations of (multiclass) logistic regression to sequences (Sutton and McCallum 2012). The multiclass logistic regression model from Equation (4.23) on page 52 can be easily adapted for sequence labeling by modeling $p(x_{1:n})_{c_{1:n}}$ as the product of the individual unnormalized probabilities $exp(\sum_k w_k f_k(c_t, x_t))$ for the different time steps t. The denominators have

to be adapted respectively to acquire values that sum to 1. While this is a way to model probabilities for label sequences $c_{1:n}$ that are conditioned on the observed token sequence $x_{1:n}$, there is no dependency between the labels as there is with the transition probabilities in an HMM. Such dependencies can be achieved by allowing more general feature functions that not only depend on one label c_t and one observation x_t but on multiple of these elements. A simple but often used example is the linear-chain CRF, where two neighboring labels c_t and c_{t-1} are used in the feature functions (Sutton and McCallum 2012, p. 288):

$$p(x_{1:n})_{c_{1:n}} = \frac{\prod_t exp(\sum_k w_k f_k(c_t, c_{t-1}, x_{1:n}, t))}{\sum_{c'_{1:n}} \prod_t exp(\sum_k w_k f_k(c'_t, c'_{t-1}, x_{1:n}, t))} \tag{4.26}$$

The feature functions $f_k(c_t, c_{t-1}, x_{1:n}, t)$ now depend on the labels at the time steps $t-1$ and t. Furthermore, we have added the whole input sequence $x_{1:n}$ and the time step t. This way, the feature functions can depend on observations for all time steps, e. g., features for the previous or the next word can be included.

A simple choice for feature functions that lead to a linear-chain CRF that resembles an HMM are two types of functions that serve similar purposes as the emission (Equation (4.27)) and transition (Equation (4.28)) probabilities in an HMM, see e. g., Silfverberg et al. (2014).

$$f_{c'_t,j}(c_t, c_{t-1}, x_{1:n}, t) = 1_{\{c'_t = c_t\}} h(x_{1:n}, t)_j \tag{4.27}$$

$$f_{c'_t,c'_{t-1}}(c_t, c_{t-1}, x_{1:n}, t) = 1_{\{c'_{t-1} = c_{t-1}\}} 1_{\{c'_t = c_t\}} \tag{4.28}$$

Using these feature functions, Equation (4.26) can be factorized as in the following equation:

$$p(x_{1:n})_{c_{1:n}} = \frac{\prod_t \Psi_t \Phi_t}{\sum_{c'_{1:n}} \prod_t \Psi_t \Phi_t} \tag{4.29}$$

with

$$\Psi_t = exp(\sum_{c'_t,j} w_{c'_t,j} f_{c'_t,j}(c_t, c_{t-1}, x_{1:n}, t)), \text{ and}$$

$$\Phi_t = exp(\sum_{c'_t,c'_{t-1}} w_{c'_t,c'_{t-1}} f_{c'_t,c'_{t-1}}(c_t, c_{t-1}, x_{1:n}, t))$$

This factorization with the definitions from Equation (4.27) and Equation (4.28) can be visualized as the factor graph (Sutton and McCallum 2012, p. 275) shown

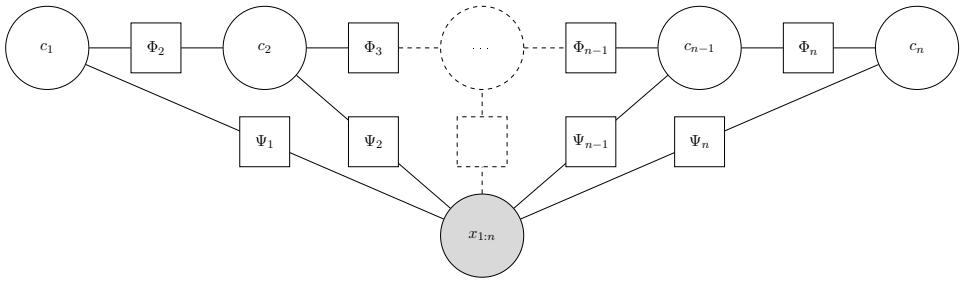


Figure 4.6.: Factor graph for a linear-chain Conditional Random Field. The graph corresponds to the CRF given in Equation (4.29)

in Figure 4.6. This graph depicts the labels $c_{1:n}$ and the tokens $x_{1:n}$ in circles, the factors are depicted in rectangles. The lines connect the factors with the labels and tokens that are included in it. Observed variables—in this case the sequence of tokens—are colored in gray. The graph does not show individual nodes for the tokens since the feature functions take the whole token sequence as input and extract the relevant features for the given time step. These features might include features from tokens before or after the given time step.

However, CRFs are not limited to this kind of connections. One variant is to use the observed $x_{1:n}$ in the transition features shown in Equation (4.28) as well. Another variant is to include more context, which leads to a higher-order CRF instead of a linear-chain CRF. Higher-order CRFs can be described by using the whole label sequence in the feature functions:

$$p(x_{1:n})_{c_{1:n}} = \frac{\prod_t exp(\sum_k w_k f_k(c_{1:n}, x_{1:n}, t))}{\sum_{c'_{1:n}} \prod_t exp(\sum_k w_k f_k(c'_{1:n}, x_{1:n}, t))} \tag{4.30}$$

Training a CRF is more complex than training an HMM. The goal is to find a set of parameters W that lead to a CRF that best describes the training data. Mathematically this can be modeled using the likelihood function:

$$l(W) = \sum_{(i)} p(x^{(i)}_{1:n})_{c^{(i)}_{1:n}}$$

$l(W)$ can be calculated on the training data $D = (x^{(i)}_{1:n}, c^{(i)}_{1:n})$. The aim is now to find $\arg\max_W l(W)$. This is equivalent to finding $\arg\min_W -log(l(W))$. This minimization can be solved using gradient-based methods. We describe such minimization methods for training CNNs in Section 4.8.3.

For tagging, we are looking again for the sequence of tags with the highest probability given the input sequence, i. e., $\arg\max_{c_{1:n}} p(c_{1:n}|x_{1:n})$. In contrast to an HMM, these probabilities can be directly computed with a CRF. For linear-chain CRFs this can be done efficiently using similar algorithms as for HMMs (Sutton and McCallum 2012, Section 4.1).

In general, though, both training and tagging CRFs can be computationally expensive since in both cases the probabilities for all possible tag sequences have to be computed. For instance, when minimizing the (log-)likelihood with a gradient-based method in order to find the best model for the training data, the gradient contains the marginal probabilities for the tag sequences, i. e., a sum over all possible tag sequences $c_{1:n}$.

> The training and also decoding times thus depend polynomially on the size of the tagset and exponentially on the order of the CRF. This probably explains why CRFs, despite their outstanding accuracy, normally only are applied to tasks with small tagsets such as Named Entity Recognition and Chunking; if they are applied to tasks with bigger tagsets – e.g., to part-of-speech (POS) tagging for English – then they generally are used as 1^{st}-order models.
>
> (Müller, Schmid, and Schütze 2013, p. 322).

In order to allow using higher-order CRFs for POS tagging, Müller, Schmid, and Schütze (2013) present pruned CRFs: during training and tagging probable tag sequences are determined using the CRF with a lower order, starting with a 0-order CRF. That is, for a given time step the tag probabilities are first calculated based only on the observed sequence (0-order CRF). Then a pruned first-order or linear-chain CRF is built by adding transition features but removing those tags from the possible sequences that have a low probability according to the 0-order CRF. Thus, during training and inference, fewer sequences have to be considered for the first-order CRF. This pruning strategy is repeated until the desired order is reached. We use such pruned CRFs in our POS tagging experiments.

4.8. Features for Describing Spelling Variants

In the previous sections, we have described ML algorithms that learn to separate data points that belong to different classes. For these algorithms, the data points need to be represented in a vector space. For words—or in our case pairs of words—the simplest way to represent them in a vector space is a one-hot representation, i. e., a vector in which each dimension represents a specific word. To represent a word, only one dimension is set to 1, all other dimensions are set to 0. This representation,

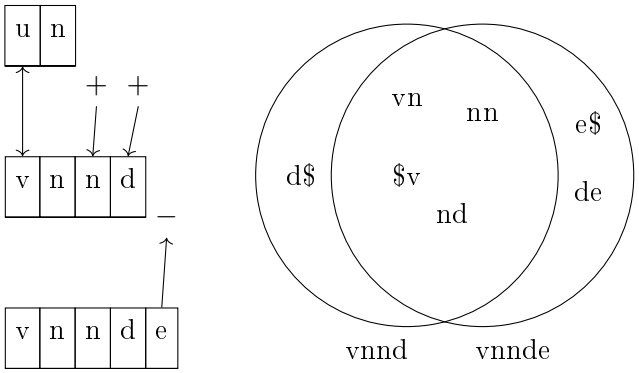
however, has a lot of drawbacks, one being that it does not allow an ML algorithm like an SVM to learn generalizations over different words or pairs of words as they are all represented differently and with equal distance to each other. Therefore, the data points are presented to the SVM with their values for predefined features. In this section, we present the features that we use for spelling variant detection. We also present distance or similarity measures that we use for generating spelling variant candidates in the pipeline.

As defined in Chapter 3, spelling variants are different surface realizations of the same morphological word. While not always the case, spelling variants often share similarities regarding this surface realization. Additionally—as spelling variants are basically the same morphological word—the contexts in which they appear are similar. Therefore, we use surface features and contextual features. In the following two sections, we present an overview of the surface and contextual features that are used in the experiments of this thesis. Furthermore, we also apply parameterized feature maps which map a representation of a data point into a different space (cf. Section 4.3.2). We apply this technique for the token-based spelling variant detection using a CNN, which we describe in Section 4.8.3.

4.8.1. Surface Similarities

Surface similarity between two word forms *a* and *b* can be operationalized by looking at how to transform type *a* into type *b*. The changes that are allowed to do this are called *edit operations*, e. g., deleting, adding or substituting a letter. These edit operations can be used to quantify the surface similarity between two words. For this, each *edit operation* is assigned a cost. By finding a list of edit operations to transform *a* into *b* and summing the costs associated with the edit operations, a numerical distance between the two word forms can be computed. The cost of the sequence of edit operations to transform *a* into *b* with the minimal cost is called *edit distance*. Given a pair of types it can be found efficiently using dynamic programming (Wagner and Fischer 1974).

There are several variants of edit distances that differ with respect to the allowed edit operations and their associated cost. One is the Levenshtein distance (Levenshtein 1966), which allows insertions, deletions and substitutions, each with a cost 1. We use it to generate spelling variant candidates for a given type by extracting all types from a dictionary that have a small edit distance to the type under consideration (cf. Section 7.1). Given the types *vnnde*, *vnnd* and *un*—all three variants of the GML morphological word *unde* ('and')—using this generator with a distance of 2 or 1 would include *vnnde* as spelling variant candidate for *vnnd* but not *un* as the pair *un* and *vnnde* has a Levenshtein distance of 3 (see Figure 4.7a on the facing page). To efficiently find all types from the dictionary with a given edit distance to the target type, we use Levenshtein automata as proposed by Schulz and Mihov (2002).



(a) Levenshtein distance

(b) Jaccard index

Figure 4.7.: Examples for the Levenshtein distance and the Jaccard Index

An alternative way of measuring the similarity of two word forms is by looking at the combinations of characters (character n-grams) that appear in the words. The similarity can then be calculated by using the Jaccard Index (Levandowsky and Winter 1971). Figure 4.7b shows an example for *vnnd* and *vnnde* using character 2-grams. The words are padded with *$* before extracting the n-grams. The Jaccard Index is $\frac{4}{7}$, the number of elements in the intersection of the sets (4) divided by the number of elements in the union of the sets (7). Instead of only using consecutive characters, gaps between the two characters can be allowed which leads to skip-grams (Guthrie et al. 2006).

We compare the Jaccard Index and other n-gram based similarity measures with edit-operation based measures in Section 7.1.1. When representing a pair of words for the SVM, we experiment with different representations based on character n-grams around mismatches between the words. This does not only take character n-grams into account but also their position within the words. The features are described in more detail in Section 7.2.2. For the token-based filter, we represent the spelling variant candidate as a sequence of one-hot encodings of the individual characters and use the CNN with an embedding layer to learn relevant surface features for the individual types (cf. Section 4.8.3).

4.8.2. Contextual Similarities

The context in which a word appears has often been used to create representations for words in NLP. Based on the distributional hypothesis, which states that words with similar meanings often occur in similar contexts (cf. Harris 1954), such representations can be seen to approximate the meaning of words.

One way to obtain context representations is to assign words to different clusters based on their context. Brown clustering (Brown et al. 1992) is one way to achieve such a clustering. It has been frequently used to generate features for NLP applications (Derczynski, Chester, and Bøgh 2015). Brown clustering is based on a class-based n-gram model, a specific type of language model. A statistical language model assigns a probability to a word w_i given its preceding words w_1^{i-1} (Manning and Schütze 1999, Chapter 6). N-gram models apply the Markov assumption (cf. Section 4.7.1) and reduce the history of a word to $n-1$ words, i. e., in an n-gram model $p(w_i|w_1^{i-1}) = p(w_i|w_{i-n-1}^{i-1})$. In the case of a 2-gram or bigram model, only the previous word is considered, $p(w_i|w_1^{i-1}) = p(w_i|w_{i-1})$. The probabilities of the n-grams can be estimated based on a training corpus. A class-based n-gram model now does not consider the actual sequence of words preceding w_i but the history of classes that the words are assigned to for a given partitioning, i. e., $p(w_i|w_{i-n-1}^{i-1}) = p(w_i|c_i)p(c_i|c_{i-n-1}^{i-1})$ (Brown et al. 1992, p. 471).

Brown et al. (1992) show that the probability that a class-based bigram model assigns to the training texts depends only on "the average mutual information of adjacent classes" (p. 472). By finding a clustering of words that optimizes this average mutual information, one finds a clustering that maximizes the probability of a class-based bigram model for the training data. While Brown et al. (1992) do not give a method for exactly finding such a clustering, they present a method to approximate it by starting with each word in a separate cluster and greedily merging clusters based on their average mutual information. The implementation of this method that we use for the experiments in this thesis is from Liang (2005).[19] We use Brown clusters to filter spelling variant candidates in Section 7.1.3.

Another way to use context to learn representations for words is to create vector representations such that words that appear in similar contexts are represented by similar vectors. Baroni, Dinu, and Kruszewski (2014) distinguish *count* and *predict* models for this. Count models are—as the name suggests—based on counting words that appear in the context of a target word. By putting these counts in a vector where each dimension represents a context word, a word representation can be obtained from a corpus. In practice, instead of raw counts other values are used, e. g., positive Pointwise Mutual Information (PPMI). Furthermore, the dimensionality of the resulting vectors is reduced by applying Singular Value Decomposition (SVD) (Manning and Schütze 1999, Chapter 15.4.2).

Predict models have been popularized by Mikolov et al. (2013). The authors present two models for learning word representations. In the Skip-Gram with Negative Sampling (SGNS) model, which we use in this thesis, a neural network is trained to predict context words of a given word. The pairs of target and contexts

19. The source code is available at `https://github.com/percyliang/brown-cluster`, last visited October 4, 2021.

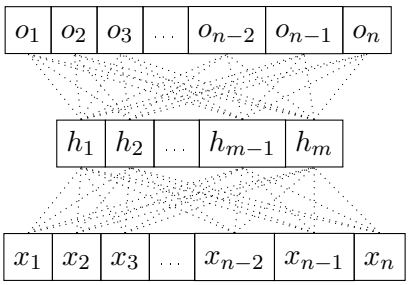


Figure 4.8.: Skip-gram model for word embedding

words are skip-bigrams (cf. Section 4.8.1), wich give the model its name. The neural network is shown in Figure 4.8. The input vector $x = (x_1, \ldots, x_n)$ is a one-hot vector encoding the target word. The output layer $o = (o_1, \ldots, o_n)$ is a softmax layer (cf. Section 4.5) over the vocabulary. By training this network, an embedding function is learned. This embedding function maps one-hot encoded words to dense vector representations h, which are then used and thereby optimized to predict context words. After training the network, only the embedding function is used to obtain the word representations. The size m of the hidden layer—the word embedding—is a hyperparameter.

In practice, training such a network with a huge number of labels—the whole vocabulary—is computationally expensive. This is why Mikolov, Yih, and Zweig (2013) present approximations for computing the full softmax layer. SGNS uses negative sampling. Negative sampling can be thought of as turning the multiclass classification into multiple binary classification tasks: Instead of learning a single probability distribution over the whole vocabulary with softmax, for each word of the vocabulary a logistic regression classifier is trained to decide whether this word appears in the context of a given input word or not. Now the classifiers are only trained with actual pairs of target and context words from the corpus and a fixed number of randomly sampled words for each target word, representing negative examples. So for each word in the vocabulary, not all classifiers are updated but only the classifiers for the actual context words and for the negative samples.

A shortcoming of PPMI-SVD and SGNS is that both methods only create word representations for words in the training data. An extension of SGNS that solves this shortcoming is fastText (Bojanowski et al. 2017). In the fastText model, the input is a one-hot representation of a word combined with a vector that encodes the character n-grams contained in the word. Each dimension represents a character n-gram. The dimensions representing the character n-grams contained in the target word are set to 1, all other dimensions to 0. This way, the model learns an embedding for the words in the vocabulary but also for the character n-grams. The vector h,

which is used as the word embedding, is the sum of the individual embeddings for the target word and the embeddings of its character n-grams. An embedding for an OOV word can be generated by simply summing the character n-gram embeddings. We do not make use of this feature of fastText, however, using character n-grams "provides very good word vectors even when using very small training datasets" (Bojanowski et al. 2017, p. 141), which is helpful in low-resource settings.

While prediction-based models have been declared to be superior to count-based models (cf. Baroni, Dinu, and Kruszewski 2014), Hamilton, Leskovec, and Jurafsky (2016) have found with diachronical data that "[b]etween [count-based PPMI-]SVD and [prediction-based] SGNS the results are somewhat equivocal, as both perform best on two out of the four tasks" (p. 1494) that they examine. In our pipeline for spelling variant detection, we use word representations obtained using PPMI-SVD. We use these representations as features for the type-based SVM (Section 7.2.2) and the token-based CNN filter (Section 7.2.3). For POS tagging, we compare PPMI-SVD, SGNS and fastText embeddings as word representations in Section 8.3.

Since context-based word representations are extracted from unlabeled texts, they can be computed on a large background corpus. Using these representations as features when training an ML algorithm on a smaller set of labeled training data is a simple example of semi-supervised learning (cf. Section 4.1.4). For the token-based filter we combine this semi-supervised learning with a function that learns task-specific representations on the training data using a CNN that we describe in the next section.

4.8.3. Convolutional Neural Network

For the token-based filter, we apply a CNN that filters out spelling variant candidates that do not fit in the given token context. For this, we apply a CNN similar to that of Kim (2014). A CNN is a specific type of neural network. We have described the basics of neural networks for binary classification in Section 4.3.2.

As noted above, the feature map h in Equation (4.18) on page 48 can consist of multiple composed functions. In our case, we use an embedding layer e and a convolution layer c, such that $h(x) = (c \circ e)(x)$. We describe both layers in more detail.

Conceptually, an embedding layer is simply a lookup that is often used as the first layer in a neural network when working with categorical data: for each of the input categories, an associated vector is chosen. Mathematically, an embedding layer can be described as $e(x) = w_e x$, where w_e is a matrix. The input vectors x are from $\{0,1\}^n$ with $\|x\| = 1$, the already described one-hot vectors. The dimension that contains the 1 denotes the category encoded in the vector. An embedding function is useful to encode similarities between distinct categories (Goldberg 2017, Chapter 8). In the case of spelling variation, we use embeddings for words and

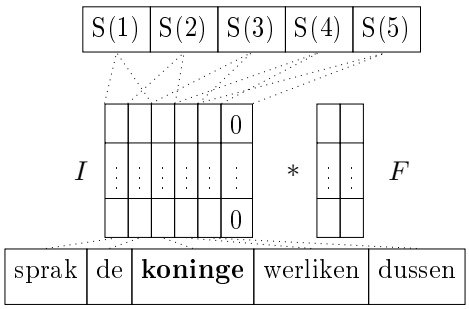
characters. This allows the network, for example, to learn to embed characters like u and v that often appear interchangeably in the data to a similar representation. The embedding function can be trained directly with the neural network. However, it is also possible and common to use an embedding function with parameters fixed before training. In NLP this is often done with word embeddings obtained from a background corpus using the distributional hypothesis as described in Section 4.8.2.

Since we are working with tokens, the input to the CNN is a potential spelling variant with its left and right context, which we present to the CNN as a sequence of one-hot encoded words that are mapped to a sequence of vectors using the described feature maps. The embedding approach that we use applies word embeddings obtained using the distributional hypothesis combined with a vector representation obtained from the sequence of characters using embeddings and convolutions as well. So the sequence of words is mapped to a sequence of vectors that represent the contexts in which a word appears and its surface form. Hence, e is a function composed of multiple functions in our case. This specific model is described in more detail in Section 7.2.3. As input to the convolutional function, we use word representations that are combined from a prelearned context-based representation as described in Section 4.8.2 and surface-based representations that are learned with the rest of the parameters on the training data.

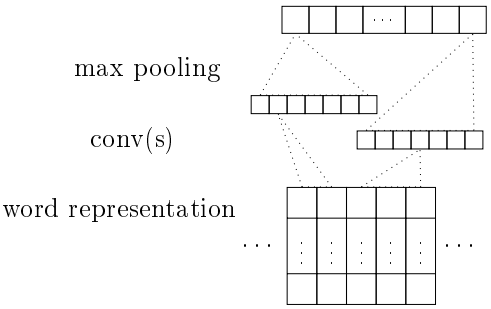
Such a representation, $e(x)$, for each word is then used as the input of c before the logistic regression is applied. c is a function that transforms the sequence of vectors to a single vector. In our case, we apply a convolution layer, hence the name CNN.[20]

The input to this convolution is a matrix, where each column is the dense vector representation of the corresponding element in the input sequence. A one-dimensional convolution operation, which we apply in this work, can be seen as sliding a matrix—the filter F—with the same number of rows m but a smaller number of columns n over the input matrix I. At each position, the overlaying matrix elements are multiplied and finally summed. The results are collected in a vector $S(i)$. This is shown in Figure 4.9a. The input is padded with column vectors containing zeroes. The number of padding vectors is $n - 1$. The vectors are evenly distributed to the left and to the right of I starting at the right. This way, the resulting vector will have exactly as many elements as the input sequence.

20. The convolution function described here—and that is implemented in the Keras/Tensorflow framework that we apply for our neural network experiments—is actually a cross-correlation and not a convolution in the strict sense. However, we follow the usage in neural network literature and use the term convolution (Goodfellow, Bengio, and Courville 2016, p. 329).



(a) Convolution between the input I and a filter F



(b) Multiple convolutions combined with max pooling

Figure 4.9.: Convolutional Neural Network diagrams

The $S(i)$ are given by the following equation, which shows the convolution and also includes a bias b and the activation function r (in our case ReLU):

$$S(i) = r(b_i + (F * I)(i)) = r(b_i + \sum_m \sum_n I(m, n + i - 1)F(m, n)) \tag{4.31}$$

i denotes the position of the sliding window, and $(F*I)$ is the convolution between the filter of the convolution operation F and the input I. The values of the filter $F(i, j)$ are parameters that are learned during training.

In a CNN, usually multiple convolutional filters with differing lengths are applied. This leads to multiple resulting vectors. One way to combine these vectors into one vector that we apply in this thesis is pooling, specifically max pooling (Goodfellow, Bengio, and Courville 2016, Chapter 9.3). With max pooling each vector resulting from the convolutions is summarized by its maximum. Conceptually, this can be understood as each filter scanning the input for the presence of a specific feature and reporting the result, i. e., the presence or absence of that feature. So after applying the convolutions with max pooling, we have a vector with one number for each convolutional filter. The whole convolutional layer with multiple filters and pooling is shown in Figure 4.9b.

In sum, the CNN that we use applies a logistic regression on a feature vector. This feature vector is not the input of the neural network but it is obtained by applying convolutions with max pooling to the input, which is obtained via an embedding function from the categorical inputs. Like the logistic regression function, the convolution is a parameterized function as well. The parameter settings are learned in combination with the parameters for the logistic regression.

Training the CNN means finding a parametrization that leads to a good distinction between negative and positive examples on the test set. For this, a loss function

is used, which quantifies the quality of the predictions on the dataset. We use cross entropy (Mitchell 1997, p. 170):

$$-\sum_i (d_i \ln(p_i) + (1 - d_i) \ln(1 - p_i)) \tag{4.32}$$

d_i is the label (1 for spelling variants and 0 for non spelling variants) and p_i is the predicted probability of the pair being a spelling variant, i. e., the output of the neural network. The summands for the individual data points are 0 if $p_i = d_i$ and will get higher, the more p_i deviates from d_i.

The aim is now to find a parametrization for the neural network such that the cross entropy (Equation (4.32)) is minimal. Because neural networks are complex non-linear functions, it is not always possible to directly find a minimum. Instead, neural networks are usually optimized by starting with a random parametrization and adapting this parametrization iteratively such that Equation (4.32) gets smaller (Bishop 2006, p. 237). To inform the updates of the parametrization the gradient of the loss function can be used to guide the update into the direction where the decrease is highest (Bishop 2006, pp. 239).[21] The basic form of a gradient-based iterative minimization is given in the following formula:

$$w_t = w_{t-1} - \mu \nabla L(w_{t-1}) \tag{4.33}$$

w_{t-1} are the parameters of the network before the update and w_t are the parameters after the update. L is the loss function, in our case cross entropy given in Equation (4.32), and μ is the learning rate, a hyperparameter that has to be defined by the user. Instead of calculating the gradient $\nabla L(w_{t-1})$ exactly for the whole training data, it is often approximated by taking only a randomly chosen subset, so-called mini batches, of the training data for calculating an update (Goodfellow, Bengio, and Courville 2016, Chapter 8.1.3).

For training the CNN, we use the gradient-based optimization method called Adam (Kingma and Ba 2015) with mini batches. Adam incorporates two ideas into basic mini-batch-based gradient descent. First, instead of using only the gradient calculated on the current mini batch for the update, Adam uses a decaying average of the previous gradients thereby avoiding oscillation of the updates. Secondly, instead of using the same learning rate for each parameter, Adam applies individual learning rates for each parameter. This is achieved by calculating the learning rate using a decaying average of the squared past gradients.

21. The function that we use is actually not fully differentiable. One reason for this is the ReLU activation, for which the gradient is not defined for an input of zero. In practice this is not problematic as the input to the ReLU will seldom be exactly zero and if, any of the subgradients can be used (Goodfellow, Bengio, and Courville 2016, 188).

The update rule of Adam is:

$$w_t = w_{t-1} - \frac{\mu}{\sqrt{\frac{v_t}{1-\beta_2^t}} + \epsilon} \frac{m_t}{1-\beta_1^t} \tag{4.34}$$

m_t, the decaying average of the gradients, is defined as:

$$m_t = \beta_1 m_{t-1} + (1-\beta_1)\nabla L(w_{t-1}) \tag{4.35}$$

v_t, which is used to calculate the actual learning rate, is defined as:

$$v_t = \beta_2 v_{t-1} + (1-\beta_2)(\nabla L(w_{t-1}))^2 \tag{4.36}$$

The decay rates β_1 and β_2 as well as ϵ are hyperparameters. ϵ can be seen as a smoothing constant to avoid divisions by zero. μ is the learning rate. We use the following settings: $\mu = 0.001$, $\beta_1 = 0.9$, $\beta_2 = 0.999$ and $\epsilon = 10^{-7}$ (see Kingma and Ba 2015, p. 2).

The divisions by $(1-\beta_1)$ and $(1-\beta_2)$ can be understood by looking at m_t and v_t as estimators of the mean and the uncentered variance of the gradients with respect to a stochastic loss function. In our case, the loss function is stochastic due to using mini batches. As m_t and v_t are initialized with zero, these estimators are biased. The divisions lead to unbiased estimators.

While gradient descent and similar methods find a good parametrization for the network, this parametrization is not guaranteed to be the optimal solution for the training set. It might also be a local minimum. Therefore, fitting a neural network to training data using a gradient-based method does not necessarily lead to the same decision boundaries for multiple iterations. However, neural networks with the parameterized feature map tend to overfit a dataset, i. e., find a solution that works well on the training set but does not generalize well to other datasets. Therefore, the parametrization that leads to minimal loss on the training set might not be a parametrization that generalizes best to other datasets. Consequently, when training neural networks measures are used to avoid overfitting. In this thesis, we apply a simple method by only training for a fixed number of epochs—an epoch is a pass over all training instances—instead of updating until convergence, i. e., until a (local or global) minimum is reached (Hastie, Tibshirani, and Friedman 2009, p. 398).

After having described the various ML methods that we apply in our experiments, in the next chapter, we present the evaluation settings that we use for spelling variant detection.

Chapter 5.

Intrinsic Evaluation of Spelling Variant Detection

In this chapter, we introduce two evaluation settings that we employ for comparing our approaches to simplification and spelling variant detection, which we present in Chapter 6 and Chapter 7. With these two evaluation settings, we perform an intrinsic evaluation, i. e., we measure the quality of the proposed techniques directly regarding spelling variation. In Chapter 8, we present extrinsic evaluations, i. e., we evaluate the pipeline for spelling variant detection regarding its utility for POS tagging and lemmatization (see Resnik and Lin 2010 for a description of intrinsic and extrinsic evaluation).

The two evaluation settings that we present in this chapter are inspired by two different perspectives that are relevant for Digital Humanities: From an IR perspective, spelling variation leads to a low recall when searching for documents containing a specific word unless the user knows the possible spelling variants. This makes searching in non-standard texts challenging. Spelling variant detection can facilitate it by generating spelling variants for the query term. From an NLP perspective, spelling variation leads to a situation in which training information that actually belongs to one and the same morphological word is distributed over its different spelling variants, e. g., *steit* and *steyt* in Example (3) on page 8. Furthermore, spelling variation leads to a high number of OOV words when applying a trained supervised ML model to unseen data. This causes NLP tools to perform much worse than they would on standard data. This effect can be mitigated by using the generated spelling variants instead of unknown words when applying an NLP tool. To estimate the utility of our approaches for both applications, we introduce and use two evaluation settings that capture relevant aspects of these tasks.

In the following sections, we describe the two evaluation settings and how we apply them in our experiments.

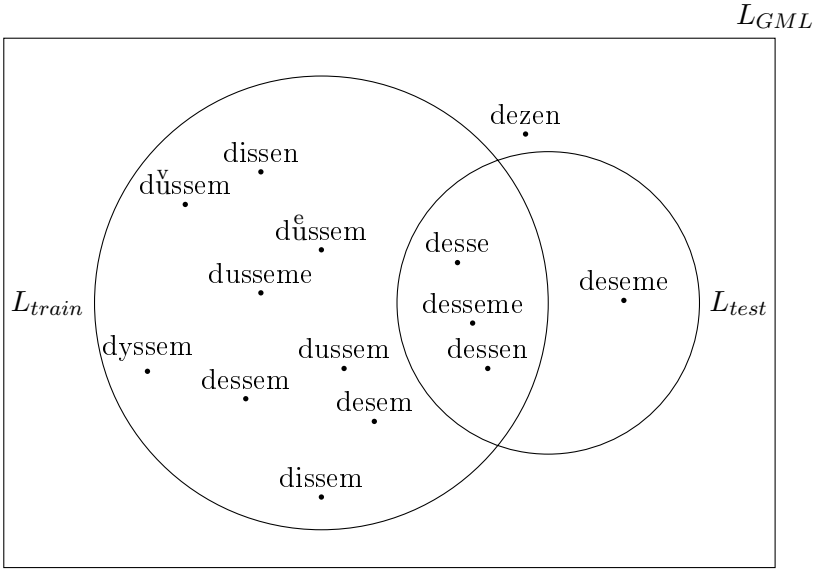


Figure 5.1.: Spelling variants of *desse, dit* in the Reference Corpus Middle Low German/Low Rhenish (1200–1650) (ReN 0.3)

5.1. Evaluation Settings

Before we discuss the two settings in more detail, we need to clarify the in- and output for our simplification and spelling variant detection approaches: Given a dictionary L_{T} that contains the types that appear in textual data T and some token t, i. e., a type that may or may not occur in T and its context, the aim is to produce a list of all types of L_{T} that are spelling variants for t (cf. the definitions in Chapter 3).

With both evaluation settings, we evaluate the pipeline on tokens from the test (respectively the development) set. Which dictionary is used for producing and evaluating the spelling variants for these tokens (L_{train} or L_{test}) depends on the evaluation setting.

We illustrate these two settings with the spelling variants of the GML masculine-neuter demonstrative pronoun *desse, dit* in the dative singular form. Figure 5.1 shows their distribution over the datasets. For our example, we use the training, development and test sets from the ReN as described in Section 2.3. From these sets, we extract three dictionaries L_{train}, L_{devel}, and L_{test}, each containing the types that appear in the respective set of the data. L_{GML}, the unknown complete dictionary of GML, is approximated by the complete dataset, $L_{train} \cup L_{develop} \cup L_{test}$. We illustrate the situation when applying the pipeline on the test set, therefore the development set is not marked explicitly in the figure.

Detection of spelling variants in texts (text-eval) With the first evaluation setting, we evaluate for each token in the test set the identification of its spelling variants in the very same test set, i. e., in L_{test}. It is motivated by an IR or search scenario for which we assume that the aim is to search in data that is distinct from the data that has been used to train the pipeline. In particular, for each occurrence of *desse*, *desseme*, *dessen*, and *deseme* in the test set, the pipeline has to determine the spelling variants that appear in this set, i. e., for each token a subset of *desse*, *desseme*, *dessen* and *deseme*.

Detection of known spelling variants for unknown types (OOV-eval) With the second evaluation setting, we evaluate the mapping of words from the test data that do not appear in the training data, i. e., OOV words, to spelling variants in the training data. It is motivated by the assumption that the training data is used to train the spelling variant detection pipeline as well as an NLP tool. Hence, when applying the NLP tool to unseen data, e. g., the test set, the aim is to produce spelling variants that are known to the tool (i. e., types from L_{train}) for unknown types in the test data. In our example this means generating the spelling variants for each token of *deseme* as a subset of *dissen*, *dů̈ssem*, *dů̈ssem*, *dusseme*, *dyssem*, *dussem*, *dessem*, *desem*, *dissem*, *desse*, *desseme*, and *dessen*.

In sum, spelling variants are always generated and evaluated with respect to a specific dictionary and not with regard of an (unknown) complete dictionary L_{GML}. Consequently, in our example, *dezen* is never generated because it does neither appear in the training data nor in the test data.

5.2. Effects of Data Sparsity for a Data-Based Definition of Morphological Word

A crucial requisite for evaluating which tokens should be considered as spelling variants is a proper definition of what a morphological word is. This is not a trivial task because a word can be defined in different ways (cf. for lemmatization Weißweiler and Fraser 2018). For our evaluation based on the GML texts in the ReN 0.3, we employ a definition that is based on the lexical and syntactic annotation of the corpus which we have described in Section 2.2.1: Two tokens with the same POS label, the same morphology and the same lemma[22] are considered to be instances of the same morphological word—and consequently, if the spelling differs, spelling variants (cf. Chapter 3).[23]

22. The lemmatization in the corpus includes word sense disambiguation such that homonyms are distinguished.

23. This definition leads to a broad definition of spelling variation, as types that are not spelling variants in a strict sense might be conflated due to the lemmatization. One example are the adverbs *vele* 'a lot' and *mehr* 'more' that are derived from the positive and the comparative form of the adjective *vele* and are therefore lemmatized the same.

Type	POS	Morph.	Freq.
pylatus	proper noun (M)	NOM.SG	13
pilatus	proper noun (M)	GEN.SG	1
gheystlyk	adjective	F.DAT/ACC	2
geystlich	adjective	N.NOM	1
frage	verb	1SG	3
vrage	verb	IMP.SG	2
vrage	common noun (F)	ACC.SG	1
vraghe	common noun (F)	NOM.SG	1
frage	common noun (F)	NOM.SG	1
frage	verb	3SG	1
vrage	verb	3SG	1
vrage	verb	2PL	1

Table 5.1.: Examples for spelling variants that do not appear with the same annotation in the Reference Corpus Middle Low German/Low Rhenish (1200–1650) (ReN 0.3)

This definition has a data sparsity issue when used for evaluation: Infrequent but ambiguous types might not appear as instances of every possible morphological word that they can represent. Therefore, some pairs of types that are seen as actual spelling variants by human experts are counted as false positives according to this evaluation scheme. This is illustrated by the examples *pilatus* ('Pontius Pilate'), *gheystlich* ('clerical') and *frage* ('(the/to) question') in Table 5.1. The column POS and Morph. show the categories with which the types appear in the corpus.[24]

All the types in one of the three sections of the table are considered spelling variants by experts. However, only *vraghe* and *frage*, printed in bold in the table, and *frage* and *vrage*, printed in italics in the table, are considered to be spelling variants by the data driven definition since they are the only two pairs of types that appear with the same annotation in the texts. So, in a type-based approach, *frage*, *vraghe* and *vrage* would be considered spelling variants, in a token-based approach *vraghe* and *frage* would be considered spelling variants when used as the nominative singular of the noun as well as *frage* and *vrage* when used as the 3rd person singular of the verb. Though, e. g., *pilatus* could appear in nominative case, due to data sparsity it only appears in genitive case and is therefore not considered as spelling variant of *pylatus* by our data-driven definition. This problem is obviously connected

24. For a description of the abbreviations see Footnote 1 on page 1.

Type	Morph	Freq.
koning	NOM.SG	51
common noun (M)	**ACC.SG**	22
	GEN.SG	6
	DAT.SG	5
koninge	**DAT.SG**	15
common noun (M)	NOM.PL	5
	ACC.PL	1
	ACC.SG	1

Table 5.2.: Ambiguity of *koning* and *koninge* in the Reference Corpus Middle Low German/Low Rhenish (1200–1650) (ReN 0.3)

to low-frequency types: If two low-frequency types are ambiguous, they are likely to not overlap with their grammatical function in the corpus. Therefore, the actual precision of the pipeline will be higher than measured on this dataset.

It is important to note that we use two different parts of the dataset for assessing which types are spelling variants. For training, we only use the training set to define spelling variants since this is the only data source that is available when training. For evaluating, we use the whole dataset in order to get closer to the real spelling variant relation. This leads to the situation that the training set contains pairs of types that are labeled as not being spelling variants but that are considered spelling variants for the evaluation. For GML *unde* ('and'), the training set contains the following spelling variants: [*en*, *un*, *und*, *van*, *vn*, *vnd*, *vnde*, *vnnd*, *vnnde*, *vude*, *v̆nde*, *. . . nde*].[25] The full corpus, however, also contains *noch*, *unnde*, and *vnn* as further variants. While *noch* appears in the training set, it does only appear as spelling variant of *unde* in the development set. In general, a spelling variant relation induced from data as described above will contain pairs of types that are falsely labeled as not being spelling variants. This has to be taken into account when training tools for spelling variant detection, e. g., by addressing this as PU learning (see Section 4.1.1). We come back to this when describing the actual pipeline for spelling variant detection in Chapter 7.

5.3. Token-Level Evaluation

For most experiments in the following sections, the evaluation is performed on the token level. If this is not the case, it is pointed out in the description of the eval-

25. The ellipses in *. . . nde* encodes that the beginning of the word is not readable in the original manuscript.

P	R	F_1	C
0.32	1.00	0.49	14.38 ± 16.31

Table 5.3.: Precision, recall, F_1, and average number of candidates for the lookup approach (training set)

uation. Using the token level for evaluation has two implications. First, each type is weighted by its frequency as each token instance of a type is counted separately. We argue that it is more important to get frequent tokens right, which—with the exception of high-frequency function words—holds for a search application. Second, ambiguities that appear on the type level are resolved on the token level (see Jurish 2010b for a discussion of token-level normalization). This is illustrated by the example *koning* and *koninge* ('king'). Table 5.2 on the preceding page shows all morphological annotations of these two types in the corpus. As can be seen from these examples, *koninge* is analyzed as spelling variant of *koning* if the latter appears in dative or—as in example (9b)—in accusative case but not in nominative case as in example (9a).

(9) a. Do sprak de koning
Then said the king.NOM

b. Vnde wundede den koning
And injured the king.ACC
(Alexander Helmst.)

To see the effect of resolving type ambiguities on the token level, Table 5.3 shows micro-averaged precision (P), recall (R), F_1 values (see Section 4.2), and the average number of spelling variant candidates (C)—given as arithmetic mean with standard deviation—on the training data, when for each token, all types that appear as spelling variants of the respective type are added. While the recall is 1, the precision is rather low (0.32) due to the ambiguity on the token level. This low precision of 0.32 is an upper bound for type-based approaches that detect all spelling variants (i. e., that have a recall of 1).

We use the evaluation settings presented in this chapter for the experiments with simplification and spelling variant detection in the following two chapters.

Chapter 6.

Simplification

When applying NLP tools to non-standard data, spelling variation has to be taken care of. One way to do this is by removing variation in a preprocessing step. Such preprocessing can be a full normalization or a simplification to reduce some of the variation in the data.[26] Simplification can be as simple as removing extensive repetitions of characters as they are used for emphasis in Twitter messages (Han and Baldwin 2011) or more sophisticated by mapping allographs to one representation (e. g., Odebrecht et al. 2017 where characters or character-combinations specific for Early New High German are mapped to their modern equivalents, or even by spelling out abbreviations as described by Dipper 2010 for GMH).

In all the examples above, hand-crafted rules are used and their effectiveness of reducing variation (without at the same time introducing ambiguities) is not directly evaluated, only in terms of the task that the simplification rules are used for. We evaluate rule-based simplification for POS tagging and lemmatization in Chapter 8. In this chapter, we present a way to learn a set of simplification rules from spelling variant pairs, and compare this ruleset (SUP) with two approaches to simplification using hand-crafted rules (KOL and NIE) by evaluating them on GML data in the two settings described in Chapter 5.

6.1. Creating Simplification Rules – Different Approaches

The first set of simplification rules (KOL) consists of 26 rewrite rules that are applied to all types. These rules have been developed by linguists using three GML texts (Koleva et al. 2017).[27] They use regular expressions with lookahead and lookbehind, such as in Example (10) on the next page, which substitutes g with gh unless the g appears after an n or a g or before any of g, h or t.

26. See the definition of normalization and simplification in Chapter 3.

27. The script has been created by Melissa Farasyn in the project 'Corpus of Historical Low German' (CHLG; `http://www.chlg.ac.uk/index.html`) and contains rules by Melissa Farasyn with additions by Sarah Ihden and Katharina Dreessen both from the project 'Reference Corpus Middle Low German/ Low Rhenish (1200-1650)'.

(10) (?<![ng])g(?![ght]) → gh

There are no other limitations regarding the rules or their order. An advantage of such a ruleset is that it allows for a very fine-grained handling of spelling variation, e. g., treating characters differently according to their context. However, a disadvantage is that creating such a ruleset requires expert knowledge and is time-consuming. Furthermore, adding new rules is hard since interactions with existing rules have to be considered.

We compare this ruleset with other rulesets (NIE and SUP) that are deliberately more restricted regarding the expressiveness of the individual rules. While this leads to sacrifices regarding the granularity of the rules, we argue that it is easier to create and maintain such a ruleset.

The rulesets are simply defined by a set of undirected correspondences between characters or one character and a character bigram as in Example (12a) on the facing page. This allows the correspondences to model the edit operations given in Example (11).

(11) a. deletion/insertion with one character from the left or right context (e. g., $gh \rightarrow g$)
b. substitution (e. g., $i \rightarrow j$)
c. merges/splits (e. g., $ij \rightarrow y$)

Using these edit operations to define a ruleset leads to rule patterns that are similar to the rule patterns used by Pilz et al. (2006) with the difference that the rules are not weighted.

To transform the set of undirected correspondences into a ruleset with a unique order, two sorting operations are applied. First, each of the correspondences is sorted internally by length and then alphabetically such that correspondences like [‘g’, ‘gh’] and [‘gh’, ‘g’] or [‘i’, ‘j’] and [‘j’, ‘i’] lead to the same rules irrespective of their order, namely $gh \rightarrow g$ and $i \rightarrow j$, respectively. Then the rules themselves are sorted by length and alphabetically, to get the fixed ordering in which the rules are applied. Finally, rules that would substitute the same character are substituted with rules that map all of the characters involved in these rules to the same character. As an example, the two correspondences [‘i’, ‘j’] and [‘y’, ‘i’] lead, after sorting, to the two rules $i \rightarrow j$ and $i \rightarrow y$, in that order. As all instances of i are replaced by j due to the application of the first rule, the second rule would never be applied, leaving the characters j and y separate in the simplified language. To avoid this, these rules are replaced with the rules $i \rightarrow y$ and $j \rightarrow y$. This matches the fact that i, j, and y are equivalent according to the given correspondences. The ruleset for the correspondences in Example (12a) is given in Example (12b).

(12) a. [[‘g’, ‘gh’], [‘i’, ‘j’], [‘y’, ‘i’]]
b. $gh \rightarrow g$
$i \rightarrow y$
$j \rightarrow y$

For a specific type, the rules are applied in the specified order. With the rules in Example (12b) the spelling variants *ghjnc* and *ginc* (‘went’) will both be simplified to *gync*.

For these rulesets that are obtained from correspondences between characters, we compare a manually created set of such correspondences (NIE) with a set extracted from the training set (SUP). We have created NIE on the basis of a list of graphemes and allographes in GML taken from Niebaum (2000), a comprehensive overview of allographs in GML.

For SUP, we take all the spelling variant pairs from the training set that have an edit distance of 1 using the edit operations given in Example (11), each with a cost of 1. We extract the correspondences from these pairs. This set of correspondences is then pruned using measures for support and confidence (see Section 4.3). As support, we use a simple frequency threshold for the correspondences, i. e., a minimal number of pairs that a correspondence appears with. In the experiments, we use 10, 20, 30, and 40 as frequency thresholds. Among others, this helps to avoid extracting correspondences from errors (potential misspellings as well as transcription and annotation errors, cf. Chapter 3). As a confidence measure, we use the precision for the correspondences (Ernst-Gerlach and Fuhr 2006) on the training set. We experiment with the values between 0.4 and 0.85, incrementing in steps of 0.05.

The two manually created rulesets represent two approaches to rule-based simplification: KOL exemplifies a ruleset that has been manually created with a high precision in mind and allowing the rule creators to use regular expressions with lookahead and lookbehind on the left-hand side of the rule. NIE on the other hand uses a simple mechanism to define the ruleset. Furthermore, the list of rules is based on a comprehensive list of GML grapheme variants given in Niebaum (2000). Hence, while KOL lays more emphasis on precision, NIE is more tailored towards recall.

6.2. Evaluation

Table 6.1 on the next page shows the results for the two rulesets NIE and KOL and the rulesets extracted from the training set (SUP) with different thresholds for the frequency (Freq.) and the precision (Prec.) for the included rules. Regarding the manually created rulesets, they behave as expected: NIE has the higher recall in both settings (0.33 / 0.34). However, the aim of Niebaum (2000) is a presentation of possible grapheme variants and therefore does not take the precision of the variation

		text-eval				OOV-eval			
Freq.	Prec.	P	R	F_1	C	P	R	F_1	C
Kol		0.81	0.16	0.27	0.56 ± 0.98	0.65	0.09	0.16	0.22 ± 0.71
Nie		0.16	0.33	0.21	5.77 ± 7.32	0.09	0.34	0.14	6.00 ± 11.52
20	0.45	0.68	0.39	**0.50**	1.61 ± 1.72	0.44	0.31	**0.36**	1.08 ± 2.27
30	0.45	0.69	0.39	**0.50**	1.55 ± 1.72	0.46	0.27	0.34	0.91 ± 1.98
40	0.45	0.70	0.37	0.49	1.50 ± 1.68	0.46	0.24	0.32	0.81 ± 1.80
30	0.40	0.50	0.44	0.47	2.47 ± 3.56	0.33	0.31	0.32	1.41 ± 3.52
20	0.40	0.47	**0.45**	0.46	2.66 ± 3.81	0.31	**0.35**	0.33	1.70 ± 3.93
40	0.40	0.50	0.43	0.46	2.42 ± 3.56	0.33	0.28	0.30	1.28 ± 3.27
20	0.50	0.78	0.29	0.42	1.03 ± 1.11	0.58	0.25	0.35	0.65 ± 1.31
20	0.55	0.78	0.29	0.42	1.03 ± 1.11	0.58	0.25	0.35	0.65 ± 1.31
30	0.50	0.79	0.28	0.42	0.99 ± 1.10	0.58	0.24	0.34	0.63 ± 1.27
30	0.55	0.79	0.28	0.42	0.99 ± 1.10	0.58	0.24	0.34	0.63 ± 1.27
10	0.45	0.43	0.40	0.41	2.58 ± 2.73	0.27	0.29	0.28	1.63 ± 3.57
40	0.55	0.80	0.27	0.41	0.95 ± 1.07	0.58	0.21	0.31	0.55 ± 1.15
40	0.50	0.80	0.27	0.41	0.95 ± 1.07	0.58	0.21	0.31	0.55 ± 1.15
10	0.40	0.40	0.40	0.40	2.83 ± 3.06	0.26	0.30	0.28	1.80 ± 3.92
30	0.60	0.79	0.27	0.40	0.95 ± 1.07	0.59	0.23	0.33	0.59 ± 1.21
20	0.60	0.79	0.27	0.40	0.95 ± 1.07	0.59	0.23	0.33	0.60 ± 1.22
10	0.55	0.48	0.33	0.39	1.88 ± 2.05	0.36	0.27	0.31	1.16 ± 2.34
40	0.60	0.80	0.26	0.39	0.91 ± 1.04	0.59	0.20	0.30	0.52 ± 1.10
10	0.65	0.79	0.26	0.39	0.90 ± 1.07	0.61	0.23	0.33	0.57 ± 1.09
10	0.60	0.76	0.26	0.39	0.97 ± 1.13	0.55	0.24	0.33	0.66 ± 1.30
10	0.50	0.45	0.32	0.38	2.01 ± 2.27	0.34	0.25	0.29	1.16 ± 2.47
20	0.65	0.82	0.25	0.38	0.84 ± 0.95	0.62	0.20	0.31	0.50 ± 1.00
30	0.65	0.82	0.25	0.38	0.84 ± 0.95	0.62	0.20	0.31	0.50 ± 1.00
10	0.70	0.79	0.25	0.38	0.88 ± 1.07	0.62	0.22	0.32	0.54 ± 1.06
40	0.65	0.83	0.24	0.37	0.80 ± 0.92	0.64	0.19	0.29	0.45 ± 0.93
30	0.70	0.82	0.24	0.37	0.82 ± 0.95	0.63	0.19	0.29	0.47 ± 0.96
20	0.70	0.82	0.24	0.37	0.82 ± 0.95	0.62	0.19	0.30	0.48 ± 0.97
40	0.70	0.83	0.23	0.36	0.78 ± 0.91	0.64	0.18	0.28	0.42 ± 0.89
10	0.75	0.81	0.16	0.27	0.56 ± 0.82	0.65	0.14	0.23	0.32 ± 0.80
30	0.75	0.84	0.16	0.26	0.51 ± 0.82	0.64	0.13	0.22	0.31 ± 0.78
20	0.75	0.84	0.16	0.26	0.51 ± 0.82	0.64	0.13	0.22	0.31 ± 0.79
40	0.75	0.84	0.15	0.26	0.51 ± 0.81	0.65	0.13	0.21	0.29 ± 0.75
10	0.80	0.85	0.07	0.13	0.22 ± 0.53	0.51	0.02	0.04	0.06 ± 0.27
40	0.80	**0.94**	0.06	0.12	0.19 ± 0.51	0.52	0.02	0.04	0.05 ± 0.25
30	0.80	**0.94**	0.06	0.12	0.19 ± 0.51	0.52	0.02	0.04	0.05 ± 0.25
20	0.80	**0.94**	0.06	0.12	0.19 ± 0.52	0.51	0.02	0.04	0.06 ± 0.26
30	0.85	**0.94**	0.01	0.01	0.02 ± 0.13	**0.67**	0.00	0.00	0.00 ± 0.05
40	0.85	**0.94**	0.01	0.01	0.02 ± 0.13	**0.67**	0.00	0.00	0.00 ± 0.05
20	0.85	0.92	0.01	0.01	0.02 ± 0.13	0.53	0.00	0.00	0.01 ± 0.09
10	0.85	0.64	0.01	0.01	0.02 ± 0.16	0.56	0.00	0.00	0.01 ± 0.09

Table 6.1.: Precision, recall, F_1, and average number of candidates for different simplification rulesets (development set) – sorted by F_1 value of text-eval

into account. Consequently, the precision of the ruleset is quite low. KOL shows a better precision. This is expected as the ruleset from Koleva et al. (2017) applies the rules in a context-dependent fashion and has been created with high precision in mind.

The best of the rulesets regarding F_1 that are learned from the spelling variant pairs has a better recall then KOL and NIE in the text-eval setting (0.39). In the OOV-eval setting, the recall is slightly lower than that of NIE (0.31). And this ruleset does not show the same low precision as the ruleset based on Niebaum (2000), leading to the best F_1 values in the experiment for both settings, with having a frequency threshold of 20 and a precision threshold of 0.45. When precision is more important than recall, there are settings that lead to rulesets with a precision similar to that of KOL and a recall comparable with NIE for text-eval, e. g., a frequency threshold of 40 with a precision threshold of 0.5 or 0.55 (precision: 0.80 vs. 0.81, recall: 0.27 vs. 0.33).

In the OOV-eval setting, there is a smaller gap between the recall of NIE and that of the learned rules. The reason for this might be that the rules learned from the training set model the general variation in the development set well, as seen by the performance in the text-eval setting. But the rules miss unknown variants of words from the training set in the development set that are modeled by the more comprehensive ruleset NIE. What is interesting is that the recall obtainable with the learned ruleset is higher than that obtained by NIE. Table 6.2 on the following page shows all the correspondences from NIE and SUP. One reason for the better recall of the learned rules is that this set contains rules which cover sources of variation that go beyond simple allographs, e. g., the omission of a grapheme instead as in the rule $ge \rightarrow g$.

In this chapter, we have presented an approach to learn a simple ruleset from spelling variants. Compared with two different manually created rulesets we are able to obtain better results for the task of spelling variant detection. While evaluated here in the context of detecting spelling variants, as mentioned above, the rules can be used to remove variation from non-standard data before applying statistical NLP methods to it.

The correspondences between characters obtained by the presented method can also be useful for linguists studying spelling variation. For comparison, Dipper and Waldenberger (2017) present a method for automatically extracting variation patterns between dialects. While similar, their rules are directed, describing a difference between two dialects but cannot be used to describe the variation inside a text.

Regarding the task of spelling variant detection, the presented simplification rules still have a recall below 0.5, even the ruleset NIE based on a comprehensive list of graphemic variation in GML. This can be explained by the fact that spelling variation arises not only from graphematic variation. In the following section, we look into obtaining better results for spelling variant detection by extracting spelling variants from a dictionary using surface similarity and by filtering these spelling variant candidates afterwards.

LHS	RHS	Nie	Sup (20, 0.45)	LHS	RHS	Nie	Sup (20, 0,45)
aa	a	x		ph	v	x	
ae	a	x	x	re	r		x
ai	a	x		sc	s	x	
a͵e	a	x	x	ss	s	x	x
ch	c		x	sz	s		x
ch	g	x	x	td	t	x	
ch	k	x	x	th	d	x	
dh	d	x		th	t	x	x
de	d		x	tt	t	x	x
dt	d		x	ui	u	x	
ee	e	x	x	uw	u	x	
eh	e		x	u͵e	u	x	x
ei	e	x	x	u͵o	u	x	x
ey	e	x	x	u͵v	u		x
e͵e	e	x		a	o	x	
ff	f		x	b	p	x	
ff	v	x		c	g		x
ge	g		x	c	k	x	x
gh	g	x	x	d	t	x	x
ii	i	x		e	h		x
ij	i	x	x	e	i	x	
ij	y		x	e	e		x
i͵e	i	x		e	y	x	
jh	g	x		f	v	x	x
jh	j	x		h	e		x
ll	l	x	x	i	j	x	x
me	m		x	i	y	x	x
nc	n		x	j	y	x	x
nd	n		x	k	q	x	
nn	n		x	o	u	x	
oe	o	x	x	ö	o	x	
oi	o	x		ö	ø	x	
oo	o	x		s	z	x	x
ou	o	x		u	v	x	x
oy	o	x		u	w	x	
o͵e	o	x	x	ü	u	x	
o͵v	o	x		ü	y	x	
ph	p	x		v	w	x	

Table 6.2.: Comparison of correspondences between characters

Chapter 7.

A Pipeline for Spelling Variant Detection

In this chapter, we present experiments for spelling variant detection. Given a token and a list of types—the dictionary—the aim is to identify spelling variants for the given token from the dictionary.

The general approach that we apply here consists of three steps. In the first step, spelling variant candidates for the given type are retrieved from the dictionary. For this, we experiment with different measures for surface similarity (cf. Section 4.8.1). In this first *candidate generation* step, the context information is ignored. In a second step, the generated candidates are filtered using context information and/ or more sophisticated ways to evaluate the surface similarity. In this step, context information is not the specific context of a given token, but the context in which a type can appear. Hence, the methods presented up to this point generate spelling variant candidates on the type level. In a third and last step, these candidates are filtered taking the specific token context into account.[28] Figure 7.1 on the next page gives an overview of these different steps in the whole pipeline.

The presentation of the different steps is divided into two parts: In Section 7.1, we present methods that do not need annotated training data and are therefore usable in an unsupervised setting (cf. Section 4.1.3). For candidate generation using surface similarity, in this setting, we experiment with different edit distances and variants of the Jaccard Index using character n-grams as similarity features. For the type-based filter, we apply Brown clusters (cf. Section 4.8).

For supervised settings (cf. Section 4.1.1), we present approaches in Section 7.2. We use the training data to improve the candidate generation by combining the list of generated candidates with known candidates for a given type. The similarity-

28. When generating candidates using a dictionary as in the experiments in the thesis the generation step can be seen as a filter as well: The whole dictionary is filtered to find likely candidates. Hence, the pipeline can also be seen as two filter steps: first, a broad filter to get probable candidates from the dictionary and then a second more fine-grained filter that is only applied to these probable candidates. However, we keep the distinction between generator and filter as the candidate generation step does not necessarily have to rely on filtering a given dictionary (see Chapter 9).

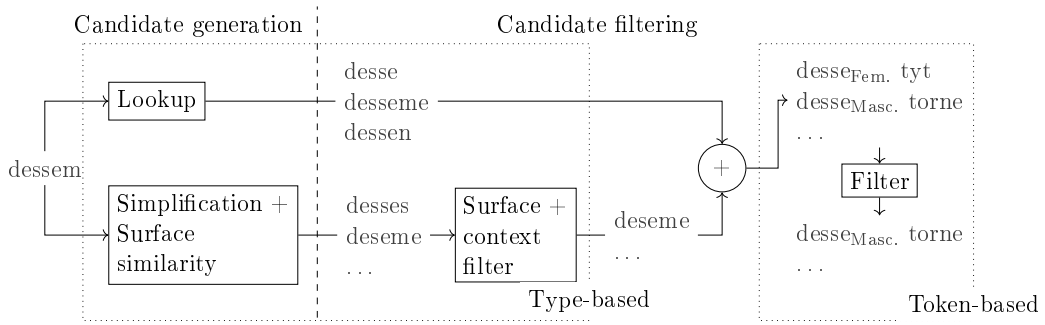


Figure 7.1.: Overview of the spelling variant detection pipeline

based generation is also combined with the simplification approach presented in the previous chapter by first removing variation from the data before measuring the similarity between types. For the type-based filter in the second step, different filters are presented using the observed differences in the surface forms from the training data to refine the surface-similarity measures. We use variants of an edit distance that assign lower costs for edit operations that are known to potentially lead to spelling variants from the training data. We apply a variant of SPSim (Gomes and Lopes 2011) which uses either a cost of 0 or 1 and a self developed distance measure that uses costs between 0 and 1. We compare these with an SVM and a Bagging-SVM (BSVM) (Mordelet and Vert 2014), which is better suited for the training data, using both surface and contextual features (Section 7.2.2).

In the third step, another filter is applied, using a supervised classifier. This time token-context information is taken into account. The idea here is to first reduce the set of generated spelling variant candidates to actual spelling variants for the given type and afterwards removing spelling variants that are not possible in the given context. For the token-based filtering, we apply a CNN to encode the token sequence using a combination of context and surface based embeddings to represent the individual words (Section 7.2.3).

The runtime of the candidate generation step is governed by searching in the dictionary for candidates with an edit distance that is lower than or equal to the given threshold. For candidate filtering, the trained classifiers (SVM and CNN) have to be applied to each of the candidates, which can be done in parallel.

The intermediate results presented in the following sections are obtained from the development set. Section 7.3 contains final results and an error analysis on the test set.

		text-eval				OOV-eval			
Method	Parameters	P	R	F_1	C	P	R	F_1	C
Lev	1	0.21	0.52	0.30	6.95 ± 5.83	0.20	0.27	0.23	2.10 ± 3.22
Lev	2	0.04	0.82	0.07	64.12 ± 55.31	0.04	0.64	0.07	25.86 ± 47.71

Table 7.1.: Precision, recall, F_1, and average number of candidates for the Levenshtein distance (development set)

7.1. Unsupervised Detection of Spelling Variants

For the unsupervised setting, we evaluate different surface (or string) similarity measures for quickly narrowing down likely spelling variant candidates in a dictionary for a given type. In a second step, these types are filtered using Brown clusters (Brown et al. 1992).

The most obvious candidate for string similarity is the Levenshtein distance (Levenshtein 1966). However, its usage for normalization has been criticized for two main reasons: firstly for its computational complexity (Jin 2015) and secondly for its coarse-grainedness as for the standard Levenshtein distance all differences have the same cost (Bollmann 2013a). Regarding the computational complexity: different approaches have been proposed that make extracting spelling variant candidates from a dictionary computationally tractable. One way is to first use anagram-hashing to retrieve the set of anagrams and their neighbors defined by deletion, insertion and substitution (Reynaert 2004) and only filter this set using the Levenshtein distance. This way, the Levenshtein distance does not have to be computed for the whole dictionary. For our experiments, we have used the Levenshtein automaton proposed by Schulz and Mihov (2002) that has been shown to be efficient for retrieving similar words from a dictionary. With this approach, the runtime of the generation step is linear with regard to the size of the dictionary (Schulz and Mihov 2002). The runtime can also be improved by incorporating filtering techniques (Mihov and Schulz 2004).

Regarding the coarse-grainedness, Table 7.1 shows recall, precision and F-score as well as the average number of candidate pairs per type (arithmetic mean and standard deviation) for spelling variant detection using only the Levenshtein distance with a maximum distance of 1 and 2 respectively. The precision is always very low, even at a Levenshtein distance of 1. There are two reasons for the low precision. One is the fact that many types do not have spelling variants and the other is the coarse-grainedness of the distance.

Table 7.1 also shows how the number of candidates increases dramatically with a higher Levenshtein distance. This is illustrated by Example (13). Here, the number of spelling variant candidates for *spyse* 'dish/food' is increased by the factor of 7

between Levenshtein distance 1 and 2 while no new spelling variants are discovered (only *spise* and *spysze* are actual spelling variants of *spyse*, whereas *spysen* '(to) dine' is a related verb).

(13) *spyse*

a. Distance 1:
spysen, ***spysze***, ***spise***

b. Distance 2:
spise, *syde*, *sesse*, ***spysze***, *pyne*, *spysen*, *pyze*, *syne*, *spere*, *syst*, *spyede*, *swyge*, *ryse*, *spade*, *wyse*, *syme*, *spyker*, *swyne*, *spele*, *sluse*, *spyl*, *pryse*

In the following sections, we compare the Levenshtein distance with other surface similarity measures, i. e., n-gram similarities that have been used for normalization and different variants of the edit distance in order to obtain a better balance between precision and recall.

7.1.1. N-Gram Similarity

One alternative to edit-based distance measures like the Levenshtein distance that have been used for normalization or detecting spelling variants are similarity measures based on character n-grams (cf. Section 4.8.1). Kestemont, Daelemans, and de Pauw (2010) compare the Levenshtein distance with Dice's coefficient (Dice 1945) using character bigrams of the words to detect spelling variants in Middle Dutch texts to improve lemmatization. While they report a better performance for the Levenshtein distance, Jin (2015) achieves good results with the Jaccard Index (Levandowsky and Winter 1971) for candidate generation in normalizing English Twitter data and also proposes a weighted version.[29] This weighted version is given in Equation (7.1).

$$JaccardIndex_w(f(t_1), f(t_2)) = \frac{\sum_{\mathrm{f} \in f(t_1) \cap f(t_2)} w(\mathrm{f})}{\sum_{\mathrm{f} \in f(t_1) \cup f(t_2)} w(\mathrm{f})} \quad (7.1)$$

Here, $f(t) \subseteq F$ is the set of similarity features for a type t and $w : F \to \mathbb{R}$ is a weight function. Both can be chosen differently allowing to fine-tune the measure

29. As $Dice(x, y) = \frac{2 * JaccardIndex(x,y)}{JaccardIndex(x,y)+1}$ (Egghe 2010) Dice's coefficient and the Jaccard Index will give the same results in our threshold setting. We restrict ourselves to the Jaccard Index.

					text-eval				OOV-eval			
	Sim.	Min	Max	Skip	P	R	F_1	C	P	R	F_1	C
	Levenshtein 1				0.21	**0.52**	0.30	6.95 ± 5.83	0.20	0.27	0.23	2.10 ± 3.22
best settings for text-eval	0.35	2	∞	4	0.31	0.34	**0.32**	3.10 ± 2.89	0.23	0.17	0.20	1.11 ± 2.13
	0.3	2	∞	2	0.30	0.34	**0.32**	3.21 ± 2.99	0.21	0.14	0.17	1.03 ± 1.87
	0.35	1	∞	3	0.30	0.34	**0.32**	3.18 ± 3.03	0.22	0.15	0.17	1.02 ± 2.13
	0.35	1	∞	4	0.29	0.35	**0.32**	3.35 ± 3.14	0.22	0.18	0.20	1.24 ± 2.59
	0.4	2	4	3	0.29	0.36	**0.32**	3.37 ± 2.82	0.18	0.28	0.22	2.36 ± 2.99
	0.4	2	4	4	0.28	0.37	**0.32**	3.65 ± 3.14	0.17	**0.31**	0.22	2.76 ± 3.66
	0.4	1	4	3	0.27	0.38	**0.32**	3.90 ± 3.34	0.17	0.29	0.21	2.69 ± 3.63
	0.4	2	∞	4	0.32	0.29	0.31	2.56 ± 2.45	**0.26**	0.14	0.18	0.81 ± 1.31
	0.3	3	∞	3	0.31	0.30	0.31	2.67 ± 2.47	0.23	0.16	0.19	1.09 ± 1.96
	0.3	3	∞	4	0.31	0.31	0.31	2.74 ± 2.50	0.23	0.19	0.21	1.28 ± 2.19
best settings for OOV-eval	0.25	4	∞	4	**0.34**	0.23	0.27	1.90 ± 1.98	0.24	0.24	**0.24**	1.48 ± 1.98
	0.35	4	4	4	0.29	0.17	0.22	1.66 ± 2.11	0.21	0.26	0.23	1.87 ± 2.30
	0.5	2	4	4	**0.34**	0.23	0.28	1.93 ± 1.95	0.25	0.19	0.22	1.21 ± 1.54
	0.4	4	4	4	0.32	0.15	0.20	1.28 ± 1.71	0.25	0.20	0.22	1.27 ± 1.58
	0.5	1	4	4	0.33	0.24	0.27	2.02 ± 1.99	0.24	0.21	0.22	1.32 ± 1.72
	0.45	3	4	4	0.31	0.17	0.22	1.49 ± 1.87	0.24	0.20	0.22	1.27 ± 1.60
	0.5	3	3	4	0.32	0.24	0.27	2.11 ± 2.21	0.21	0.23	0.22	1.74 ± 2.37
	0.45	2	4	4	0.31	0.25	0.27	2.20 ± 2.21	0.21	0.23	0.22	1.69 ± 2.37
	0.45	1	4	4	0.29	0.32	0.31	3.06 ± 2.70	0.20	0.24	0.22	1.85 ± 2.65
	0.4	3	4	4	0.32	0.24	0.27	2.13 ± 2.24	0.20	0.25	0.22	1.85 ± 2.51

Table 7.2.: Precision, recall, F_1, and average number of candidates for the Jaccard Index (development set)

for specific data and tasks. For normalizing Twitter data, Jin (2015) uses bigrams and skip-1-bigrams as similarity features. The weight for each feature is set to 1. We test this measure with different features.

Jin (2015) notes that this similarity measure with the given similarity features is not a metric since a similarity value of 1 does not mean that the compared strings are identical. This can be seen with the example *lool* and *loool* when using only the set of character bigrams as features: both words have the same featureset, namely *$l*, *lo*, *oo*, *ol* and *l$*, and therefore have a similarity of 1. This, however, is only the case if the similarity features do not contain the whole strings as a feature. In our experiments, we use one setting where we do not limit the maximum length of the n-grams but allow for an n-gram-size up to the length of the words.

Table 7.2 shows the 10 best results regarding F-value for the unweighted Jaccard Index (i. e., setting the weight for all features to 1) using character n-grams with a minimum length $\in \{1, 2, 3, 4\}$ and a maximum length $\in \{2, 3, 4, \infty\}$ optionally allowing skips of the length $\in \{1, 2, 3, 4\}$ for both evaluation settings with similarity thresholds $\in \{0.1, 0.15, 0.2, 0.25.0.3, 0.35, 0.4, 0.45, 0.5\}$.

As can be seen from the data, the Jaccard Index reaches slightly better F_1 values than the Levenshtein distance (0.32 / 0.24 vs. 0.30 / 0.23). This result is reached

				text-eval				OOV-eval			
Sim.	Min	Max	Skip	P	R	F_1	C	P	R	F_1	C
Levenshtein 1				0.21	0.52	0.30	6.95 ± 5.83	0.20	0.27	0.23	2.10 ± 3.22
0.35	2	4	4	0.23	0.42	**0.30**	5.06 ± 4.39	0.12	0.36	0.18	4.41 ± 5.98
0.15	3	∞	1	0.24	0.41	**0.30**	4.83 ± 3.75	0.13	0.30	0.18	3.5 ± 4.3
0.30	1	4	2	0.16	**0.52**	0.25	8.78 ± 6.95	0.08	0.45	0.14	8.25 ± 9.20
0.20	2	∞	1	**0.21**	0.45	0.29	5.95 ± 4.91	0.14	0.29	0.19	3.22 ± 4.60
0.35	4	4	4	0.29	0.17	0.22	1.66 ± 2.11	0.21	0.26	**0.23**	1.87 ± 2.30
0.25	3	∞	3	0.27	0.37	0.31	3.81 ± 3.45	0.16	**0.27**	0.20	2.51 ± 4.21
0.40	3	4	4	0.32	0.24	0.27	2.13 ± 2.24	**0.20**	0.25	0.22	1.85 ± 2.51

Table 7.3.: Precision, recall, F_1, and average number of candidates for the Levenshtein distance and the Jaccard Index (development set)

through a better trade off between precision and recall, i. e., the recall is lower than with the Levenshtein distance while the precision is higher. Notably, the best choice of similarity features differs for both evaluation settings.

For better comparison, Table 7.3 shows the best results of the Jaccard Index with the same recall, precision and F_1 value each as the Levenshtein distance of 1. The comparison reveals the different trade offs between precision and recall: with the same F_1 value as the Levenshtein distance, the recall is still lower for the Jaccard Index. With the same recall, precision is lower, leading to a lower F_1 value.

We look into another string similarity measure that uses character n-grams as similarity features named *Proxinette* (Hathout 2009, 2014). Proxinette has been designed to measure the morphological similarity of lexemes and to find morphologically related words in a dictionary. We use Proxinette in the variant described in Hathout (2014), where character n-grams are the only features.

The basic intuition behind Proxinette is the same as the basic intuition behind the Jaccard Index: the more character n-grams two types share, the more similar they are. In contrast to the Jaccard Index, the n-grams in Proxinette are weighted by their frequency in the corpus: More frequent n-grams contribute less to the similarity. In the original version of Proxinette, the character n-grams have a minimum length of three. However, this leaves spelling variants such as *yck* and *ic* ('I') unconnected leading to a similarity value of 0. Therefore, we vary the minimal n-gram length as a parameter in our experiments.

Proxinette returns the similarity of two types as a number between 0 and 1: 0 means no similarity (i. e., no shared n-gram), while higher values denote a higher similarity. The similarity features used in Proxinette are not only character n-grams of given lengths, but all possible n-grams above a given length including the whole type. This is a way to prevent a similarity of 1 for two different types as noted above.

					text-eval				OOV-eval			
	Sim.	Min	Max	Skip	P	R	F_1	C	P	R	F_1	C
Jacc.	0.35	1	∞	0	0.22	0.24	0.23	3.01 ± 2.57	0.13	0.16	0.14	1.91 ± 2.36
	0.30	1	∞	0	0.17	0.32	0.22	5.43 ± 5.93	0.12	0.28	0.16	3.72 ± 4.38
Proxinette	0.025	3	∞	0	0.10	0.15	**0.12**	4.23 ± 3.91	0.08	0.17	0.10	3.53 ± 2.28
	0.01	2	∞	0	0.07	0.25	0.11	9.53 ± 7.16	0.05	0.23	0.08	7.12 ± 3.81
	0.01	1	∞	0	0.08	0.19	0.11	6.62 ± 5.09	0.06	0.22	0.09	5.97 ± 3.21
	0.025	2	∞	0	0.10	0.13	0.11	3.46 ± 3.11	0.08	0.14	**0.11**	2.62 ± 1.79
	0.05	3	∞	0	0.13	0.09	0.11	2.04 ± 2.10	0.10	0.11	0.10	1.56 ± 1.35
	0.01	3	∞	0	0.06	0.24	0.10	10.61 ± 9.12	0.04	0.26	0.07	9.18 ± 5.15
	0.025	1	∞	0	0.12	0.09	0.10	2.16 ± 2.19	0.09	0.13	**0.11**	2.09 ± 1.53
	0.075	3	∞	0	0.20	0.06	0.09	0.88 ± 1.24	0.12	0.07	0.09	0.88 ± 0.91
	0.05	2	∞	0	0.19	0.06	0.09	0.89 ± 1.29	0.12	0.09	0.10	1.09 ± 1.02
	0.05	1	∞	0	0.21	0.04	0.07	0.57 ± 0.96	0.13	0.07	0.09	0.84 ± 0.87
	0.1	3	∞	0	0.21	0.03	0.06	0.45 ± 0.79	0.14	0.06	0.08	0.61 ± 0.75
	0.075	2	∞	0	0.22	0.03	0.05	0.39 ± 0.75	0.14	0.06	0.08	0.62 ± 0.74
	0.001	1	∞	0	0.01	0.62	0.03	118.19 ± 56.87	0.01	0.55	0.02	67.40 ± 26.86
	0.001	3	∞	0	0.02	0.49	0.03	84.23 ± 71.57	0.01	**0.59**	0.02	85.80 ± 38.86
	0.075	1	∞	0	0.20	0.02	0.03	0.22 ± 0.53	**0.16**	0.05	0.08	0.48 ± 0.66
	0.1	2	∞	0	**0.23**	0.01	0.03	0.18 ± 0.47	**0.16**	0.04	0.07	0.43 ± 0.62
	0.001	2	∞	0	0.01	**0.66**	0.02	147.80 ± 78.53	0.01	**0.59**	0.02	80.79 ± 35.84
	0.1	1	∞	0	0.16	0.00	0.01	0.09 ± 0.30	0.18	0.04	0.06	0.32 ± 0.54

Table 7.4.: Precision, recall, F_1, and average number of candidates for Proxinette (development set)

Table 7.4 on the preceding page shows the results using Proxinette for spelling variant candidate generation with varying minimal n-gram size ($\in \{1, 2, 3\}$) and different similarity thresholds ($\in \{0.1, 0.075, 0.05, 0.025, 0.01, 0.001\}$). For a better comparison, it also shows the best results using the Jaccard Index without skips and using n-grams up to the whole type. As is obvious, using the Jaccard Index shows a better trade off between precision and recall. The reason for this can be seen by comparing how the similarity scores are computed. Differing from the Jaccard Index, in Proxinette the similarity score is obtained by the probability of a random walk in the bipartite graph with types and similarity features as vertices. This leads to a weight for the features of $\frac{1}{deg(\mathrm{f})}$, where $deg(\mathrm{f})$ is the degree of the vertex f that represents the feature f in the bipartite graph used for calculating Proxinette, i. e., the number of types having this similarity feature.

Equation (7.2) on the current page shows the calculation of Proxinette similarity scores using a random walk and an equivalent formulation using the sum of the weighted features common to t_1 and t_2 that allows a direct comparison with the Jaccard Index in Equation (7.1) on page 84:

$$Proxinette(t_1, t_2) = \sum_{\mathrm{f} \in f(t_1) \cap f(t_2)} \frac{1}{deg(t_1)} * \frac{1}{deg(\mathrm{f})} = \frac{\sum_{\mathrm{f} \in f(t_1) \cap f(t_2)} \frac{1}{deg(\mathrm{f})}}{deg(t_1)} \overset{w(\mathrm{f}) = \frac{1}{deg(\mathrm{f})}}{=} \frac{\sum_{\mathrm{f} \in f(t_1) \cap f(t_2)} w(\mathrm{f})}{\sum_{\mathrm{f} \in f(t_1)} 1} \tag{7.2}$$

As in Equation (7.1), $f(t) \subseteq F$ is the set of similarity features for a given type t. $deg(t)$ is the degree of the vertex representing the type t in the bipartite graph. It is the number of similarity features used to represent a type, i. e., $deg(t) = |f(t)| = \sum_{\mathrm{f} \in f(t_1)} 1$. By setting $w(\mathrm{f}) = \frac{1}{deg(\mathrm{f})}$, the numerator is the same as the numerator of the weighted Jaccard Index in Equation (7.1). However, the denominator differs: One problem with respect to spelling variation is that only the features of the type for which spelling variants should be retrieved, $\mathrm{f} \in f(t_1)$, are considered in the denominator. Essentially, this means that insertions at the beginning or end of a word do not influence the similarity. This results in compounds being found as spelling variant candidates as can be seen with Example (14). The example shows some spelling variant candidates generated using Proxinette for the nominative singular of *hus* ('house') containing the compounds *hus frouwe* and *husman* ('woman/man of the house'). Actual spelling variants are given in bold face.

(14) *hus*

a. Proxinette min-n-gram: 1, sim: 0.1:
husz, ***huse***, *husze*, *hus frouwe*, *husman*, *husmans*

Another difference between Proxinette and the weighted Jaccard Index is that the features are not weighted in the denominator of Proxinette leading to the situation that a similarity of 1 is not possible if $w(\mathrm{f}) < 1$ is true for any similarity feature f for a given word t.

To overcome these shortcomings of Proxinette for spelling variant detection, we use the frequency-based weighting of similarity features with the Jaccard Index. However, in the original form, the features are weighted by the absolute number of types they appear with ($\frac{1}{deg(\mathrm{f})}$). Therefore, the weights depend on the corpus size. We use $1 - relFreq(f)$ as a weight function. Thereby we keep the general idea of giving more weight to infrequent similarity features while avoiding the dependency on the corpus size.[30] The relative frequency is estimated based on the dictionary from which spelling variants are generated leading to a weight of 0 for all features that appear in every type of the dictionary, and 1 for features that do not appear in it.

Table 7.5 on the next page shows the 10 best results regarding F-value for the Jaccard Index using frequency-based weighting and character n-grams with a minimum length $\in \{1, 2, 3, 4\}$ and a maximum length $\in \{2, 3, 4, \infty\}$ optionally allowing skips of the length $\in \{1, 2, 3, 4\}$ for both evaluation settings with similarity thresholds $\in \{0.1, 0.15, 0.2, 0.25.0.3, 0.35, 0.4, 0.45, 0.5\}$. For comparison with the unweighted Jaccard Index, the two best results from Table 7.2 on page 85 have been added as well. The data show that frequency-weighting leads to a better precision. With regard to the F-value however, there is no improvement in the OOV-eval setting and only a small improvement in the text-eval setting.

To conclude, with a similarity measure based on character n-grams with possible skips it is possible to obtain better F-values than using the Levenshtein distance. However, the improvements are small and come from a better precision and a lower recall. When the parameters for the Jaccard Index are chosen such that the recall is comparable with that of the Levenshtein distance, the precision is lower. Hence, the Levenshtein distance is a better candidate generator if filtering is applied afterwards.

In the next section, we look at different edit distances in order to improve the recall without the high loss in precision that comes with a Levenshtein distance of 2.

30. This weight is only dependent on the size of the corpus in the sense that bigger corpora lead to better estimates of the relative frequency.

					text-eval				OOV-eval			
	Sim.	Min	Max	Skip	P	R	F_1	C	P	R	F_1	C
Levenshtein 1					0.21	0.52	0.30	6.95 ± 5.83	0.20	0.27	0.23	2.10 ± 3.22
Jacc.	0.35	2	∞	4	0.31	0.34	0.32	3.10 ± 2.89	0.23	0.17	0.2	1.11 ± 2.13
	0.25	4	∞	4	0.34	0.23	0.27	1.90 ± 1.98	0.24	0.24	**0.24**	1.48 ± 1.98
wJacc. – best text-eval	0.35	2	∞	4	0.34	0.33	**0.33**	2.69 ± 2.37	0.25	0.16	0.20	0.99 ± 1.59
	0.35	1	∞	3	0.33	0.33	**0.33**	2.82 ± 2.60	0.24	0.13	0.17	0.85 ± 1.44
	0.35	1	∞	4	0.33	0.34	**0.33**	2.84 ± 2.50	0.24	0.17	0.20	1.05 ± 1.80
	0.4	1	4	3	0.31	0.34	**0.33**	3.12 ± 2.43	0.18	0.27	0.22	2.28 ± 2.78
	0.3	2	∞	4	0.29	**0.37**	**0.33**	3.48 ± 2.99	0.20	0.23	0.21	1.79 ± 2.91
	0.4	2	4	3	0.32	0.34	0.32	2.96 ± 2.32	0.19	0.25	0.22	2.09 ± 2.63
	0.3	2	∞	2	0.31	0.33	0.32	2.94 ± 2.75	0.21	0.13	0.16	0.93 ± 1.63
	0.4	2	4	4	0.31	0.33	0.32	2.98 ± 2.42	0.19	0.29	0.23	2.32 ± 2.97
	0.3	1	∞	2	0.30	0.34	0.32	3.10 ± 2.82	0.21	0.14	0.17	1.03 ± 1.77
	0.3	2	∞	3	0.30	0.35	0.32	3.24 ± 2.91	0.21	0.18	0.20	1.30 ± 2.35
wJacc. – best OOV-eval	0.25	4	∞	4	0.34	0.23	0.27	1.88 ± 1.96	0.25	0.23	**0.24**	1.45 ± 1.90
	0.35	4	4	4	0.30	0.17	0.22	1.59 ± 1.97	0.22	0.26	0.23	1.82 ± 2.18
	0.4	2	4	4	0.31	0.33	0.32	2.98 ± 2.42	0.19	0.29	0.23	2.32 ± 2.97
	0.4	1	4	4	0.30	0.35	0.32	3.18 ± 2.52	0.18	**0.30**	0.23	2.49 ± 3.19
	0.5	1	4	4	**0.38**	0.21	0.27	1.59 ± 1.58	**0.26**	0.19	0.22	1.13 ± 1.42
	0.45	3	4	4	0.32	0.16	0.21	1.36 ± 1.71	0.25	0.20	0.22	1.24 ± 1.55
	0.4	4	4	4	0.32	0.15	0.20	1.26 ± 1.69	0.25	0.20	0.22	1.26 ± 1.57
	0.45	1	4	4	0.32	0.26	0.29	2.29 ± 1.94	0.22	0.23	0.22	1.58 ± 1.98
	0.45	2	4	4	0.33	0.24	0.28	2.01 ± 1.96	0.22	0.22	0.22	1.50 ± 1.83
	0.5	3	3	4	0.34	0.21	0.26	1.70 ± 1.86	0.22	0.23	0.22	1.58 ± 1.91

Table 7.5.: Precision, recall, F_1, and average number of candidates for the Jaccard Index with frequency-weighted feautures (development set)

7.1.2. Different Edit Distances

In the previous section, we have seen that with surface similarity measures based on character n-grams it is possible to get better F_1 values compared to using the Levenshtein distance. However, this improvement comes from an improvement in precision and a loss in recall. Hence, we use the Levenshtein distance as generator for the subsequent filtering. Still, the problem remains that the Levenshtein distance is very coarse-grained and cannot be easily fine-tuned. While using a distance of 1 leads to an average of about 7 candidates for each word in the text-eval setting, using a distance of 2 already leads to an average number of about 64 candidates, resulting in a drop in precision from 0.21 to 0.04 (cf. Table 7.1 on page 83).

In this section, we present experiments with two options to find an edit distance that is more fine-grained so that we obtain a better recall without the loss in precision that comes with a Levenshtein distance of 2. The first option is the addition of edit operations to the Levenshtein distance. The second option is avoiding a fixed distance threshold for all types and to make the threshold dependent on the length of the type for which the candidates are obtained.

Regarding the addition of edit operations to the Levenshtein distance, we introduce two variants that can be found in the literature and a new variant:

First, transpositions (T), $ab \rightarrow ba$, are sometimes added to the atomic edit operations that have a cost of 1 (Damerau 1964). A second modification is the treatment of arbitrary merges and splits (MS) (i. e., $ab \leftrightarrow c$) as atomic operations with a cost of 1. In GML texts, this appears with *ij* and *y*, e. g., in the types *gij* and *gy*, which are spelling variants of *ji* ('you', 2PL.NOM). As a third modification, we introduce a new edit operation. It is inspired by a common preprocessing step for social media data, i. e., the deletion of repeated characters (Han, Cook, and Baldwin 2013). Instead of removing repetitions in a preprocessing step, we integrate this directly into the metric by allowing repetitions of common characters (Re) with a cost of 0 (or: after a match, insertions/deletions of the matched character have a cost of 0):

$$\forall_{n \in \mathbb{N}_1, x \in \Sigma} d(x, x^n) = 0 = d(x^n, x)$$

With this addition, the edit distance is no longer a metric in the mathematical sense as for example $d(lol, looool) = 0$. We have shown a similar property for n-gram based distances on page 85 in Section 7.1.1. Furthermore, allowing repetitions removes the higher penalty of additions when compared to the Jaccard Index. This behavior of the Jaccard Index and the standard Levenshtein distance regarding repetitions has been highlighted by Jin (2015) as one of the advantages of the character n-gram based Jaccard Index.

As noted above, we use Levenshtein automata (Mihov and Schulz 2004) in order to search the set of all types for the neighbors of a given type. T and MS are

			text-eval				OOV-eval			
T	MS	Re	P	R	F_1	C	P	R	F_1	C
-	-	-	**0.21**	0.52	**0.30**	6.95 ± 5.83	**0.20**	0.27	**0.23**	2.10 ± 3.22
x	-	-	**0.21**	0.52	**0.30**	7.05 ± 5.92	**0.20**	0.28	**0.23**	2.14 ± 3.29
-	x	-	0.09	0.64	0.15	20.28 ± 20.01	0.10	0.35	0.16	5.27 ± 9.66
-	-	x	0.20	0.59	0.29	8.31 ± 7.05	0.18	0.34	**0.23**	2.96 ± 5.28
x	x	-	0.09	0.64	0.15	20.37 ± 20.10	0.10	0.36	0.16	5.31 ± 9.71
x	-	x	0.20	0.59	0.29	8.38 ± 7.12	0.18	0.34	**0.23**	2.99 ± 5.34
-	x	x	0.09	**0.69**	0.15	22.28 ± 21.01	0.09	**0.41**	0.15	7.12 ± 13.49
x	x	x	0.09	**0.69**	0.15	22.35 ± 21.09	0.09	**0.41**	0.15	7.15 ± 13.54

Table 7.6.: Precision, recall, F_1, and average number of candidates for different edit distances (development set)

added to the automata as described by Schulz and Mihov (2001). Re is added by introducing states of the form $(i^{\#e}, x)$ for a position i in a word, an error value e, and a character $x \in \Sigma$ to the automaton. These states can be reached from a match with character x in position $i-1$. From this state, insertions and deletions of x have a cost of 0.

Table 7.6 gives the results of using the different edit distances with a maximum distance of 1. These results show that adding transpositions and repetitions to the Levenshtein distance leads to a small improvement in recall with only a small loss in precision for both the text-eval and the OOV-eval setup. While adding merges and splits leads to an even higher recall, the loss in precision is higher too. Adding transpositions and repetitions leads to the same F_1 value as using the Levenshtein distance in the OOV-eval setting with more emphasis on recall. In the text-eval setting, the F_1 value decreases hardly (from 0.3 to 0.29) while recall improves from 0.52 to 0.59. This is exactly what we want to achieve: a better recall with a loss in precision that is as small as possible.

Regarding the usage of different distance thresholds depending on the given type, one option is to normalize the edit distance by the length of this type (Yujian and Bo 2007). Setting a threshold $0 < t$ for the ratio between the length of the type and the edit distance, the maximum edit distance d to find spelling variant candidates for a given type T with length l_T can be computed by using the following formula, note that we use a minimum Levenshtein distance of 1 in order to generate candidates for short types as well:

$$d = \max(1, \lfloor l_T * t \rfloor)$$

	text-eval				OOV-eval			
t	P	R	F_1	C	P	R	F_1	C
0.1	**0.21**	0.52	**0.30**	6.95 ± 5.83	**0.20**	0.27	0.23	2.10 ± 3.22
0.15	**0.21**	0.52	**0.30**	6.96 ± 5.82	**0.20**	0.27	0.23	2.13 ± 3.22
0.2	**0.21**	0.53	**0.30**	7.05 ± 5.78	**0.20**	0.31	**0.24**	2.39 ± 3.23
0.25	0.20	0.53	0.29	7.29 ± 5.70	0.16	0.39	0.22	3.78 ± 4.64
0.3	0.20	0.54	0.29	7.77 ± 5.64	0.12	0.47	0.19	6.27 ± 7.13
0.35	0.16	0.56	0.25	9.95 ± 8.00	0.06	0.56	0.11	13.30 ± 17.93
0.4	0.10	**0.59**	0.18	16.04 ± 16.87	0.03	**0.68**	0.06	31.15 ± 36.82

Table 7.7.: Precision, recall, F_1, and average number of candidates for different thresholds for the normalized Levenshtein distance (development set)

Dist.	text-eval				OOV-eval			
	P	R	F_1	C	P	R	F_1	C
Lev(1)	**0.21**	0.52	**0.30**	6.95 ± 5.83	**0.20**	0.27	0.23	2.10 ± 3.22
$ed_{T,Re}$	0.20	0.59	0.29	8.49 ± 7.07	0.17	0.38	**0.24**	3.33 ± 5.34
Lev(2)	0.04	**0.82**	0.07	64.12 ± 55.31	0.04	**0.64**	0.07	25.86 ± 47.71

Table 7.8.: Precision, recall, F_1, and average number of candidates for the Levenshtein distance and $ed_{T,Re}$ (development set)

Using different distance thresholds again results in an improvement in recall with a smaller loss in precision than using a fixed distance > 1 for all types (cf. Table 7.7). 0.2 leads to the best F_1 value in the OOV setting and one of the best F_1 values in the text-eval setting. In both evaluation settings recall is improved without any loss in precision. We use this threshold for further experiments.

Table 7.8 shows the comparison between Levenshtein distances 1, 2, and the edit distance with the modifications presented above ($ed_{T,Re}$), i. e., the combination of allowing transpositions and repetitions and using a different distance based on the length of the type with a threshold of 0.2. Through these modifications of the Levenshtein distance, similar F_1 values are reached but with more emphasis on recall. This is exactly opposite to the results using character n-gram similarities where comparable F_1 values have been reached with more balance between precision and recall. For the following experiments, this combination is used to generate candidates ($ed_{T,Re}$).

One way to improve the precision is applying a filter to the generated candidates. So, before we present ways to improve the candidate generation when training data

for a specific dataset is available, we evaluate filtering the candidates using Brown clusters.

7.1.3. Context-Based Filter

The methods for generating spelling variant candidates described so far exploit only the fact that spelling variants often have similar surface forms. But, since spelling variants are variants of the same morphological word, they should also appear in similar contexts. Therefore, we look into contextual features as described in Section 4.8.2.

As simple context-based features, we use Brown clusters (Brown et al. 1992) using the implementation by Liang (2005). Since Brown clusters only need unannotated text in order to be trained (cf. Section 4.8), they are a simple way to apply a filter in an unsupervised setting. For obtaining the clusters, we use the unannotated background corpus described in Section 2.3.

Table 7.9 on the facing page shows the results for the Levenshtein distances 1, 2 and $\text{ed}_{T,Re}$ filtered with Brown clusters. The results show that using a small number of clusters works best. Furthermore, they show that the modified Levensthein distance leads to the best results regarding F_1.

The filter is only applied to types that are known from the background corpus used to train the Brown clusters, unknown types are left as spelling variants. This can explain the different results for the two settings. While applying the filter always leads to a better F_1 value in the text-eval setting, the F_1 value decreases in the OOV-eval setting except for using a Levenshtein distance of 2. As the OOV-eval setting tests on unknown words with respect to the training data, more often the words will not be included in the Brown clusters and therefore the filter is not applied. Consequently, the increase in precision is not as pronounced as in the text-eval setting. The OOV-eval setting is used to approximate the utility for NLP tasks, where usually training data is available that might also be usable to improve the spelling variant detection. Methods for this are presented in the next section.

7.2. Supervised Detection of Spelling Variants

In this section, we build on the presented methods for unsupervised detection of spelling variants in order to improve them using training data in form of pairs of spelling variants. We first present ways to use the training data for the candidate generation process. Afterwards, we present filters that use surface similarity as well as type-based and token-based context.

		text-eval				OOV-eval			
Dist.	Brown	P	R	F_1	C	P	R	F_1	C
Lev(1)	-	0.21	0.52	0.30	6.95 ± 5.83	0.20	0.27	0.23	2.10 ± 3.22
Lev(1)	25	0.53	0.32	0.39	1.67 ± 1.50	0.34	0.14	0.19	0.61 ± 1.11
Lev(1)	50	**0.59**	0.28	0.38	1.30 ± 1.31	**0.38**	0.12	0.18	0.48 ± 0.96
$ed_{T,Re}$	-	0.20	0.59	0.29	8.49 ± 7.07	0.17	0.38	**0.24**	3.33 ± 5.34
$ed_{T,Re}$	25	0.53	0.37	**0.44**	1.95 ± 1.79	0.31	0.18	0.23	0.89 ± 1.65
$ed_{T,Re}$	50	0.58	0.31	0.41	1.50 ± 1.51	0.33	0.15	0.21	0.71 ± 1.38
Lev(2)	-	0.04	**0.82**	0.07	64.12 ± 55.31	0.04	**0.64**	0.07	25.86 ± 47.71
Lev(2)	25	0.17	0.45	0.25	7.50 ± 6.67	0.09	0.31	0.14	5.20 ± 11.43
Lev(2)	50	0.18	0.37	0.24	5.72 ± 5.48	0.10	0.27	0.14	4.28 ± 10.15

Table 7.9.: Precision, recall, F_1, and average number of candidates for edit distances with Brown clusters (development set)

7.2.1. Improving Candidate Generation with Training Data

When the spelling variant relation for the training data is known, it can be used to improve the candidate generation step. In this section, we evaluate two ways to improve the candidate generation which we have described in Section 7.1. First, by a lexicon-based variant generation, i. e., by a simple lookup of known spelling variants. And, second, by applying simplification rules as described in Chapter 6 to reduce variation before generating spelling variant candidates.

Maybe the simplest approach for spelling variant detection with training data is a lexicon-based lookup of spelling variants. The lexicon lists all known variants for each known type. It can be extracted from the training data. For normalization, a similar memorization approach that normalizes each known type to its most frequent normalization from the training data is a strong baseline (Robertson and Goldwater 2018). Table 7.10 on the next page shows the results for the lookup approach. This approach cannot generalize to unknown types, therefore it only produces spelling variant candidates in the text-eval setting. In the OOV-eval setting where the evaluation is only done on unknown types, no candidates are generated. This leads to a precision of 1[31] and a recall of 0.

For the text-eval setting, the lexicon-based lookup is well suited for short and frequent words and thus complements the generation of candidates based on surface similarity. The GML personal pronoun *ji* ('you', 2PL.NOM) for example has the following spelling variants in the training data: *ghy*, *gi*, *gij*, *gv*, *gy* and *je*. The variant *ghy* will be especially hard to identify with an approach that is based on surface similarity since the two types *ji* and *ghy* share no character. Their Levenshtein distance is 3, which is greater than the length of the type *ji*.

31. As described in Chapter 4, we define precision to be 1 when there are no candidates generated as there are no falsely generated candidates.

text-eval				OOV-eval			
P	R	F_1	C	P	R	F_1	C
0.36	0.74	0.49	5.68 ± 7.90	1.00	0.00	0.00	0.00 ± 0.00

Table 7.10.: Precision, recall, F_1, and average number of candidates for the lookup approach (development set)

	text-eval				OOV-eval			
Dist.	P	R	F_1	C	P	R	F_1	C
Lev(1)	**0.22**	0.85	**0.35**	10.61 ± 10.84	**0.20**	0.27	0.23	2.10 ± 3.22
$ed_{T,Re}$	0.21	0.88	0.33	11.94 ± 11.91	0.17	0.38	**0.24**	3.33 ± 5.34
Lev(2)	0.04	**0.95**	0.08	65.43 ± 56.47	0.04	**0.64**	0.07	25.86 ± 47.71

Table 7.11.: Precision, recall, F_1, and average number of candidates for edit distances with lexicon-based lookup (development set)

Table 7.11 shows this lexicon-based lookup combined with the Levenshtein distance and $ed_{T,Re}$. Adding known spelling variants improves recall in the text-eval setting while keeping the same difference between the edit distances (cf. Table 7.8). The following approaches are combined with this lexicon-based lookup, so we show how well these approaches generalize to spelling variants not covered by a simple lookup.

A commonly used addition of the basic Levenshtein distance is to assign different costs to different substitutions or other edit operations (cf. Bollmann, 2012, and Pettersson, Megyesi, and Nivre, 2013, for normalization of historical texts, Gomes and Lopes, 2011, for cognate identification). These weights are trained on the specific data. Here, we present a similar approach by combining the rule-based simplification presented in Chapter 6 with an edit distance. For this, the rules are applied to all types of the dictionary and to the target type to map them to a simplified language. Then, all types whose simplified version is below the given distance threshold are seen as spelling variant candidates. Since differences between types that are covered by rules are removed before the edit distance is calculated, these differences do not contribute to the distance. Therefore, this approach is similar to assigning a cost of 0 to the specific edit operations encoded by the rules. Consequently, spelling variants with a higher distance than the specified threshold can be obtained. This is an efficient way to add weights to some edit-operations, especially when compared to using the weighted Levenshtein distance (cf. Bollmann 2013a, p. 24). While this, however, allows only for coarse weighting of the edit operations, note that the type-based filter applied afterwards, allows for a more fine-grained distinction of surface differences.

		text-eval				OOV-eval			
Freq.	Prec.	P	R	F_1	C	P	R	F_1	C
Lev(1)		**0.22**	0.85	**0.35**	10.61 ± 10.84	**0.20**	0.27	0.23	2.10 ± 3.22
$ed_{T,Re}$		0.21	0.88	0.33	11.94 ± 11.91	0.17	0.38	0.24	3.33 ± 5.34
Kol		0.19	0.89	0.31	13.17 ± 12.50	0.15	0.52	0.24	5.12 ± 9.06
Nie		0.01	**0.94**	0.02	236.95 ± 166.05	0.00	**0.88**	0.00	594.24 ± 674.78
10	0.70	0.15	0.92	0.25	17.45 ± 15.02	0.13	0.68	0.21	8.25 ± 13.78
10	0.75	0.15	0.89	0.26	16.24 ± 14.64	0.14	0.61	0.22	6.87 ± 11.86
10	0.80	0.18	0.89	0.30	13.64 ± 13.68	0.17	0.45	**0.25**	4.03 ± 6.70
10	0.85	0.20	0.88	0.33	12.12 ± 12.08	0.17	0.38	0.24	3.42 ± 5.59
10	0.90	0.21	0.88	0.33	11.94 ± 11.91	0.17	0.38	0.24	3.33 ± 5.34
20	0.70	0.16	0.92	0.27	15.86 ± 14.17	0.13	0.66	0.22	7.53 ± 12.18
20	0.75	0.17	0.91	0.29	14.96 ± 13.91	0.15	0.61	0.24	6.42 ± 10.81
20	0.80	0.19	0.89	0.31	13.26 ± 13.22	0.17	0.44	**0.25**	3.92 ± 6.37
20	0.85	0.20	0.88	0.33	12.05 ± 11.95	0.17	0.38	0.24	3.39 ± 5.45
20	0.90	0.21	0.88	0.33	11.94 ± 11.91	0.17	0.38	0.24	3.33 ± 5.34
30	0.70	0.16	0.92	0.27	15.85 ± 14.17	0.13	0.66	0.22	7.50 ± 12.16
30	0.75	0.17	0.91	0.29	14.95 ± 13.91	0.15	0.61	0.24	6.39 ± 10.79
30	0.80	0.19	0.89	0.31	13.24 ± 13.22	0.17	0.44	**0.25**	3.87 ± 6.33
30	0.85	0.20	0.88	0.33	12.04 ± 11.96	0.17	0.38	0.24	3.38 ± 5.43
30	0.90	0.21	0.88	0.33	11.94 ± 11.91	0.17	0.38	0.24	3.33 ± 5.34
40	0.70	0.16	0.92	0.28	15.72 ± 14.19	0.14	0.65	0.23	7.10 ± 11.54
40	0.75	0.17	0.91	0.29	14.93 ± 13.90	0.15	0.61	0.24	6.34 ± 10.65
40	0.80	0.19	0.89	0.31	13.24 ± 13.22	0.17	0.44	**0.25**	3.87 ± 6.33
40	0.85	0.20	0.88	0.33	12.04 ± 11.96	0.17	0.38	0.24	3.38 ± 5.43
40	0.90	0.21	0.88	0.33	11.94 ± 11.91	0.17	0.38	0.24	3.33 ± 5.34

Table 7.12.: Precision, recall, F_1, and average number of candidates for different rulesets combined with an edit distance (development set)

Table 7.12 on the preceding page shows the results for the different rulesets combined with $ed_{T,Re}$ from the previous section.[32] NIE, the ruleset based on the list of GML grapheme variants given in Niebaum (2000), shows the highest recall in both settings (0.94 / 0.88). However, the precision of the ruleset is extremely low, which makes the ruleset inappropriate for our purpose. KOL shows a better precision. The F_1 value is therefore a lot better. With the rulesets that are learned from the spelling variant pairs it is possible to get a better recall then KOL. And despite their simplicity, they do not show the same high loss in precision as the ruleset based on Niebaum (2000).

The utility of using simplification rules depends on the evaluation setting: in the text-eval setting the improvement in recall when compared with using the edit distance in combination with known spelling variants is only small. However, in the OOV-eval setting, there is a big improvement in recall. Still, the precision for all rulesets is quite low since it cannot be better than the precision of using only $ed_{T,Re}$. This is addressed by the filters presented in the next section. For the experiments with the filter, we use the learned rules with a frequency of at least 40 and a precision threshold of 0.75 in combination with $ed_{T,Re}$. This combination leads to one of the rulesets where the F_1 value is the same as with $ed_{T,Re}$ without using simplification (0.24) in the OOV-eval setting. However, recall is improved from 0.38 to 0.61 in this setting. In the text-eval setting, this ruleset leads to one of the results with the second-best recall (0.91). By combining this ruleset with $ed_{T,Re}$, we are able to improve recall from 0.88 (text-eval) / 0.38 (OOV-eval) to 0.91 / 0.61 with a loss in precision of just 0.04 / 0.02, while using a Levenshtein distance of 2 leads to a recall of 0.95 / 0.64 with a loss in precision of 0.17 and 0.13, respectively. We call this generator S+$ed_{T,Re}$ in the following experiments. However, as mentioned above, the precision is low (0.17 / 0.15). Therefore, we experiment with training a filter for the generated spelling variant candidates in the next section.

7.2.2. Type-Based Filter

In the previous section, we have evaluated methods how to efficiently generate spelling variant candidates for a given type from a given dictionary based on surface similarity. In this section, we look at approaches to filter these spelling variant candidates in order to remove unlikely candidates utilizing training data.

First, we look at applying a more fine-grained method to assess surface similarity. Variants of a weighted Levenshtein distance have been applied in tasks that are similar to spelling variant detection. Hauser and Schulz (2007) present methods for learning weights for detecting historical variants of modern words. Gomes and Lopes (2011) present a similarity measure for cognate detection. Such approaches

32. For this thesis, we reran the experiments with an updated version of the code, therefore the numbers differ from the results published in Barteld, Biemann, and Zinsmeister (2019).

usually employ quasimetrics, i. e., the used metrics are not necessarily symmetric since the weights learned for edit operations are learned in one direction – $i \rightarrow y$ may have a different cost than $y \rightarrow i$. This makes sense in the context of normalization and cognate detection as the comparisons made in these cases are directed as well, e. g., types from older stages of a language are compared to the modern variant. However, in the case of spelling variant detection, the weights for both directions should be equal because next to the pair {*ghe**c**omen*, ***k**omen*} from Example (4) on page 26 there does also exist the pair {*ghe**k**omen*, ***c**omen*} where the difference $c \leftrightarrow k$ appears in the opposite direction.

We present and compare two metrics with differing costs for edit operations. In the first metric, only the weights 0 and 1 exist but the context in which the substitutions appear are relevant. For the other metric, the weights take values between 0 and 1 but without context.

The first metric is an undirected version of the measure SPSim by Gomes and Lopes (2011). SPSim has been developed for cognate detection. This measure employs substitution patterns (SPs), i. e., segments of mismatches from the alignment of the types in the candidate pair with their left and right context. While the original version encodes the mismatched characters as ordered pair and is therefore a quasimetric, our version encodes the mismatches as unordered pair. Example (15) shows a pair of spelling variants and the corresponding undirected SP with a context of the length 2 denoted by the triple (left context, {pair of mismatched characters}, right context). The context is padded with *$* at the beginning and the end of the type.

(15) maria, marien
('ri', {'a', 'en'}, '$$')

The measure is trained on positive examples. When applying SPSim, SPs that appear in the training data get a cost of 0, otherwise their cost is the edit distance between the mismatched segments. Furthermore, the context is generalized, i. e., when a mismatch segment appears in the training data with at least two different contexts, the mismatch will always get a cost of 0 regardless of the context. Using this measure, pairs of types where all the changes are known get the maximal similarity of 1. This allows for improving the precision without losing on recall by setting a high threshold on the similarity for cognate detection.

Table 7.13 on the following page shows the results for using the undirected version of SPSim for identifying spelling variants. In this experiment, known spelling variants from the training data are added as spelling variant candidates. While there is an improvement in F-score from 0.33 without undirected SPSim to 0.39 in text-eval and from 0.24 to 0.31 in OOV-eval, the precision is still low (0.25 / 0.22). One reason for this is that the training data contains a lot of very generic substitutions that are learned by SPSim.

		text-eval				OOV-eval			
Dist.	Sim.	P	R	F_1	C	P	R	F_1	C
$ed_{T,Re}$	-	0.21	0.88	0.33	11.94 ± 11.91	0.17	0.38	0.24	3.33 ± 5.34
$ed_{T,Re}$	0.7	0.23	0.88	0.36	10.63 ± 10.79	0.18	0.38	0.24	3.20 ± 4.86
$ed_{T,Re}$	0.8	0.24	0.88	0.38	10.21 ± 10.61	0.19	0.37	0.25	3.03 ± 4.59
$ed_{T,Re}$	0.9	**0.25**	0.88	**0.39**	9.90 ± 10.64	**0.23**	0.37	0.28	2.46 ± 3.74
$ed_{T,Re}$	1.0	**0.25**	0.88	**0.39**	9.89 ± 10.64	**0.23**	0.37	0.29	2.41 ± 3.73
$S{+}ed_{T,Re}$	-	0.17	**0.91**	0.29	14.93 ± 13.90	0.15	**0.61**	0.24	6.34 ± 10.64
$S{+}ed_{T,Re}$	0.7	0.22	0.90	0.35	11.62 ± 11.25	0.17	0.58	0.26	5.33 ± 7.79
$S{+}ed_{T,Re}$	0.8	0.23	0.90	0.36	11.08 ± 10.99	0.18	0.57	0.27	4.93 ± 7.17
$S{+}ed_{T,Re}$	0.9	0.23	0.90	0.37	10.70 ± 11.02	0.21	0.56	0.31	4.03 ± 5.93
$S{+}ed_{T,Re}$	1.0	0.23	0.90	0.37	10.69 ± 11.02	0.22	0.56	**0.31**	3.98 ± 5.94

Table 7.13.: Precision, recall, F_1, and average number of candidates for edit distances with UndirSPSim (development set)

The following SPs are the SPs that are learned from our training data and involve a single 'a': (∅, {'a', 'o'}, ∅), (∅, {'a', 'u'}, ∅), (∅, {'a', 'e'}, ∅) and (∅, {'a', 'en'}, ∅). With this generic set of SPs, two types like *dach* ('(the) roof') and *doch* ('but') which differ only in *a* vs. any other vowel except for an *i* will have a similarity of 1. The pattern (∅, {'a', 'u'}, ∅) is learned from the two spelling variants {*sundighe*, *sandige*} and {*ghehat*, *ghehut*}. However, the first pair is likely to be an error in the original manuscript, the second example is an error in the gold annotation leading to a wrongly learned generalized SP. In order to avoid such overgeneralizations, we experiment with a metric where weights with values between 0 and 1 are allowed.

This second metric is similar to that proposed by Logačev, Goldschmidt, and Demske (2014). It is formulated as a probability for two types being spelling variants. Given an edit operation e, we estimate its probability of leading to a spelling variant, $P(e)$, by

$$P(e) = \frac{(\sum_{\{t_i,t_j\} \in tr(e)} 1_S(\{t_i, t_j\})) + 1}{|tr(e)| + 2} \tag{7.3}$$

$tr(e)$ is defined as $\{\{t_i, t_j\} | t_i \overset{e}{\leftrightarrow} t_j\}$, i. e., the set of all unordered pairs of types $\{t_i, t_j\}$ from the training data, such that t_i can be transformed into t_j or t_j into t_i by applying e. 1_S is the characteristic function of the spelling variant relation S. To avoid zero probabilities due to data sparsity, Laplace smoothing is used (Manning and Schütze 1999, Section 6.2.2). The $P(e)$ for an edit operation e that does not appear in the training data is set to 1 (instead of $\frac{1}{2}$) – thereby the probabilities capture negative evidence against the assumption that an edit operation leads to a spelling variant.

		text-eval				OOV-eval			
Dist.	Prob.	P	R	F_1	C	P	R	F_1	C
$ed_{T,Re}$	-	0.21	0.88	0.33	11.94 ± 11.91	0.17	0.38	0.24	3.33 ± 5.34
$ed_{T,Re}$	0.05	0.27	0.87	0.41	8.93 ± 9.93	0.28	0.34	0.31	1.90 ± 2.74
$ed_{T,Re}$	0.10	0.29	0.86	0.44	8.19 ± 9.44	0.32	0.31	0.32	1.49 ± 2.14
$ed_{T,Re}$	0.15	0.32	0.85	0.46	7.48 ± 8.93	0.35	0.27	0.31	1.17 ± 1.63
$ed_{T,Re}$	0.20	0.33	0.84	0.47	7.12 ± 8.67	0.37	0.25	0.30	1.03 ± 1.42
$ed_{T,Re}$	0.25	0.33	0.81	0.47	6.79 ± 8.55	0.42	0.22	0.29	0.80 ± 1.16
$ed_{T,Re}$	0.50	**0.36**	0.78	**0.49**	5.99 ± 8.16	**0.56**	0.12	0.20	0.34 ± 0.61
S+$ed_{T,Re}$	-	0.17	**0.91**	0.29	14.93 ± 13.90	0.15	**0.61**	0.24	6.34 ± 10.65
S+$ed_{T,Re}$	0.05	0.25	0.89	0.39	10.05 ± 10.52	0.25	0.52	0.34	3.23 ± 5.09
S+$ed_{T,Re}$	0.10	0.28	0.88	0.42	8.81 ± 9.71	0.29	0.46	**0.36**	2.38 ± 3.70
S+$ed_{T,Re}$	0.15	0.31	0.87	0.45	7.85 ± 9.07	0.33	0.39	**0.36**	1.82 ± 2.78
S+$ed_{T,Re}$	0.20	0.32	0.85	0.47	7.40 ± 8.74	0.36	0.36	**0.36**	1.51 ± 2.26
S+$ed_{T,Re}$	0.25	0.33	0.82	0.47	6.98 ± 8.58	0.39	0.31	0.35	1.20 ± 1.89
S+$ed_{T,Re}$	0.50	**0.36**	0.78	**0.49**	6.02 ± 8.16	0.51	0.14	0.22	0.42 ± 0.80
upper bound		0.34	1.00	0.51	8.08 ± 10.32	0.78	1.00	0.87	1.97 ± 3.96

Table 7.14.: Precision, recall, F_1, and average number of candidates for edit distances with edit probabilities (development set)

Given a pair of two types (t_1, t_2), we estimate the probability of (t_1, t_2) being spelling variants by the product of the probabilities of all the atomic edit operations that transform t_1 into t_2. Using the null hypothesis that the pairs are spelling variants, any set of possible spelling variants can be filtered by removing those for which the probability is below a given threshold.

Results for this filter are shown in Table 7.14. Known spelling variants from the training data are added as well. Both filtering approaches improve the F_1 values in both evaluation settings. However, the difference between the two variants of a weighted edit distance are obvious: if we allow only a cost of 1 or 0 this leads to a filter that filters out wrong spelling variant candidates with a high precision, i.e., it does only slightly affect the recall of the pipeline as most filtered types are actually not spelling variants. However, due to the conservative filtering, the precision of the pipeline gets only small improvements (cf. Table 7.13). With costs for edit operations ranging between 0 and 1, precision can be improved further, with respective losses in recall. Since the generator so far has a high recall with a low precision, this leads to a better balance and therefore improved F_1 values.

In the text-eval setting, the approach using the edit probabilities is close to the upper bound for a type-based approach with a recall of 1. We get this result by taking all spelling variants for each type: This *gold* approach reaches a F_1 value of 0.51 while the best F_1 value so far is 0.49. In the OOV-eval setting, however, the *gold* approach reaches an F_1 value of 0.87 where the best result so far is 0.36.

As a third approach, we follow Ciobanu and Dinu (2014) who apply a binary classifier to cognate recognition. We train a binary classifier on positive and negative examples for spelling variants to filter out overgenerated candidate pairs (cf. Section 4.3). To represent the similarities and differences in the strings, we experiment with undirected SPs as defined above as well as paired character n-grams around mismatches (Ciobanu and Dinu 2014), and all paired character n-grams (Ciobanu and Dinu 2015) extracted from the aligned sequences, see Example (16).

(16) maria, marien
2-grams: {$m, $m}, {ma, ma}, ...,
{ia, ie}, {a_, en}, {_$, n$}
2-grams(mis): {ia, ie}, {a_, en}, {_$, n$}

For the n-grams, we test all combinations of lengths in $\{1, 2, 3\}$. Similarly, for the SPs we use context sizes of $\{0, 1, 2\}$. Furthermore, we combine the n-grams with SPs.

One benefit of using a binary classifier is that it is straightforward to combine different features for filtering. We combine the surface features with contextual features. As contextual feature, we use the cosine similarity between dense vector representations (vec) obtained using PPMI-SVD calculated on the background corpus (cf. Section 4.8.2 and Section 2.3). As suggested by Levy, Goldberg, and Dagan (2015), we have tested different hyperparameters for the creation of the dense vectors: dimension ($\{125, 250, 375, 500\}$), context window ($\{2, 5\}$), and frequency threshold ($\{10, 25, 50, 75, 100\}$). We have also combined the contextual feature with the best performing surface features and the surface features with the best performing contextual feature.

The experiments presented below have been conducted with the ReN 0.1. Differing from the other experiments in this section, the evaluation of different hyperparameter settings has been done on the type-level and with the simple Levenshtein distance. Afterwards, we present the performance of the best settings with the ReN 0.3, evaluating it on the token level and with S+ed$_{T,Re}$ in the text-eval and OOV-eval settings.

For classification, an SVM is trained (cf. Section 4.3). We use an RBF kernel and train the model with the wrapper for LibSVM (Chang and Lin 2011) from Weka (Witten et al. 2017) doing a grid-search over the values $\{1, 2, \ldots, 5\}$ and $\{10^{-2}, \ldots, 10^{2}\}$ for the hyperparameters C and γ on the development set.

The classifier is trained on positive and negative examples. As positive examples, we use all the pairs of spelling variants appearing in the training data (1834 pairs). In order to obtain negative examples, we extract pairs of types with a Levenshtein distance of 1 and 2 that do not appear with the same annotation using only types that appear at least 10 times in the training data. The frequency threshold is used to

reduce the probability that the pair is actually a pair of spelling variants that—due to ambiguity of the types—does not occur as the same morphological word in the training data. We use all pairs with a Levenshtein distance of 1. In order to get the same number of negative and positive examples, we apply random undersampling (cf. Section 4.4) by adding only a random sample of pairs with a Levenshtein distance 2 to the negative examples. We apply the trained classifier to all candidate pairs generated by using the Levenshtein distances $1, 2$ and 3. Table 7.15 on the next page shows the best results for different feature combinations.

Overall, all the features lead to an improvement in F-score over the best F-score obtained using the Levenshtein distance (0.20) and the undirected SPSim (0.26). Combining the different types of surface features does not improve the results.

Using only n-grams around mismatches leads to better overall result than using all n-grams in terms of F-score (0.38 against 0.36), but using all n-grams leads to a slightly better recall (0.43 against 0.42). This is different from the result obtained by Ciobanu and Dinu (2015) for discriminating between cognates and borrowings. Both n-gram features lead to better results than using SPs, which lead to an F-score of 0.34. However, the differences between these three feature types are small and are not stable across different splits of the dataset.

Using only contextual features, the results are comparable to the results with surface features, regarding the F-score (0.36). However, this F-score results from a higher recall and a lower precision. A context size of 2, a small frequency threshold (10) and the dimensions 500 and 375 lead to the best results on the dataset.

Combining surface and contextual features results in the best F-score (0.42) using this approach. However, in experiments with vectors obtained from a selection of the background corpus (739,576 tokens), adding the contextual features has led to no improvement over using only surface features. This is important regarding true low-resource settings where even unlabeled data that can be used as a background corpus is scarce.

Regarding the generation method, the best F-scores are obtained using a Levenshtein distance of 1. The increase in recall obtainable by adding further candidate pairs with a higher Levenshtein distance corresponds to a larger drop in precision.

We now present the results for the SVM filter on the ReN 0.3. As generator, we use the different edit distances described above. Furthermore, we present an alternative way for handling the training data. Above the classifier is trained on a training set that consists of generated candidates, which are labeled whether they are an actual spelling variant pair or not. This dataset has two characteristics that are relevant for training the classifier: Firstly, as pointed out above, the set of instances labeled as negative contains falsely labeled instances. Therefore, this might best be approached as PU learning (Li and Liu 2005). Secondly, the dataset is highly imbalanced.

When using a standard SVM as above, we address these two issues of the training set by including only pairs of types where each type appear at least 10 times in the

Lev	Filter			R	P	F_1	C
1	-	-	-	0.58	0.12	0.20	1.85 ± 2.42
2	-	-	-	0.88	0.02	0.04	15.99 ± 20.06
1	SPSim (0.9)	-	-	0.48	0.18	0.26	1.11 ± 1.51
2	SPSim (0.9)	-	-	0.65	0.09	0.15	2.93 ± 3.92
	SVM						
	Features	C	γ				
1	n-gram(mis): 1, 2, 3	2	10^{-1}	0.42	0.34	0.38	0.62 ± 0.86
1	n-gram: 1, 2, 3	3	10^{-1}	0.43	0.31	0.36	0.68 ± 0.91
1	SP: 0, 1	2	10^{0}	0.37	0.31	0.34	0.60 ± 0.84
1	vec: 500, 2, 10	4	10^{-1}	0.52	0.28	0.36	0.83 ± 1.08
1	vec: 500, 2, 50, n-gram(mis): 1, 2, 3	2	10^{-1}	0.47	0.37	0.42	0.62 ± 0.87
1	vec: 375, 2, 25, n-gram(mis): 1, 2, 3	2	10^{-1}	0.47	0.38	0.42	0.62 ± 0.86
2	vec: 500, 2, 10, n-gram: 3, n-gram(mis): 1, SP: 2	4	10^{-1}	0.58	0.17	0.26	1.48 ± 1.81

Table 7.15.: Precision, recall, F_1, and average number of candidates for the binary classification approach ReN 0.1

data to get a set of reliable negative instances and by random undersampling to tackle the imbalance. We compare this approach with a bagging classifier. Bagging addresses both the imbalanced data and the PU learning (Galar et al. 2012; Mordelet and Vert 2014). As a base classifier, we use an SVM as well (cf. Section 4.4). For comparing the two approaches, we use the best hyperparameter ($C = 2$, $\gamma = 10^{-1}$) and feature combinations (vec, 375, 2, 25, n-gram(mis): 1, 2, 3) from the experiments above.

Table 7.16 on the facing page shows the results. For comparison, we repeat the results of using only lookup generation as well as S+$\text{ed}_{T,Re}$ with edit probabilities as filter and edit probabilities (EP) combined with Brown clusters. We have also included the results for the perfect type-based spelling variant detection as an upper bound.

When comparing the SVM with the BSVM, the results are mixed: In the text-eval setting, the BSVM performs comparably to the SVM. But in the OOV-eval setting it outperforms the SVM in terms of F_1 value for all generators. This is achieved by a better balance between precision and recall.

As for candidate generation in the text-eval setting, a Levenshtein distance of 1 and $\text{ed}_{T,Re}$ show comparable results in terms of F_1 value. However, this F_1 value is close to the upper bound for type-based spelling variant detection (with a recall of 1). This underlines the need for token-based spelling variant detection. In the OOV-eval setting using $\text{ed}_{T,Re}$ leads to better results.

For type-based spelling variant detection, we conclude that the combination of $\text{ed}_{T,Re}$ with a BSVM filter is comparable to the results with a Levenshtein distance of 1 and an SVM but with a better recall in the text-eval setting. In the OOV-eval

Dist.	Filter	text-eval P	R	F_1	C	OOV-eval P	R	F_1	C
lookup		**0.36**	0.74	0.49	5.68 ± 7.90	1.00	0.00	0.00	0.00 ± 0.00
S+$ed_{T,Re}$	EP (0.5)	**0.36**	0.78	0.49	6.02 ± 8.16	**0.51**	0.14	0.22	0.42 ± 0.80
S+$ed_{T,Re}$	EP (0.5) + Brown	0.67	0.41	0.51	1.72 ± 1.91	0.57	0.08	0.14	0.22 ± 0.55
Lev(1)	–	0.22	0.85	0.35	10.61 ± 10.84	0.20	0.27	0.23	2.10 ± 3.22
Lev(1)	SVM	**0.36**	0.80	**0.50**	6.17 ± 7.91	0.48	0.20	0.29	0.64 ± 0.91
Lev(1)	BSVM	0.35	0.81	0.49	6.49 ± 8.16	**0.51**	0.22	0.31	0.67 ± 0.94
S+$ed_{T,Re}$	–	0.17	0.91	0.29	14.84 ± 13.96	0.15	0.58	0.24	5.96 ± 10.72
S+$ed_{T,Re}$	SVM	0.33	0.87	0.47	7.44 ± 8.45	0.23	0.54	0.32	3.64 ± 6.62
S+$ed_{T,Re}$	BSVM	0.34	0.87	0.49	7.20 ± 8.54	0.41	0.48	**0.44**	1.81 ± 2.87
Lev(2)	–	0.04	**0.95**	0.08	65.43 ± 56.47	0.04	**0.64**	0.07	25.86 ± 47.71
Lev(2)	SVM	0.17	0.92	0.29	14.99 ± 14.38	0.10	0.63	0.18	9.37 ± 15.29
Lev(2)	BSVM	0.17	0.91	0.29	14.51 ± 15.78	0.21	0.60	0.31	4.46 ± 6.97
upper bound		0.34	1.00	0.51	8.08 ± 10.32	0.78	1.00	0.87	1.97 ± 3.96

Table 7.16.: Precision, recall, F_1, and average number of candidates for the Support Vector Machine filter (development set)

setting, this combination—$ed_{T,Re}$ with a BSVM filter—leads to the best results overall. In the next section, we look at a token-based filter for improving these results.

7.2.3. Token-Based Filter

The approaches that we have presented above work on the type level, i. e., candidates are generated and filtered regardless of the specific context of the target word. However, as noted above, two types might be spelling variants only in specific contexts. Therefore, we apply a filter that uses the context to filter out spelling variant candidates. For this, we use a neural network that is similar to the CNN architecture proposed by Kim (2014) for sentence classification on the spelling variant candidate and n context types before and after (cf. Section 4.3). The network is depicted in Fig. 7.2a on the next page. The words are represented using the same (context) embeddings as with the type-based classifier. Differing from Kim (2014), the embeddings are not fine-tuned in the training. Furthermore, we concatenate a second (surface) embedding to the context embedding. This is similar to Chakrabarty, Pandit, and Garain (2017), but instead of using a long short-term memory layer to obtain the surface (what they call syntactic) embedding, we create it using convolutional filters of length 2 and 3 that are applied to the (padded) sequence of characters for each word. This part of the network is depicted in Fig. 7.2b. The surface representation is not pre-trained.

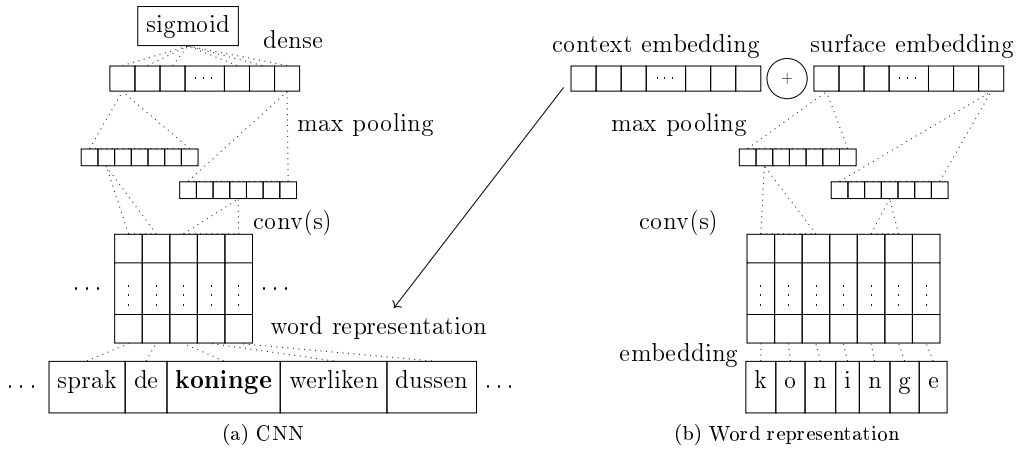


Figure 7.2.: Architecture of the Convolutional Neural Network used as token-based filter

For the sequence of words, we apply convolutional filters of length 2 up to the context length + 1. In this way, the longest filters see each of the context words to one side together with the target word. We apply 50 filters of each length for the surface embedding as well as to extract features from the word representations.

For the training data, we generate all candidates for each of the tokens in the training set. Each pair of token and candidate is labeled whether it is a spelling variant pair or not. Again, the data is imbalanced. To account for this, the network is trained on batches of 20 positive and 20 negative pairs sampled randomly from the training data. We train for 10 epochs, where one epoch consists of training on $\frac{n_{pos}}{20}$ batches with n_{pos} denoting the number of positive training examples. The parameters are updated via backpropagation using Adam (cf. Section 4.8.3). We use ReLU for non-linearity (cf. Section 4.3.2).

Table 7.17 on the facing page shows the results for different context sizes. Furthermore, the table contains results for configurations using only the context embeddings or the surface embeddings, respectively. This shows that both representations help to filter out candidates that are not spelling variants. However, the results for the text-eval and the OOV-eval setting diverge: For the text-eval setting, applying the token-based filter improves the results with all settings. A context size of 1 or 2 using both context embeddings and surface embeddings to represent the words gives the best results. For the OOV-eval setting, however, applying the token-based filter leads to a drop in F_1 value since the loss in recall is not matched by the gain in precision. In the next section, we look at possible reasons for this.

		text-eval				OOV-eval			
Embedd	Size	P	R	F_1	C	P	R	F_1	C
S+ed$_{T,Re}$-BSVM		0.34	**0.87**	0.49	7.20 ± 8.54	0.41	**0.48**	**0.44**	1.91 ± 2.93
both	1	**0.56**	0.63	**0.59**	3.16 ± 3.48	0.66	0.21	0.31	0.30 ± 0.77
both	2	0.53	0.65	0.58	3.45 ± 4.04	0.62	0.22	0.33	0.35 ± 0.85
both	3	0.53	0.62	0.57	3.24 ± 4.02	0.62	0.19	0.29	0.29 ± 0.79
cont.	1	0.53	0.65	0.58	3.44 ± 3.94	**0.69**	0.19	0.29	0.26 ± 0.72
cont.	2	0.48	0.63	0.55	3.63 ± 4.62	0.62	0.20	0.30	0.31 ± 0.84
cont.	3	0.49	0.56	0.52	3.20 ± 4.47	0.64	0.17	0.27	0.26 ± 0.71
surf.	1	0.48	0.69	0.56	4.03 ± 4.61	0.62	0.22	0.33	0.35 ± 0.86
surf.	2	0.45	0.72	0.55	4.45 ± 5.24	0.59	0.23	0.33	0.38 ± 0.93
surf.	3	0.47	0.68	0.56	4.02 ± 4.81	0.59	0.20	0.30	0.34 ± 0.83

Table 7.17.: Precision, recall, F_1, and average number of candidates for the Convolutional Neural Network filter (development set)

		text-eval				OOV-eval			
Dist.	Filter	P	R	F_1	C	P	R	F_1	C
Lookup		0.29	0.69	0.41	5.62 ± 7.72	1.00	0.00	0.00	0.00 ± 0.00
Lev(1)	–	0.16	0.85	0.27	12.17 ± 11.41	0.11	0.24	0.16	3.91 ± 5.55
S+ed$_{T,Re}$	–	0.14	**0.87**	0.25	14.23 ± 12.55	0.09	**0.39**	0.14	8.35 ± 12.74
Lev(1)	SVM	0.25	0.83	0.39	7.72 ± 8.53	0.22	0.24	0.23	2.06 ± 2.58
S+ed$_{T,Re}$	BSVM	0.28	0.82	0.41	6.95 ± 8.08	0.27	0.33	**0.30**	2.28 ± 2.88
+ CNN (both, 1)		**0.47**	0.57	**0.52**	2.80 ± 3.37	**0.41**	0.18	0.25	0.80 ± 1.34
+ CNN (surf., 3)		0.36	0.64	0.46	4.12 ± 5.29	0.35	0.19	0.25	1.01 ± 1.59

Table 7.18.: Precision, Recall, F_1, and average number of candidates for different pipelines (test set)

7.3. Results and Error Analysis

In this section, we give results for the best supervised pipelines on the test set, present an error analysis and discuss some ideas how to improve the spelling variant detection.

Table 7.18 presents results for selected approaches on the test set: the simple lookup, Levenshtein distance 1 and S+ed$_{T,Re}$ without a filter and with SVM, BSVM and CNN filter. These pipelines show the same behavior as on the development set: With type-level generation and filtering, on the one hand, we are able to improve the recall compared to a simple lookup approach. In the text-eval setting, the loss in precision is low, leading to a comparable F_1 value. In the OOV-eval setting, where a lookup is not available, both recall and precision are improved compared

		text-eval				OOV-eval			
Dist.	Filter	P	R	F_1	C	P	R	F_1	C
Lookup		**0.54**	0.03	0.05	0.03 ± 0.19	1.00	0.00	0.00	0.00 ± 0.00
Lev(1)	–	0.21	0.51	0.30	1.29 ± 1.40	0.14	0.25	0.18	1.55 ± 2.35
S+ed$_{T,Re}$	–	0.17	**0.65**	0.28	1.97 ± 2.29	0.10	**0.49**	0.17	4.26 ± 6.59
Lev(1)	SVM	0.29	0.51	0.37	0.93 ± 1.06	0.21	0.25	0.23	1.04 ± 1.51
S+ed$_{T,Re}$	BSVM	0.45	0.59	**0.51**	0.69 ± 0.91	0.34	0.44	**0.38**	1.18 ± 1.78
+ CNN (both, 1)		**0.54**	0.33	0.41	0.32 ± 0.64	**0.43**	0.25	0.31	0.51 ± 1.09
+ CNN (surf., 3)		0.51	0.41	0.45	0.43 ± 0.76	0.39	0.28	0.32	0.64 ± 1.23

Table 7.19.: Precision, Recall, F_1, and average number of candidates for different pipelines on types with frequency lower 10 (test set)

to candidate generation using the Levenshtein distance. In both settings S+ed$_{T,Re}$ with a BSVM filter leads to an improvement over the standard Levenshtein distance with a distance of 1 and an SVM filter.

Token-level filtering, on the other hand, is only helpful (in terms of F_1 value) in the text-eval setting. In the OOV-eval setting, the loss in recall is too high compared to the gain in precision. The reason for this seems to be the high number of types that are infrequent in the test set for OOV-eval. This is evident in Table 7.19, which shows the results calculated without the types that appear more than 10 times in the whole dataset. Here, the token-based filter leads to a high drop in recall in both settings resulting in a lower F_1. One reason for this might be that the context embeddings for such infrequent terms are either not existent or not very reliable. This is consistent with the fact that using only surface embeddings as features for the filter leads to better results with infrequent types. The gain in precision that the use of context embeddings brings is not enough for infrequent types.

The results also indicate that—especially for the OOV-eval setting where a spelling variant lookup cannot be used—the recall of the generator still is a bottleneck for the pipeline: Even though we have been able to improve the recall in the OOV-eval setting from 0.24 with Levenshtein distance 1 to 0.39 using S+ed$_{T,Re}$, the recall is still low.

When generating types in the OOV-eval setting—i. e., without the possibility of using a lookup—and generating types from the test data for the test data, the generator using S+ed$_{T,Re}$ misses spelling variants for 615 types. We have sampled 100 types from these and have looked for patterns regarding the variants that were not generated.

What we observe from this sample is that most of the missed spelling variants have a Levenshtein distance of 2, combining two variation patterns that are infrequent in the data and/or have a low precision. Therefore, they are not captured by the simplification rules. One example is the type *ober* ('over'), for which the two

variants *aver* and *auer* are missed. The difference ['a', 'o'] appears in 129 spelling variant pairs in the training data, however the precision of this conflation set on the training set is only 0.33. The differences ['b', 'v'] and ['b', 'u'] each only appear in one spelling variant pair in the training set and their precision is well below 0.1.

Interestingly, many of the missing spelling variants involve prefixation and suffixation. One example of a prefix that often appears in spelling variant pairs is *g(h)e*: *richten*, *gerichten* ('(to) sentence'); *coft*, *ghecoft* ('(to) buy'); *nemet*, *genomen* ('(to) take'). While in modern Standard German this is an inflectional prefix, in historical variants of German (as well as Low German), the usage of this prefix often counts as a mere variation. An example for a suffix from the data is *en*—or better the lack of the suffix *en*. The variation stems from the fact that this inflectional suffix can be missing, e. g., in the pairs *hunderden*, *hundert* ('hundred') and *uint*, *vinden* ('(to) find').

In this chapter, we have evaluated pipelines for spelling variant detection intrinsically using two evaluation settings that try to measure their utility for two tasks, i. e., IR or simply searching in texts and statistical NLP. In the next chapter, we examine different approaches for improving the performance of NLP tools for texts with spelling variation. Some of these approaches use spelling variant detection. Since the experiments in this chapter have shown better results without token-based filter in the OOV-eval setting, we apply a type-based filter in these experiments.

Chapter 8.

Dealing with Spelling Variation for Natural Language Processing

In this chapter, we present experiments with two standard NLP tasks on non-standard data. We train and apply existing tools for statistical POS tagging and lemmatization on GMH and GML texts. For the experiments, we use a low-resource setting, using the parts of the ReM and the ReN described in Section 2.4 for training and testing. In Section 8.1, we describe the tools. Since these tools have been developed with standard data in mind, we look at methods to adapt them to data with spelling variation in Section 8.2. In Section 8.3 and Section 8.4, we present experiments with different adaptation methods. Section 8.5 concludes this chapter with an overview of the results.

8.1. Tools for Part-of-Speech Tagging and Lemmatization

For the experiments with POS tagging, we use Marmot (Müller, Schmid, and Schütze 2013), a CRF tagger, as the main tool. For lemmatization, we use Lemming (Müller et al. 2015), a log-linear model in which Edit Trees (ETs) are applied to reduce the number of possible labels. The underlying ML algorithms are described in Chapter 4. To make sure that these tools are good starting points for adaptation, we compare them with other tools as baselines. For this, we use all the tools with their default settings.

For POS tagging, we compare Marmot with HunPos (Halácsy, Kornai, and Oravecz 2007) and RFTagger (Schmid and Laws 2008). Especially RFTagger and the closely related TreeTagger (Schmid 1995) are often used in DH contexts, e. g., by Echelmeyer, Reiter, and Schulz (2017) who train the TreeTagger on GMH. Both, HunPos and RFTagger, are similar in that they are HMM taggers. But they differ in some details, which we describe briefly in the next paragraphs.

HunPos is a trigram HMM tagger based on ideas found in TnT (Brants 2000). In addition to using trigrams for the transition probabilities, tag bigrams are used for the emission probabilities as well. OOV words are handled by learning probabilities

over classes of infrequent words with the same suffixes and using these probabilities for calculating the emission probabilities of OOV words.

RFTagger allows to choose the number of previous tags considered for the transition probabilities. Similar to HunPos, we use the two preceding tags. Emission probabilities are only based on the current tag. For the emission probabilities of OOV words, the RFTagger uses classes of known words with the same suffix as well. The main difference between the two taggers is that the RFTagger uses decision trees for the transition probabilities. This method has been introduced by Schmid (1994). The decision trees use binary tests on the identity of the preceding tags to assemble the tag n-grams in different classes. The probabilities are then calculated for the different classes in the training data. While the description of the classes can use the identity of all tags in the n-gram, it can also incorporate the identity of only one or two of the three tags. Therefore, with limited training data, small classes of exact tag n-grams that would be used in the standard MLE estimation of the probabilities are avoided. This is useful in low-resource scenarios.

Another feature of the RFTagger—that it shares with Marmot—is that it decomposes POS tags with morphological descriptions in order to estimate transition probabilities. However, since we only use POS tags without morphological description in the experiments, this feature is not used.

Marmot differs from HunPos and RFTagger in that it employs a CRF instead of an HMM. It uses a pruned CRF as described in Section 4.7.2 to allow for effective training of higher-order CRFs. In our experiments, we use a second-order CRF. So Marmot will use tag trigrams as well as HunPos and RFTagger.

Since Marmot is a CRF tagger, words are represented by features extracted by feature functions. Marmot uses the following features as described by Müller, Schmid, and Schütze (2013):

> [T]he current, preceding and succeeding words as unigrams and bigrams and for rare words prefixes and suffixes up to length 10, and the occurrence of capital characters, digits and special characters. We define a rare word as a word with training set frequency ≤ 10. We concatenate every feature with the POS [...] tag [...].
>
> [...] We also use an additional binary feature, which indicates whether the current word has been seen with the current tag or – if the word is rare – whether the tag is in a set of open tag classes.
>
> (p. 325)

In the POS tagging experiments using additional unlabeled data, we use word representations as additional features for Marmot as described in Müller and Schütze (2015).

In the lemmatization experiments, we always assume that the data contains POS tags and use the gold POS tags when training and when applying the tools. As baseline, we use a simple POS dependent lookup, i. e., known combinations of type and POS tag are lemmatized with the corresponding lemma that appears most frequently in the training data. OOV words are lemmatized with the type itself. This method is implemented in a script which is contained in the RFTagger release and is used to supply the lemmatization when using the RFTagger.[33]

The main approach to lemmatization that we use is Lemming (Müller et al. 2015). As described in Section 4.6, Lemming combines a log-linear model with a support function to exclude unlikely lemmas from the calculation. The support function applies ETs that are learned individually for the different POS tags from the training data. For a given word, lemma candidates are generated with the learned ETs. The word that is to be lemmatized, w, is represented for the log-linear model of Lemming with the following features as described by Müller et al. (2015):

> We use the following three *edit tree features* of Chrupała (2008). (i) The edit tree e. (ii) The pair $\langle e, w \rangle$. This feature is crucial for the model to memorize irregular forms, e.g., the lemma of *was* is *be*. (iii) For each form affix (of maximum length 10): its conjunction with e. These features are useful in learning orthographic and phonological regularities, e.g., the lemma of *signalling* is *signal*, not *signall*.
>
> We define the following *alignment features*. Similar to Toutanova and Cherry (2009) (TC), we define an alignment between w and l. [...] [T]he alignment of umgeschaut-umschauen is: u-u, m-m, ge-ϵ, s-s, c-c, h-h, a-a, u-u, t-en. Each alignment pair constitutes a feature in our model. [...]
>
> We define two simple *lemma features*. (i) We use the lemma itself as a feature, allowing us to learn which lemmata are common in the language. (ii) Prefixes and suffixes of the lemma (of maximum length 10). This feature allows us to learn that the typical endings of Spanish verbs are *ir*, *er*, *ar*.
>
> [...] For each feature listed previously, we create a conjunction with the POS [...].
>
> (pp. 2269–2270)

In the following section, we present existing approaches to adapt tools for POS tagging and lemmatization to historical data.

33. The script is called `lemma-lookup.perl` and is found in the folder `cmd` of the RFTagger release (`https://www.cis.uni-muenchen.de/~schmid/tools/RFTagger/data/RFTagger.zip`, last visited October 4, 2021).

8.2. Strategies for Dealing with Spelling Variation

Spelling variation negatively affects tools for automatically annotating or working with natural language data. In this section, we present existing work using different approaches for automatically annotating historical texts. On page 7 in Section 1.1, we have identified two general ways for dealing with spelling variation: (1) adapting the respective tool to the non-standard data and (2) using automatic procedures for simplification and spelling variant detection as presented in Chapter 6 and Chapter 7 to remove the variation before applying a tool. In Section 8.3 and Section 8.4, we present case studies to see how these two general ways can be used to mitigate the effects of spelling variation on the basic NLP tasks POS tagging and lemmatization.

While spelling variation is an important aspect when working with historical data like texts from GML, there are other problems that have to be faced as well. One of these is the low-resource nature of historical texts: Annotated training data is sparse and also unlabeled data for supporting domain adaption methods and semi-supervised approaches is not as readily available as it is for most contemporary languages. Hence, we concentrate on low-resource settings, i. e., training a POS tagger/ lemmatizer and the tools to deal with spelling variation on a small amount of data. However, for POS tagging we also look at a third option to improve the performance: the usage of available external resources, mainly additional unlabeled texts.

In the following part, we give an overview of the usage of these three strategies in the existing literature:

1) Tool adaptation. Tool adaptation can be done by including specific features to the models that allow them to learn spelling variation patterns. One example for this is the solution used by Koleva et al. (2017), who present experiments on GML texts with a memory-based learner and a CRF tagger with different sets of features that include, among others, prefix and suffix n-grams. The authors conclude that a tagger using such features handles spelling variation itself while applying the rule-based simplification Kol (cf. Chapter 6), which the authors refer to as normalization, only leads to marginal improvements of the tagging accuracy. For lemmatization, Kestemont, Daelemans, and de Pauw (2010) use a memory-based learner for Middle Dutch texts. For OOV words, they produce known words that have a high probability of being a spelling variant and use their lemmatization in the training data to predict the lemma of the unknown word.

In Section 8.3.2, we present experiments with adding additional features to Marmot, a CRF-based POS tagger. In Section 8.4.2, we present experiments with Lemming, a statistical lemmatizer where the labeling approach for lemmatization (cf. Section 4.6) is adapted to deal with spelling variation by incorporating lemmas of likely spelling variants in the support function.

2) Reduction of spelling variation. This approach deals with spelling variation not within the used tools, but in a preprocessing step. Dipper (2010) performs tagging experiments on a corpus of GMH texts—an early version of a part of the ReM—in three different versions (cf. Section 2.2): A strict *transcription* that captures many peculiarities of the script, for example superposed characters, a *simplified* version, where most of these peculiarities are removed, e. g., superposed characters are brought into sequence, and a *normalized* version where spelling variation is reduced by mapping the words to an artificial GMH standard that is traditionally used by philologists. Dipper's experiments show that training and tagging lead to better results with more variation removed: Normalized data is better than simplified data, which in turn is better than using the strict transcription. Rule-based simplification is also employed in other works, e. g., by Adesam and Bouma (2016) for POS and morphological tagging of Old Swedish.

One limitation for these approaches is that the simplification as well as the normalization has been done with manually created rules or semi-automatically and—to our knowledge—there is no work that explores the utility of simplification with automatically learned rules or automatic normalization in the sense of mapping words to a standard form when training a tagger except the work of Goot, Plank, and Nissim (2017), who present experiments on English tweets. They come to the conclusion that while normalization improves tagging accuracy, using word representations obtained from a large amount of unlabeled data gives larger improvements. Combining both only leads to small improvements over using only word representations. However, this might be different in low-resource settings since there is fewer data available for obtaining good word representations.

Logačev, Goldschmidt, and Demske (2014) as well as Barteld, Schröder, and Zinsmeister (2015) improve the results for POS tagging by detecting likely pairs of spelling variants and substituting unknown words with a known word that is a spelling variant. One way to learn spelling variation patterns for POS tagging and/ or lemmatization is using distant or weak supervision, by not learning patterns from exact spelling variant pairs but leveraging the given annotation to approximate spelling variant pairs (Kestemont, Daelemans, and de Pauw 2010; van Halteren and Rem 2013; Logačev, Goldschmidt, and Demske 2014).

In Section 8.3.3 and Section 8.4.3, we present experiments with rule-based simplification, normalization and spelling variant detection for reducing variation when training and applying a POS tagger/ lemmatizer.

3) Usage of external resources. The usage of external resources mainly aims at overcoming the lack of training data. External resources can be as simple as additional unlabeled texts. But they can also be existing tools. In the case that a closely related standardized variant exists, normalization to this variant can be used to apply tools that exist for this variant to the non-standard data achieving reasonable results (Bollmann 2013b; Tjong Kim Sang et al. 2017). However, the

datasets in our experiments differ substantially more from contemporary Standard German than e. g., Early New High German that has been used in the experiments of Bollmann (2013b). While there have also been experiments to combine normalization with domain adaptation (Yang and Eisenstein 2016), we do not aim to apply a POS tagger developed for contemporary German to the data after normalization. However, instead of directly using the annotation provided by the tools for the standardized language, these annotations can also be added as a feature when training an annotation tool on the non-standard data.

In Section 8.3.4, we present experiments with applying a POS tagger trained on normalized GMH text on automatically normalized texts and on using word embeddings learned on additional unlabeled data.

8.3. Experiments with Part-of-Speech Tagging

In this section, we present our experiments with POS tagging on the ReN and the ReM, using the tools described in Section 8.1. In order to overcome the problems that spelling variation poses to these tagging approaches, we experiment with the different techniques described in Section 8.2. After establishing baselines by training the taggers with their default settings in Section 8.3.1, we compare the effects of using an adapted feature set for the CRF tagger Marmot (Section 8.3.2), reducing the spelling variation before training and applying the tagger (Section 8.3.3) and making use of additional information from external resources (Section 8.3.4).

8.3.1. Baselines

We train the taggers described in Section 8.1 with their standard settings on the data to establish some baselines.[34] Table 8.1 on the next page shows the results for the tagger on the *strict* version of the datasets.[35] Marmot leads to the best results across both datasets. Significant improvements over the tagger below are marked with '*'.[36]

In the following sections, we look into how to improve these results with 1) tool adaptation, 2) reduction of spelling variation, and 3) the usage of external resources.

34. HunPos needs a token consisting only of digits in the training data in order to run successfully. Since our ReM training data does not contain such a token, we have added one to the data.

35. Tagging results are given as accuracies in percentage points (cf. Section 4.2).

36. For all experiments, we use McNemar's test (McNemar 1947) with continuity correction (Edwards 1948) and a significance level of 0.05 (cf. Section 4.2).

Tagger	ReM	ReN
Marmot	**84.05***	**85.44**
HunPos	82.32	84.78*
RFTagger	81.68	83.95

Table 8.1.: POS tagging results for the baselines (development set). '*' marks a significant improvement over the tagger below.

8.3.2. Tool Adaptation

Spelling variation increases the risk for a tagger to encounter OOV words, however spelling variants themselves will often show a large character overlap (cf. Section 4.8). Therefore, taking subword information into account seems promising for tagging historical texts as Dipper (2010) has pointed out. In this section, we present experiments on tagging the *strict* version of the texts using character n-gram features.

All the baseline taggers already use character n-gram information in some way. Both RFTagger and HunPos use suffix information to estimate tag probabilities for unknown words, the maximum length of the suffixes is set to 7 (RFTagger) and 10 (HunPos). Marmot uses prefixes and suffixes up to a given length as features for rare words, length 10 in the standard settings. To get an insight about the impact of subword information, we set the maximum affix length for Marmot to $\{4, 7, 10, 13, 16\}$. Next to prefix and suffix features, Marmot also allows integrating infix features, which is disabled by default. The length of the infixes is also governed by the maximum length parameter. The results for the different lengths with and without the usage of infixes are given in Table 8.2 on the following page.

While the best settings differ for the datasets, there are two general points: 1) Without infix features, the numbers show that the increase in the length from 4 to 7 leads to a high increase in accuracy while higher affix lengths only change the accuracy marginally, so the default value of 10 is a reasonable choice for our datasets as well. 2) Adding infix features leads to improvements for both datasets, however they are only significant in the case of the ReM. The maximum length of character n-grams does not lead to significant differences in the accuracy when using infix features. We use length 4, which leads to the best results for the ReM, for further experiments.

In the default settings of Marmot, rare words are defined as words with a training data frequency of up to 10. For Modern German, Marmot has been trained on the first 40,474 sentences of the TIGER treebank (Müller, Schmid, and Schütze 2013). This is a substantially larger dataset than the 12,000 tokens used here. Hence,

Max. length	Infix	ReM	ReN
16	+	84.83*	85.91
13	+	84.83*	85.91
10	+	84.84*	**85.94**
7	+	84.88*	85.86
4	+	**84.93***	85.89
16	-	84.07	**85.51**
13	-	84.12	**85.51**
10	-	84.05	85.44
7	-	**84.15**	85.38
4	-	83.94	84.88#

Table 8.2.: POS tagging results with character n-gram features (development set). '*' marks a significant improvement over the standard settings (Max. length 10, without infixes), '#' marks a significant loss in performance.

using the same frequency threshold leads to an effectively lower threshold for rare words. Still, it might be useful to include character n-grams for more words in order to enable the tagger to learn spelling variation patterns. We experiment with setting the maximum frequency for rare words to $\{5, 10, 15, 20, 25, 30, 35, 40, \infty\}$. The results in Table 8.3 on the next page show that 10 is a reasonable default for our data as well: Higher thresholds up to 30 seem to give better results but the improvements are not significant. We set the frequency for rare words to 30 for further experiments as this gives the best results for both datasets. Adding the features to all words (∞) does not improve the tagging accuracy.

Summing up, we can state that the tagging accuracy of Marmot can be improved for both datasets by adding infix features and using higher frequency thresholds for rare words than for Modern German. The utility of these features seems to depend on the amount of spelling variation in the data as the differences are higher for the ReM, which has a higher proportion of variation (see Table 2.1 on page 24). In the following experiments, we call Marmot with the original feature set Marmot-*orig* and with the tweaked feature set—using prefixes, suffixes and infixes of length up to 4 for words with a frequency up to 30 in the training data—Marmot-*hist*. Table 8.4 on the next page gives a comparison of both on the test sets. While Marmot-*hist* is significantly better than Marmot-*orig* for the ReM, it turns out that Marmot-*orig* is better than Marmot-*hist* for the ReN. However, on this dataset the difference is not significant.

Freq.	ReM	ReN
∞	84.81	86.09
40	84.89	85.91
35	84.93	86.06
30	**85.06**	**86.16**
25	85.03	86.12
20	84.86	86.04
15	85.04	**86.16**
10	84.93	85.89
5	84.96	85.61

Table 8.3.: POS tagging results with different frequency thresholds for rare words (development set).

Tagger	ReM	ReN
Marmot-hist	**85.86***	83.71
Marmot-orig	84.28	**84.15**

Table 8.4.: POS tagging results for Marmot with original feature set (Marmot-*orig*) and a feature set tweaked for historical texts with spelling variation (Marmot-*hist*) (test set). '*' marks a significant improvement of Marmot-*hist* over Marmot-*orig*.

Tagger	ReM	ReN
Marmot-hist	**85.98***	**86.12**
Marmot-orig	85.55	85.81

Table 8.5.: POS tagging results with simplification (development set). '*' marks a significant improvement over tagging the strict version with Marmot-*hist*.

8.3.3. Reduction of Spelling Variation

In the previous section, we have looked into character n-gram features to enable the tagger to better deal with spelling variation. An alternative approach is to preprocess the texts and remove spelling variation before applying the tagger. In this section, we look into different ways to achieve this and how they interact with the enhanced feature set of Marmot-*hist*.

A simple way to reduce spelling variation is simplification (cf. Chapter 6). One example for GMH would be to substitute the long s (ſ) with a regular round s, removing the variation between these two characters. For experiments with this approach, we use the *simple* version of the texts (cf. Section 2.2). Table 8.5 contains the tagging accuracy when training and tagging on *simple*. For the ReM, the accuracy improves significantly by nearly 1% point with Marmot-*hist*. Marmot-*orig* even improves further, rendering the differences between both variants of the tagger as insignificant, indicating that the infix features actually capture spelling variation and are less useful for datasets with less variation. Regarding the ReN, simplification only improves the tagging results for Marmot-*orig*.

To further investigate the impact of using infix features, we again experiment with setting the maximum frequency for rare words to $\{5, 10, 15, 20, 25, 30, 35, 40, \infty\}$. Table 8.6 on the facing page shows that infix features help to improve the tagging accuracy, however, with less variation it is better to add them to fewer words: For the ReN, thresholds of 35 and 40 show a significant drop in performance compared to a threshold of 10.

Dipper (2010) has already shown that making use of normalization leads to even further improvements regarding GMH. Normalization abstracts dialectal differences away. An example for this is the GMH word *maiſters* ('master'), its simplified version is *maisters* with the long s changed to a round s. Its normalized version again is *meisters*, which abstracts away from a general variation between *ai* and *ei* in GMH. In the semi-automatically normalized version of the ReM training data[37]

37. For some types, e. g., punctuation, no normalization is given. In this case, we use the simple version.

Freq.	ReM	ReN
∞	85.85	86.29
40	85.82	85.92#
35	85.92	85.99#
30	85.98	86.12
25	85.78	86.14
20	85.95	86.19
15	85.95	86.30
10	**86.02**	**86.49**
5	85.92	86.14

Table 8.6.: POS tagging results with different frequency thresholds for rare words and simplification (development set). '#' marks a significant loss in performance compared to the max. frequency setting of 10.

only 5.56% of the morphological words are realized by more than one type, which is a substantial reduction of spelling variation (cf. Table 2.1 on page 24).[38]

For experiments on automatic normalization of the ReM texts we rely on cSMTiser,[39] a normalization tool using character-level machine translation that is one of the best performing systems in the CLIN27 Shared Task (Tjong Kim Sang et al. 2017). The techniques have been described by Ljubešić et al. (2016) and Scherrer and Erjavec (2016). We train a normalization model with cSMTiser on the training set using only tokens. The model normalizes 86.23% tokens correctly on the development set.[40]

Table 8.7 on the following page shows that tagging accuracy is significantly higher when tagging the gold normalized version than the strict version for both feature sets. With automatic normalization the improvement in tagging accuracy is lower. For the automatically normalized data—as well as for the gold normalization—, using Marmot-*orig* leads to better results than using Marmot-*hist*. Although the difference is not significant, this shows again that the feature engineering has been tailored to texts with spelling variation.

As an alternative to simplification and normalization, we experiment with the pipeline approach to spelling-variant detection presented in Chapter 7. We use spelling-variant detection to substitute OOV words with their detected spelling

38. There is no normalized version of the ReN available.

39. `https://github.com/clarinsi/csmtiser`, last visited October 4, 2021.

40. Training a model to normalize whole sentences, thereby including token context into the normalization, has led to worse results with the small amount of training data.

Tagger	Normalization	ReM
Marmot-hist	gold	89.51*
Marmot-orig		**89.71***
Marmot-hist	automatic	85.08
Marmot-orig		**85.39**

Table 8.7.: POS tagging results for normalized Middle High German (development set). '*' marks a significant improvement over tagging the strict version with Marmot-*hist*.

	ReM		ReN	
Tagger	spellvar	$\text{spellvar}_{\text{pos}}$	spellvar	$\text{spellvar}_{\text{pos}}$
Marmot-hist	86.71*	93.27*	87.30*	91.88*
Marmot-orig	86.20*	92.71*	86.67	91.50*

Table 8.8.: POS tagging results with spelling variant substitution – upper bounds (development set). '*' marks a significant improvement over tagging the strict version with Marmot-*hist*.

	ReM			ReN	
Tagger	spellvar	$\text{spellvar}_{\text{norm}}$	$\text{spellvar}_{\text{pos}}$	spellvar	$\text{spellvar}_{\text{pos}}$
Marmot-hist	**85.69***	**85.69***	85.46*	86.37	**86.39**
Marmot-orig	84.78	84.76	84.71	85.91	85.62

Table 8.9.: POS tagging results with spelling variant substitution using automatic spelling-variant detection (development set). '*' marks a significant improvement over tagging the strict version with Marmot-*hist*.

variant if possible. To get an impression of how much improvement is possible with this technique, we calculate upper bounds by substituting OOV words with the most frequent spelling variant from the training data if one exists. Spelling variants are defined by having the same POS tag, morphology and lemma (spellvar) (cf. Chapter 3). Since not only correct substitutions will help the tagger but also substitutions with another known word that just has the same POS or distribution (Barteld, Schröder, and Zinsmeister 2015; Kolachina, Riedl, and Biemann 2017), we also substitute OOV words with the most frequent known type that has the same POS ($\text{spellvar}_{\text{pos}}$). This can be seen as a weakly supervised learning scenario for spelling-variant detection as described in Section 4.1.2. Table 8.8 on the preceding page shows the results for these upper bounds. In contrast to the experiments with normalization above, this time Marmot-*hist* performs better than Marmot-*orig*. This can be explained by the fact that the variation is not reduced in the training data. With spelling variant substitution, we achieve an improvement that is larger than the improvement obtained with automatic normalization but lower than the improvement using the gold normalization. Applying the not-so-strict definition of spelling variation, leads to substantial gains, which we attribute to the fact that this excludes unseen words in the task of POS tagging.

For automatic spelling-variant detection, we use a type-based approach as these led to better results in the OOV-eval setting than the token-based filter (cf. Section 7.2.3). We use a simple generator without the fine tuning of the edit distance for GML that we have presented in Section 7.1 to obtain results for a simple version. So, for all unknown types in the development data, we select all known types with a Levenshtein distance (Levenshtein 1966) of 1 as candidates and filter this set using a Bagging-SVM (cf. Section 7.2.2). We only use aligned character n-gram features for the SVM as we assume no existing background corpus to learn word representations on. From the spelling variants identified by this method, we choose the most frequent type (measured on the training data) for substitution of the OOV word.

Training pairs for the spelling-variant detection can be obtained in different ways. We test three settings: 1) using lemma, POS and morphology (spellvar), 2) using the normalization ($\text{spellvar}_{\text{norm}}$), and 3) using only POS ($\text{spellvar}_{\text{pos}}$). While 1) and 2) lead to reliable training data, option 3), the weakly supervised setting (cf. Section 4.1.2), leads to more noisy training data. However, this option is especially interesting in a low-resource setting as no additional annotation or data is needed.

Table 8.9 on the facing page shows that substituting automatically detected spelling variants for OOV words results in improvements of the tagging accuracy that are comparable to the improvements obtained with automatic normalization. As with the upper-bound experiments, Marmot-*hist* gets better results. Using only the POS annotation gives results that are only slightly worse than the ones with more reliable training data. In the case of the ReN, they are even slightly better than those.

	ReM					ReN			
Tagger	strict	norm	simple	spellvar	$\text{spellvar}_{\text{pos}}$	strict	simple	spellvar	$\text{spellvar}_{\text{pos}}$
Marmot-hist	*85.86*	87.02*	**87.15***	86.47*	85.99	83.71	84.70	84.38	84.01
Marmot-orig	84.28	86.87*	85.88	85.14#	84.91#	*84.15*	84.45	**84.81***	84.65*

Table 8.10.: POS tagging results with spelling variant reduction (test set). '*' marks a significant improvement over tagging the strict version with Marmot-*hist* for the ReM and Marmot-*orig* for the ReN, '#' marks a significant loss in accuracy.

Table 8.10 shows a comparison of automatic normalization, simplification and spelling variant substitution trained with lemma, POS and morphology and only POS on the test set. All the methods for spelling reduction improve the respective tagger. For the ReM, the combination of Marmot-*hist* and simplification leads to the best results. For the ReN, this combination leads to the second best result. We obtain the best results with the combination of Marmot-*orig* and spelling-variant detection using lemma, POS and morphology.

8.3.4. External Resources

In the previous sections, we have limited ourselves to using approximately 12,000 tokens as training data, for some experiments exploiting additional annotations like normalization or lemma. In this section, we experiment with using other resources in addition to the training data. These fall into two categories: word representations learned on additional unlabeled data (cf. Section 4.8.2) and—in the case of the ReM—an existing tagger for normalized GMH.

For additional unlabeled data, we use the texts from the corpora that are not used in the POS tagging experiments. In the case of the ReM these are 392 texts, 2,437,090 tokens, in the case of the ReN only 44 texts, 259,192 tokens. We try three different ways to obtain word representations from these datasets: PPMI-SVD, SGNS and fastText (cf. Section 4.8.2). For PPMI-SVD and SGNS, we use hyperwords (Levy, Goldberg, and Dagan 2015). We run the tools with standard settings on the additional texts and use the resulting word representations as an additional feature for Marmot (Müller et al. 2015). While fastText allows obtaining representations for OOV words by summing the representations of character n-grams, we do not use this feature as Marmot needs to be trained with a fixed set of word representations. However, learning representations for words and character n-grams simultaneously is —according to Bojanowski et al. (2017)—beneficial for small datasets and improves the representations for rare words. It might help in the case of spelling variation as well.

Table 8.11 on the facing page shows the results for tagging the strict version using the different representations. Marmot-*hist* leads to the best results since

Tagger	ReM PPMI-SVD	SGNS	fastText	ReN PPMI-SVD	SGNS	fastText
Marmot-hist	85.26	85.22	**85.95***	86.21	86.32	**86.37**
Marmot-orig	84.30#	84.25#	85.24	85.52#	85.56#	85.82

Table 8.11.: POS tagging results with word embedding feature (development set).'*' marks a significant improvement over tagging the strict version with Marmot-*hist*, '#' marks a significant loss in accuracy.

Tagger	ReM PPMI-SVD	SGNS	fastText	ReN PPMI-SVD	SGNS	fastText
Marmot-hist	85.69*	85.67*	**86.20***	86.35	**86.55**	86.25
Marmot-orig	85.08	85.04	85.78*	85.89	85.97	85.97

Table 8.12.: POS tagging results with word embedding feature and spelling-variant detection (development set).'*' marks a significant improvement over tagging the strict version with Marmot-*hist*.

we do not reduce variation in the training data. All three embedding approaches lead to improvements that are comparable with the results obtained by automatic normalization and spelling-variant detection. fastText gives the best results for both datasets, but only for the ReM the improvement is significant over tagging the strict version with Marmot-*hist*. The reason for the small improvement might be the rather small amount of unlabeled data for the ReN. Tuning the hyperparameters of the embedding methods, e. g., reducing the number of dimensions, might yield further improvements.

We also combine the embedding feature with spelling-variant detection trained with lemma, POS and morphology, see Table 8.12. By combining both, again there is a small improvement. For the ReM, this improvement is significant for all types of embeddings. For the ReN, there is no significant improvement over Marmot-*hist*, indicating that the combination of infix features and word representations already covers a lot of the spelling variation.

For GMH, there exists an independently created model for the TreeTagger (Echelmeyer, Reiter, and Schulz 2017). The tagset used to train the model is coarse-grained, consisting of only 18 tags. We use the tags predicted by this tagger as an additional feature for Marmot. We expect this TreeTagger model to work better on the normalized version of ReM than on the other versions, because the TreeTagger model has been trained on data from the *Mittelhochdeutsche Begriffsdaten-*

Tagger	TreeTagger strict	norm
Marmot-hist	85.21	88.16*
Marmot-orig	84.83	87.83*

Table 8.13.: POS tagging results for Middle High German with additional part-of-speech tags (development set).'*' marks a significant improvement over tagging the strict version with Marmot-*hist*.

Tagger	Normalization	ReM
Marmot-orig	automatic	87.35*
	gold	**90.24**

Table 8.14.: POS tagging results for normalized Middle High German with additional part-of-speech tags (development set).'*' marks a significant improvement over tagging without the additional part of speech (POS) feature.

bank (Middle High German Conceptual Database)[41], which contains texts from editions that consist of normalized GMH. This is confirmed by the results of training Marmot-*orig* and Marmot-*hist* on the strict version of the texts adding as additional feature the tag produced by the TreeTagger model a) on the strict version, or b) on the normalized version, see Table 8.13.

When using the TreeTagger tags generated on the strict version, there is no significant improvement compared to tagging the texts without the additional feature. We conclude that we need normalized input to get good improvements from the tagger in this particular setting. Hence, we also train our tagger on the normalized data. As Marmot-*orig* performs better for the normalized data, we only use the original features. Table 8.14 shows the results when training the tagger on normalized data with added tags as predicted by the TreeTagger model. Adding the tags as features leads to significant improvements only for automatic normalization.

8.4. Experiments with Lemmatization

In this section, we present our experiments with lemmatization of GMH and GML (cf. Section 2.4). We look at the two main approaches of dealing with spelling

41. `http://mhdbdb.sbg.ac.at/`, last visited October 4, 2021.

	ReM	ReN
Lookup (simple)	**75.68***	80.69#
Lemming (strict)	74.84*	**83.55***
Lookup (strict)	72.31	80.26

Table 8.15.: Lemmatization results for the baselines (development set). '*' marks a significant improvement over the tagger below. '#' marks a significant loss in accuracy.

variation described in Section 8.2: (i) tool adaptation (Section 8.4.2), and (ii) the reduction of spelling variation in the texts (Section 8.4.3). We use the tools for lemmatization described in Section 8.1. For all the experiments, we use gold POS tags for the training, development and test data.

8.4.1. Baselines

As a baseline, we use a simple lookup approach to lemmatization, using the most frequent lemma for a given pair of word form and POS tag that appears in the training data. We compare this with Lemming using the default settings. The results for both approaches are shown in Table 8.15.

As can be seen, Lemming outperforms the simple lookup approach on both datasets when using the strict version. Therefore, it is a good starting point for adaptation to non-standard texts. To show the effect of spelling variation, the table also contains the results for the lookup approach using the simplified tokens. For the ReN, where simplification reduces the amount of spelling variation only from 18.11% to 16.81% (cf. Table 2.1 on page 24), Lemming outperforms both lookup approaches. For the ReM, where the spelling variation is reduced from 22.81% to 18.12% (cf. Table 2.1 on page 24), using the simple version with the lookup-based lemmatization leads to a better result than applying Lemming to the strict version of the texts. However, as we see in the experiments with reducing spelling variation, Lemming leads to better results on this version of the texts as well (cf. Table 8.19 on page 130).

8.4.2. Tool Adaptation

Barteld, Schröder, and Zinsmeister (2016) have introduced two additions to Lemming: LCs as a means of generating lemma candidates (as opposed to the original ETs) and a generator that uses known lemmas from predicted spelling variants in addition to the candidates generated by the ETs or LCs. The authors have

introduced LCs mainly as a means to deal with morphological word-internal modifications. Word-internal modifications, however, are also introduced by spelling variation. Hence, we use LCs for our adaptation of Lemming. Table 8.16 shows the results of lemmatizing the ReM and the ReN using Lemming with ETs (Lemming-*ET*) and with LCs (Lemming-*LC*). For both, the ReM and the ReN, using LCs leads to slightly better results. However, the differences are both not statistically significant.

Tagger	ReM	ReN
Lemming-LC	**75.05**	**83.73**
Lemming-ET	74.84	83.55

Table 8.16.: Lemmatization results for Lemming with Edit Trees and Lexical Correspondences (development set).

For the second addition, we use an adapted support function for Lemming (cf. Section 4.6) to not only use lemmas generated by ETs or LCs as potential labels, but also lemmas that appear with spelling variants of the target word in the training data. For spelling variant detection, we do not apply the methods of Barteld, Schröder, and Zinsmeister (2016). Instead, we use the pipeline approach presented in Chapter 7 with the same settings as in the POS tagging experiments (cf. Section 8.3.3). We train the spelling variant detection using a weak supervision approach (cf. Section 4.1.2) with POS tag and lemma to define spelling variants. As Lemming uses POS tags as features, lemmas and POS tags are both available in the training data. Table 8.17 on the facing page shows the results. Adding information about spelling variation into the support function improves the lemmatization significantly. Again the differences between ETs and LCs are not statistically significant. For the ReM, when using LCs the results are slightly better, for the ReN they are slightly worse.

For further experiments, we distinguish between Lemming-*orig*, using ETs to generate lemma candidates, and Lemming-*hist*, using LCs and spelling variants to generate lemma candidates. Table 8.18 on the next page shows the results for both variants of Lemming on the test set. These results show that Lemming-*hist* outperforms Lemming-*orig* on both datasets.

8.4.3. Reduction of Spelling Variation

As an alternative—or addition—to adapting the annotation tool, we present experiments with removing spelling variation in the data. Table 8.19 on page 130 gives the results with the simple version of the texts. For comparison, we repeat the results for the lookup approach. The results show that the spelling variation

Tagger	ReM	ReN
Lemming-LC	**79.63***	86.60*
Lemming-ET	79.47*	**86.64***

Table 8.17.: Lemmatization results for Lemming with Edit Trees and Lexical Correspondences and spelling variant generator (development set). '*' marks a significant improvement over the respective Lemming version without spelling variant handling.

Tagger	ReM	ReN
Lemming-hist	**78.79***	**86.09***
Lemming-orig	74.10	82.48

Table 8.18.: Lemmatization results for Lemming with Edit Trees (Lemming-*orig*) and Lemming with Lexical Correspondences and spelling variant generation (Lemming-*hist*) (test set). '*' marks a significant improvement of Lemming-*hist* over Lemming-*orig*.

handling in Lemming-*hist* is capable of handling spelling variation such that using Lemming-*orig* with the simple versions leads to worse results than using Lemming-*hist* with the strict version (cf. Table 8.17). For the ReM, combining Lemming-*hist* with the simple version improves lemmatization. For the ReN there is no significant difference between using Lemming-*hist* with the simple and with the strict version. The outcome of the two approaches is similar to the outcome that we have already seen with POS tagging (cf. Table 8.5 on page 120).

For the ReM, we also use the normalized version of the texts. For automatic normalization, we apply the same cSMTiser model as for POS tagging (cf. Section 8.3.3). Table 8.20 on the next page shows the results. Normalization improves the results compared to tagging the strict version—irrespective of using Lemming-*orig* or Lemming-*hist*. While using Lemming-*hist* still improves the results significantly for normalized data, the differences are less pronounced than when lemmatizing the strict or simple version. This shows that the additions introduced in Lemming-*hist* are less necessary for data with less spelling variation.

As a last experiment with reducing spelling variation in the data, we substitute OOV words with spelling variants. Table 8.21 on page 131 shows the results for an upper-bound using gold spelling variants for the substitution with both the exact spelling variants and the spelling variants approximated with POS tag and lemma.

Tagger	ReM	ReN
Lemming-hist	**82.60***	**86.92**
Lemming-orig	78.83#	83.93#
Lookup	75.68#	80.69#

Table 8.19.: Lemmatization results for Lemming-*orig* and Lemming-*hist* on the simple version (development set). '*' marks a significant improvement over tagging the strict version with Lemming-*hist*, '#' marks a significant loss in accuracy.

Tagger	Normalization	ReM
Lemming-hist	gold	**86.99***
Lemming-orig		86.38*
Lemming-hist	automatic	**81.76***
Lemming-orig		81.28*

Table 8.20.: Lemmatization results for normalized Middle High German (development set). '*' marks a significant improvement over tagging the strict version with Lemming-*hist*.

Tagger	ReM spellvar	ReM $\text{spellvar}_{\text{poslemma}}$	ReN spellvar	ReN $\text{spellvar}_{\text{poslemma}}$
Lemming-hist	82.83*	85.87*	88.84*	90.89*
Lemming-orig	81.38*	85.60*	87.52*	90.82*

Table 8.21.: Lemmatization results with spelling variant substitution – upper-bounds (development set). '*' marks a significant improvement over tagging the strict version with Lemming-*hist*.

Tagger	ReM spellvar	ReM $\text{spellvar}_{\text{poslemma}}$	ReN spellvar	ReN $\text{spellvar}_{\text{poslemma}}$
Lemming-hist	79.77	79.95*	86.82	86.82
Lemming-orig	78.73#	79.77	85.79#	86.75

Table 8.22.: Lemmatization results with spelling variant substitution using automatic spelling variant detection (development set). '*' marks a significant improvement over tagging the strict version with Lemming-*hist*, '#' marks a significant loss in accuracy.

This improves the results in all settings. Lemming-*hist* always leads to better results, however, the differences are again less pronounced.

Table 8.22 shows the results for using automatic spelling variant detection with the same approach as used for the adapted support function of Lemming (cf. Section 8.4.2). We compare using the actual spelling variants for training the pipeline (*spellvar*) and using weak supervision with only POS tag and lemma for finding spelling variant pairs ($spellvar_{poslemma}$). Spelling variant substitution improves the performance of Lemming-*orig* in all settings. Yet, in comparison with Lemming-*hist* on the strict version, the performance is slightly worse for *spellvar* and slightly better, but not statistically significant, for $spellvar_{poslemma}$. Combining the usage of Lemming-*hist* with spelling variant substitution does improve the accuracy slightly for the ReM. For the ReN the accuracy is slightly better as well but not statistically significant. However, it is not surprising that substituting predicted spelling variants does only slightly improve the results since Lemming-*hist* already uses the predicted spelling variants for lemma candidate generation.

Table 8.23 on the following page shows the results for the different ways of reducing spelling variation on the test set. Except for normalization, using Lemming-*hist* on the strict version outperforms using Lemming-*orig* with spelling variant

Tagger	ReM strict	norm	simple	spellvar	spellvar$_{poslemma}$	ReN strict	simple	spellvar	spellvar$_{poslemma}$
Lemming-hist	*78.79*	82.02*	81.08*	79.00*	78.98*	*86.09*	86.62*	86.20	86.17
Lemming-orig	74.10#	81.26*	77.76#	78.03#	78.75	82.48#	83.34#	85.28#	85.81

Table 8.23.: Lemmatization results with spelling variant reduction (test set). '*' marks a significant improvement over tagging the strict version with Lemming-*hist*, '#' marks a significant loss in accuracy.

reduction. Combining Lemming-*hist* with spelling variant reduction always leads to improvements that are statistically significant for the ReM but not for the ReN.

8.5. Results

In this chapter, we have investigated training a POS tagger and a lemmatizer for historical German in a low-resource setting—that is training with only about 12,000 tokens—and we have looked into different ways to deal with spelling variation.

We have found that spelling variation heavily affects the performance of the automatic annotation. Therefore, removing all or most of the spelling variation has a big impact on the accuracy: Using gold normalization and Marmot without any adaptations, POS tagging accuracy improves from 84.05% to 89.71% on the development set for the ReM. For lemmatization, the accuracy goes up from 74.84% to 86.38% on the same dataset.

However, when not using gold normalization but automatic normalization with a character-based SMT model trained on the training dataset, the tagging accuracy drops to 85.39%. For lemmatization, it drops to 81.28%. While these are both statistically significant improvements, they still need training data for the automatic normalization. We have evaluated alternative ways to deal with spelling variation that result in similar improvements without the requirement of training a normalizer.

Firstly, we looked into adapting the tools. For POS tagging, we have evaluated adding features that enable the model to learn spelling variation patterns. By adding all character n-grams instead of only prefixes and suffixes as features for rare words and adapting the rare word threshold, we have been able to improve tagging accuracy to 85.06% for the ReM, which is close to the results with automatic normalization. For the ReN, the dataset with less variation, the improvement in accuracy using the adapted feature set is not significant on the development set. On the test set, the original feature set even leads to better results. This shows that when training a POS tagger on data with spelling variation, it pays off to use specialized features instead of simply using the available feature set developed for standardized languages. However, infix features—as added for the experiments in this thesis—only help for data with a certain amount of spelling variation.

For lemmatization, we have evaluated changing the support function for the lemmatizer. The main adaptation for spelling variation is to use lemmas that appear in the training data with spelling variants of OOV words. For the detection of spelling variants we have applied the pipeline presented in Chapter 7. By training it with weak supervision, using only the data available when training the lemmatizer, we avoid the need for additional data as with normalization. For the ReM the lemmatization accuracy reaches 79.63%. For the ReN it is 86.60%.

So, for both tasks, we come close to the results obtained with automatic normalization by adapting the tools to better deal with spelling variation.

Secondly, we have evaluated alternatives for normalization to reduce spelling variation. By applying rule-based simplification in combination with specialized features, POS tagging accuracy has improved to 86.02% for the ReM and to 86.49% for the ReN. The lemmatization accuracy has improved to 82.60% for the ReM and 86.92% for the ReN. So, by creating a small set of rewrite rules to reduce variation, it is possible to improve tagging accuracy more than with automatic normalization in a low-resource setting.

As another alternative to normalization, we have evaluated the substitution of OOV words with automatically detected spelling variants using the pipeline from Chapter 7. It has been trained using only the information available for the given task, i. e., POS tags for POS tagging and POS tags and lemmas for lemmatization. In combination with the specialized feature set, we have reached an accuracy of 85.46% for the ReM, which is similar to the accuracy reached with automatic normalization, and 86.39% for the ReN. For lemmatization, this approach only leads to minor improvements over simply using the adapted tools: 79.95% (vs. 79.63%) for the ReM and 86.82% (vs. 86.60%) for the ReN. This is easily explainable as the adapted lemmatizer already uses the detected spelling variants in the support function.

Overall, adapting the tools for dealing with spelling variation along with simplification and substitution of spelling variants do not lead to the same amount of improvements in tagging accuracy as using gold normalization does, but they can be performed automatically with less effort. Compared to the automatic normalizer trained on 12,000 tokens, we have been able to reach the same or comparable accuracy without needing any other training data than the data for the POS tagger and the lemmatizer. Thus, even without any additional data or resources, these approaches can be used to improve accuracy.

We have shown that the approaches to spelling variation without a reference to a standard language presented in Chapter 6 and Chapter 7 can be used to improve automatic annotation with POS tags and lemmas in a low-resource setting. Therefore, they are a viable alternative to normalization.

For future work, we see two main research questions:

1. How much training data is necessary to create normalization models that lead to better tagging results than with the alternative methods presented here?
2. How do the methods explored for this thesis in a low-resource setting behave with more training data?

The second questions is especially interesting since for historical German the ReM and the ReN are now fully available, which allows using more training data. In this context, the methods used here would have to be compared to recent neural network approaches that use character-based recurrent neural networks to create word representations for POS tagging and additionally sequence to sequence approaches for lemmatization. Two examples that have been specifically designed for historical texts with spelling variation are RNNTagger (Schmid 2019), which has been tested on POS tagging the ReM, and PIE (Manjavacas, Kádár, and Kestemont 2019), which has been tested on lemmatizing the ReN. However, for both approaches substantially more training data has been used than in our experiments: for training RNNTagger over 2,000,000 tokens from the ReM and for PIE more than 355,000 tokens from the ReN. It will be interesting to see how these approaches work in the low-resource setting that we have used in our experiments on the one hand and how our approaches compare with more training data on the other hand.

Chapter 9.

Conclusion and Further Work

In this thesis, we have presented approaches to deal with spelling variation, which is often encountered in non-standard texts, without resorting to a given standard. Since treating spelling variation as a deviation from a standard—as it is done in most work on spelling variation—limits the focus of the developed methods to non-standard texts that are closely related to existing standards, approaching spelling variation independently of a standard is important. With the methods presented and evaluated in this thesis, which do not presuppose an existing standard, it is easier to target languages farther away from any given standard or without training data for normalization.

To show the similarities and differences between the approaches for spelling variation, we have given formal definitions of spelling variation and ways to approach it like normalization and spelling variant detection in Chapter 3. Consistent with our basic aim, the given definition and measure for quantification of spelling variation in a corpus do not rely on a given standard. Spelling variation is defined and measured as consistency of the spellings in the corpus.

In Chapter 6 and Chapter 7, we have tested variants of two basic approaches for handling spelling variation on GML texts. For this, we have used two evaluation settings that model two basic use cases: searching in non-standard texts as well as training and applying NLP tools on these non-standard texts (cf. Chapter 5). The two approaches that we have evaluated are *simplification* and *spelling variant detection*.

Simplification is a common way to reduce spelling variation found in many works on non-standard texts. Often this is done by applying hand-crafted rules. In this thesis, we have presented a simple way to define a ruleset by listing corresponding characters. We have also presented an approach to automatically learn such rules from given spelling variants.

For *spelling variant detection*, we have evaluated different string similarity metrics and have presented a fine-tuned edit distance. Combined with Brown clusters for contextual filtering, this is a good starting point in unsupervised settings. For supervised settings, we have presented different ways to learn weights for the difference in the surface form and the contexts, getting the best results training an SVM

for filtering candidates. Since the context of a token is also relevant for the spelling variant relation, we have also presented a token-based filter using a CNN.

Implementations of all the presented approaches are available in our python package *SpellvarDetection* (see Appendix B), so that they are readily available for research with non-standard texts and further research into *simplification* and *spelling variant detection*.

In Chapter 8, we have looked at the impact of spelling variation for POS tagging and lemmatization. Our experiments have shown that by reducing the spelling variation using simplification or normalization, the accuracy of statistical tools for automatic annotation can be improved. As an alternative to simplification and normalization, we have also used spelling variant detection to substitute OOV words with likely spelling variants. Furthermore, we have presented experiments in which we have adapted the tools to better handle spelling variation.

After giving a short overview of the main results from our experiments in Section 9.1, we outline directions for future work in Section 9.2.

9.1. Main Findings

Regarding simplification, we have found that creating a simple ruleset from corresponding characters as described in Chapter 6 can lead to similar results as performing simplification with complex handcrafted rules. Therefore, defining equivalent characters for the non-standard language and automatically create simplification rules from these characters is a good starting point for applying simplification. When examples for spelling variants are available, equivalent characters can also be extracted from the data. With the resulting rulesets, we have been able to outperform two manually created rulesets.

Regarding spelling variant detection, we have found that the Levenshtein distance, while being coarse-grained, is still usable as a generator for spelling variant candidates. While we have been able to improve the results with a fine-tuned edit distance for GML, still the coarse-grained basic Levenshtein distance is an easy to use starting point as a generator.

We have achieved the best results filtering the spelling variant candidates with an SVM with contextual features (cosine distance between word embeddings) and surface features (character n-grams around mismatches). While both surface and contextual features improve the results, we have also achieved good results using only contextual features. This is especially relevant in low-resource settings where no background corpus is available to learn word representations from. In our experiments with token-based filtering, we have been able to improve the results in the evaluation setting that we have created for search tasks but not in the evaluation setting for NLP tasks, which focuses on infrequent words.

Regarding the usage of spelling variant detection in order to improve NLP tasks, we have found for POS tagging and lemmatization that substituting OOV words with known spelling variants improves the accuracy for both tasks. This even works when training the pipeline for spelling variant detection with distant supervision using only the annotation available for the specific task. Therefore, no additional data is needed. We have compared this approach with adaptations of the tools to non-standard data. By doing this, we have reached similar and for some tasks slightly better results than by substituting OOV words. However, while adapting a tool is different for different tasks, the substitution of spelling variants can always be applied and is therefore a simple way to improve the performance of NLP tools when applying them to non-standard data. Moreover, the combination of tool adaptation and OOV word substitution can further improve the performance

For the ReM, we have also looked at normalization to an artificial standard GMH as a way to reduce spelling variation. While normalization has led to a slightly better performance for both POS tagging and lemmatization, it needs additional training data in order to be used. So, if such training data is not available, spelling variant detection is an easy to use alternative.

9.2. Directions for Future Work

While the approaches in this thesis do not rely on the existence of normalized training data or even the existence of a target language for normalization, they rely on the data to be tokenized. Tokenization is an easy task for modern standard texts in languages where whitespace is systematically used to delimit words. This is, however, not the case for every writing system (e. g., Chinese, cf. Huang and Zhao 2006) and also not for the historical non-standard data that we have used in the experiments. Makarov and Clematide (2020) say that "[t]okenization remains a challenge for normalization of unpreprocessed non-standard data" (p. 7291). The same holds for the approaches presented in this thesis. This is especially relevant in the context of unsupervised settings where spelling variant detection is used to help users search in unprocessed data.

In order to improve simplification and spelling variant detection for tokenized texts, there are several directions that appear as promising next steps to us.

The spelling variant candidate generator used in our pipeline depends on a dictionary of types. Types that do not appear in that dictionary cannot be generated. A promising next step would be to get rid of that dependency. This could probably be done by using an encoder-decoder architecture. Neural encoder-decoder models have been applied for normalization e. g., by Lusetti et al. (2018). In order to generate different spelling variants for a given type, the encoder-decoder model could be combined with a Variational Auto-Encoder (Kingma and Welling 2014) as for instance in Serban et al. (2017).

Regarding the filter for spelling variant candidates, instead of the SVM applied in this thesis, a Siamese Neural Network architecture could be used. Rama (2016) has applied Siamese CNNs for cognate identification. His experiments lead to promising results even with small amounts training data.

Another possible branch of improvements comes from using spelling variant detection for simplification. So, instead of learning rules for simplification, one could use the learned spelling variant relation to select the *simplified* version from the set of related spelling variants. If done in a way that makes the dictionary uniform, i. e., takes variants with the same form for morphemes like *ghe* and *ge*, this might help to model spelling variation better than with simple rules.

Regarding the applications, we have looked into POS tagging and lemmatization but spelling variant detection might be useful for other tasks as well. While we have presented spelling variant detection as an alternative to normalization, it would be interesting to see, if spelling variant detection can be helpful for normalization approaches. The idea behind this is that normalization approaches have to learn two things at once: While their aim is to map non-standard words to standard words, they also have to learn to handle the variation in the non-standard words, i. e., map different variants of a non-standard word to the same standard word. Using spelling variant detection and simplification as presented in this thesis might improve normalization since it could be used to untangle these two tasks.

Two other examples for tasks where spelling variant detection might be useful are automatic error-detection for annotations and topic modeling. For annotation error-detection, Dickinson and Meurers (2003) have proposed utilizing variation n-grams, i. e., type n-grams that appear with differing annotations in the corpus, in order to identify annotation errors in manually annotated texts. We assume that spelling variation negatively affects the recall of this method since spelling variants will be grouped in different variation n-grams.

Topic modeling (Blei 2012) is popular in DH research (Meeks and Weingart 2012). That spelling variation can influence topics has been noted by Sakr and Hasegawa-Johnson (2013) who analyze Arabic texts. The authors try reducing the variation applying hand-crafted rules but are not able to improve topic modeling for their data. Applying the methods developed in this thesis seems a promising way for improving topic modeling for texts with spelling variation.

In future work, it would also be interesting to evaluate in detail how the measure for spelling variation that we have presented correlates with the performance of NLP tools on given data.

Bibliography

Adesam, Yvonne, and Gerlof Bouma. 2016. "Old Swedish Part-of-Speech Tagging between Variation and External Knowledge." In *Proceedings of the 10th SIGHUM Workshop on Language Technology for Cultural Heritage, Social Sciences, and Humanities,* 32–42. Berlin, Germany: Association for Computational Linguistics, August. doi:10.18653/v1/W16-2104.

Amrhein, Chantal, and Simon Clematide. 2018. "Supervised OCR Error Detection and Correction Using Statistical and Neural Machine Translation Methods." In "Special issue on automatic text and layout recognition," edited by Kay-Michael Würzner, Alexander Geyken, and Günter Mühlberger, *Journal for Language Technology and Computational Linguistics (JLCL)* 33 (1): 49–76. doi:10.5167/UZH-162394.

Andrews, Tara L. 2016. "Analysis of Variation Significance in Artificial Traditions Using Stemmaweb." *Digital Scholarship in the Humanities* 31 (3): 523–539. doi:10.1093/llc/fqu072.

Archer, Dawn, Merja Kytö, Alistair Baron, and Paul Rayson. 2015. "Guidelines for Normalising Early Modern English Corpora: Decisions and Justifications." *ICAME Journal* 39:5–24. doi:10.1515/icame-2015-0001.

Baron, Alistair, Paul Rayson, and Dawn Archer. 2009. "Word Frequency and Key Word Statistics in Historical Corpus Linguistics." *Anglistik: International Journal of English Studies* 20 (1): 41–67.

Baroni, Marco, Georgiana Dinu, and Germán Kruszewski. 2014. "Don't Count, Predict! A Systematic Comparison of Context-Counting vs. Context-Predicting Semantic Vectors." In *Proceedings of the 52nd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers),* 238–247. Baltimore, Maryland: Association for Computational Linguistics, June. doi:10.3115/v1/P14-1023.

Barteld, Fabian. 2017. "Detecting Spelling Variants in Non-Standard Texts." In *Proceedings of the Student Research Workshop at the 15th Conference of the European Chapter of the Association for Computational Linguistics,* 11–22. Valencia, Spain: Association for Computational Linguistics, April. Accessed October 2, 2021. https://aclanthology.org/E17-4002.

Barteld, Fabian, Chris Biemann, and Heike Zinsmeister. 2018. "Variations on the Theme of Variation: Dealing with Spelling Variation for Fine-Grained POS Tagging of Historical Texts." In *Proceedings of the 14th Conference on Natural Language Processing (KONVENS 2018),* edited by Hanno Biber, Christina Katsikadeli, and Manfred B. Sellner, 202–212. Vienna, Austria: Verlag der Österreichischen Akademie der Wissenschaften, September. doi:10.1553/0x003a12bd.

———. 2019. "Token-based Spelling Variant Detection in Middle Low German Texts." In "Language Technology for Digital Humanities," edited by Erhard Hinrichs, Marie Hinrichs, Sandra Kübler, and Thorsten Trippel, *Language Resources and Evaluation* 53 (4): 677–706. doi:10.1007/s10579-018-09441-5.

Barteld, Fabian, Katharina Dreessen, Sarah Ihden, and Ingrid Schröder. 2017. "Das Referenzkorpus Mittelniederdeutsch/Niederrheinisch (1200–1650) – Korpusdesign, Korpuserstellung und Korpusnutzung." In "Mittelniederdeutsche Literatur," edited by Anja Becker and Albrecht Hausmann, *Mitteilungen des Deutschen Germanistenverbandes* 64 (3): 226–241. doi:10.14220/mdge.2017.64.3.226.

———. 2019. "Analyse syntaktischer Phänomene mit dem Referenzkorpus Mittelniederdeutsch / Niederrheinisch (1200–1650)." In "Historische Korpuslinguistik," edited by Renata Szczepaniak, Stefan Hartmann, and Lisa Dücker, *Jahrbuch für Germanistische Sprachgeschichte* 10 (1): 261–281. doi:10.1515/jbgsg-2019-0015.

Barteld, Fabian, Sarah Ihden, Katharina Dreessen, and Ingrid Schröder. 2018. "HiNTS: A Tagset for Middle Low German." In *Proceedings of the Eleventh International Conference on Language Resources and Evaluation (LREC 2018),* 3940–3945. Miyazaki, Japan: European Language Resources Association (ELRA), May. Accessed October 2, 2021. https://www.aclweb.org/anthology/L18-1622.

Barteld, Fabian, Ingrid Schröder, and Heike Zinsmeister. 2015. "Unsupervised Regularization of Historical Texts for POS Tagging." In *Proceedings of the Workshop on Corpus-Based Research in the Humanities (CRH),* 3–12. Warsaw, Poland, December. Accessed October 2, 2021. http://crh4.ipipan.waw.pl/files/9814/4973/5451/CRH4_proceedings.pdf.

———. 2016. "Dealing with Word-Internal Modification and Spelling Variation in Data-Driven Lemmatization." In *Proceedings of the 10th SIGHUM Workshop on Language Technology for Cultural Heritage, Social Sciences, and Humanities,* 52–62. Berlin, Germany: Association for Computational Linguistics, August. doi:10.18653/v1/W16-2106.

Besch, Werner, Anne Betten, Oskar Reichmann, and Stefan Sonderegger, eds. 2003. *Sprachgeschichte. Ein Handbuch zur Geschichte der deutschen Sprache und ihrer Erforschung.* 2. vollst. neubearb. und erw. Aufl. Bk. 3. Handbücher zur Sprach- und Kommunikationswissenschaft. Berlin, Boston: De Gruyter. doi:10.1515/9783110194173.

Biemann, Chris. 2012. *Structure Discovery in Natural Language.* Theory and Applications of Natural Language Processing. Berlin, Heidelberg: Springer. doi:10.1007/978-3-642-25923-4.

Bishop, Christopher M. 2006. *Pattern Recognition and Machine Learning.* Information Science and Statistics. New York: Springer.

Blei, David M. 2012. "Probabilistic Topic Models." *Communications of the ACM* 55 (4): 77–84. doi:10.1145/2133806.2133826.

Bojanowski, Piotr, Edouard Grave, Armand Joulin, and Tomas Mikolov. 2017. "Enriching Word Vectors with Subword Information." *Transactions of the Association for Computational Linguistics* 5:135–146. doi:10.1162/tacl_a_00051.

Bollmann, Marcel. 2012. "(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),* edited by Francesco Mambrini, Marco Passarotti, and Caroline Sporleder, 3–14. Lisbon, Portugal, November. Accessed October 2, 2021. http://alfclul.clul.ul.pt/crpc/acrh2/ACRH-2_FINAL.pdf.

———. 2013a. *Automatic Normalization for Linguistic Annotation of Historical Language Data.* Bochumer Linguistische Arbeitsbereichte 13. Bochum: Sprachwissenschaftliches Institut, Ruhr-Universität. Accessed October 2, 2021. https://www.linguistics.rub.de/forschung/arbeitsberichte/13.pdf.

———. 2013b. "POS Tagging for Historical Texts with Sparse Training Data." In *Proceedings of the 7th Linguistic Annotation Workshop and Interoperability with Discourse,* 11–18. Sofia, Bulgaria: Association for Computational Linguistics, August. Accessed October 2, 2021. https://aclanthology.org/W13-2302/.

———. 2018. *Normalization of Historical Texts with Neural Network Models.* Bochumer Linguistische Arbeitsberichte 22. Bochum: Sprachwissenschaftliches Institut, Ruhr-Universität. Accessed October 2, 2021. https://www.linguistics.rub.de/forschung/arbeitsberichte/22.pdf.

Bollmann, Marcel. 2019. "A Large-Scale Comparison of Historical Text Normalization Systems." In *Proceedings of the 2019 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, Volume 1 (Long and Short Papers)*, 3885–3898. Minneapolis, Minnesota: Association for Computational Linguistics, June. doi:10.18653/v1/N19-1389.

Bollmann, Marcel, Stefanie Dipper, Julia Krasselt, and Florian Petran. 2012. "Manual and Semi-Automatic Normalization of Historical Spelling—Case Studies from Early New High German." In *Proceedings of the 11th Conference on Natural Language Processing (KONVENS 2012), LThist 2012 Workshop*, edited by Jeremy Jancsary, 342–350. Vienna, Austria: ÖGAI, September. Accessed October 2, 2021. http://www.oegai.at/konvens2012/proceedings/51_bollmann12w/.

Bollmann, Marcel, Florian Petran, and Stefanie Dipper. 2011. "Applying Rule-Based Normalization to Different Types of Historical Texts–An Evaluation." In *Proceedings of the 5th Language & Technology Conference: Human Language Technologies as a Challenge for Computer Science and Linguistics*, 339–344. Poznan, Poland, November.

Boser, Bernhard E., Isabelle M. Guyon, and Vladimir N. Vapnik. 1992. "A Training Algorithm for Optimal Margin Classifiers." In *COLT '92: Proceedings of the Fifth Annual Workshop on Computational Learning Theory*, 144–152. New York, NY, USA: Association for Computing Machinery, July. doi:10.1145/130385.130401.

Bowers, Fredson. 1989. "Regularization and Normalization in Modern Critical Texts." *Studies in Bibliography* 42:79–102. Accessed October 2, 2021. http://xtf.lib.virginia.edu/xtf/view?docId=StudiesInBiblio/uvaBook/tei/sibv042.xml;chunk.id=vol042.04;toc.depth=1;toc.id=vol042.04;brand=default.

Brants, Thorsten. 2000. "TnT – A Statistical Part-of-Speech Tagger." In *Sixth Applied Natural Language Processing Conference*, 224–231. Seattle, Washington, USA: Association for Computational Linguistics, April. doi:10.3115/974147.974178.

Brill, Eric, and Robert C. Moore. 2000. "An Improved Error Model for Noisy Channel Spelling Correction." In *Proceedings of the 38th Annual Meeting of the Association for Computational Linguistics*, 286–293. Hong Kong: Association for Computational Linguistics, October. doi:10.3115/1075218.1075255.

Brown, Peter F., Peter V. deSouza, Robert L. Mercer, Vincent J. Della Pietra, and Jenifer C. Lai. 1992. "Class-Based N-Gram Models of Natural Language." *Computational Linguistics* 18 (4): 467–479. Accessed October 2, 2021. https://dl.acm.org/doi/10.5555/176313.176316.

Chakrabarty, Abhisek, Onkar Arun Pandit, and Utpal Garain. 2017. "Context Sensitive Lemmatization Using Two Successive Bidirectional Gated Recurrent Networks." In *Proceedings of the 55th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers),* 1481–1491. Vancouver, Canada: Association for Computational Linguistics, July. doi:10.18653/v1/P17-1136.

Chang, Chih-Chung, and Chih-Jen Lin. 2011. "LIBSVM: A Library for Support Vector Machines." *ACM Transactions on Intelligent Systems and Technology* 2 (3): 1–27. doi:10.1145/1961189.1961199.

Chrupała, Grzegorz. 2006. "Simple Data-Driven Context-Sensitive Lemmatization." In *Proceedings of SEPLN.* Zaragoza, Spain, September. Accessed October 2, 2021. http://doras.dcu.ie/15272/.

———. 2008. "Towards a Machine-Learning Architecture for Lexical Functional Grammar Parsing." PhD diss., Dublin City University. Accessed October 2, 2021. http://doras.dcu.ie/550/.

Chrupała, Grzegorz, Georgiana Dinu, and Josef van Genabith. 2008. "Learning Morphology with Morfette." In *Proceedings of the Sixth International Conference on Language Resources and Evaluation (LREC'08),* 2362–2367. Marrakech, Morocco: European Language Resources Association (ELRA), May. Accessed October 2, 2021. http://www.lrec-conf.org/proceedings/lrec2008/pdf/594_paper.pdf.

Ciobanu, Maria Alina, and Liviu P. Dinu. 2014. "Automatic Detection of Cognates Using Orthographic Alignment." In *Proceedings of the 52nd Annual Meeting of the Association for Computational Linguistics (Volume 2: Short Papers),* 99–105. Baltimore, Maryland: Association for Computational Linguistics, June. doi:10.3115/v1/P14-2017.

———. 2015. "Automatic Discrimination between Cognates and Borrowings." In *Proceedings of the 53rd Annual Meeting of the Association for Computational Linguistics and the 7th International Joint Conference on Natural Language Processing (Volume 2: Short Papers),* 431–437. Beijing, China: Association for Computational Linguistics, July. doi:10.3115/v1/P15-2071.

Cortes, Corinna, and Vladimir Vapnik. 1995. "Support-Vector Networks." *Machine Learning* 20 (3): 273–297. doi:10.1007/BF00994018.

Craven, Mark, and Johan Kumlien. 1999. "Constructing Biological Knowledge Bases by Extracting Information from Text Sources." In *Proceedings of the Seventh International Conference on Intelligent Systems for Molecular Biology,* 77–86. AAAI Press, August.

Damerau, Fred J. 1964. "A Technique for Computer Detection and Correction of Spelling Errors." *Communications of the ACM* 7 (3): 171–176. doi:10.1145/363958.363994.

Derczynski, Leon, Sean Chester, and Kenneth S. Bøgh. 2015. "Tune Your Brown Clustering, Please." In *Proceedings of the Conference on Recent Advances in Natural Language Processing (RANLP),* 110–117. Hissar, Bulgaria: INCOMA Ltd. Shoumen, September. Accessed October 2, 2021. https://aclanthology.org/R15-1016.

Dice, Lee R. 1945. "Measures of the Amount of Ecologic Association between Species." *Ecology* 26 (3): 297–302. doi:10.2307/1932409.

Dickinson, Markus, and Detmar W. Meurers. 2003. "Detecting Errors in Part-of-Speech Annotation." In *10th Conference of the European Chapter of the Association for Computational Linguistics,* 107–114. Budapest, Hungary: Association for Computational Linguistics, April. Accessed October 2, 2021. http://aclweb.org/anthology/E03-1068.

Dipper, Stefanie. 2010. "Pos-Tagging of Historical Language Data: First Experiments." In *Semantic Approaches in Natural Language Processing: Proceedings of the 10th Conference on Natural Language Processing 2010,* edited by Manfred Pinkal, Ines Rehbein, Sabine Schulte im Walde, and Angelika Storrer, 117–121. Saarbrücken, Germany: universaar: Saarland University Press, September. Accessed October 2, 2021. http://universaar.uni-saarland.de/monographien/volltexte/2010/12/pdf/konvens_2010.pdf.

———. 2011. "Morphological and Part-of-Speech Tagging of Historical Language Data: A Comparison." In "Annotation of Corpora for Research in the Humanities," edited by Francesco Mambrini, Marco Passarotti, and Caroline Sporleder, *Journal for Language Technology and Computational Linguistics (JLCL)* 26 (2): 25–37. Accessed October 2, 2021. https://jlcl.org/content/2-allissues/13-Heft2-2011/2.pdf.

———. 2015. "Annotierte Korpora für die Historische Syntaxforschung: Anwendungsbeispiele anhand des Referenzkorpus Mittelhochdeutsch." *Zeitschrift für germanistische Linguistik* 43 (3): 516–563. doi:10.1515/zgl-2015-0020.

Dipper, Stefanie, Karin Donhauser, Thomas Klein, Sonja Linde, Stefan Müller, and Klaus-Peter Wegera. 2013. "HiTS: ein Tagset für historische Sprachstufen des Deutschen." In "Das Stuttgart-Tübingen Wortarten-Tagset – Stand und Perspektiven," edited by Heike Zinsmeister, Ulrich Heid, and Kathrin Beck, *Journal for Language Technology and Computational Linguistics (JLCL)* 28 (1): 85–137. Accessed October 2, 2021. https://jlcl.org/content/2-allissues/10-Heft1-2013/5Dipper.pdf.

Dipper, Stefanie, Anke Lüdeling, and Marc Reznicek. 2013. "NoSta-D: A Corpus of German Non-Standard Varieties." In *Non-Standard Data Sources in Corpus-Based Research,* edited by Marcos Zampieri and Sascha Diwersy, 69–76. ZSM-Studien 5. Aachen: Shaker.

Dipper, Stefanie, and Sandra Waldenberger. 2017. "Investigating Diatopic Variation in a Historical Corpus." In *Proceedings of the Fourth Workshop on NLP for Similar Languages, Varieties and Dialects (VarDial),* 36–45. Valencia, Spain: Association for Computational Linguistics, April. doi:10.18653/v1/W17-1204.

Dürscheid, Christa. 2016. *Einführung in die Schriftlinguistik.* 5., aktualisierte und korrigierte Auflage. Göttingen: Vandenhoeck & Ruprecht.

Echelmeyer, Nora, Nils Reiter, and Sarah Schulz. 2017. "Ein PoS-Tagger für "das" Mittelhochdeutsche." In *Dhd 2017. Digitale Nachhaltigkeit. Konferenzabstracts,* 141–147. Bern, Switzerland, February.

Edwards, Allen L. 1948. "Note on the "Correction for Continuity" in Testing the Significance of the Difference between Correlated Proportions." *Psychometrika* 13 (3): 185–187. doi:10.1007/BF02289261.

Egghe, Leo. 2010. "Good Properties of Similarity Measures and Their Complementarity." *Journal of the American Society for Information Science and Technology* 61 (10): 2151–2160. doi:10.1002/asi.21380.

Eisenstein, Jacob. 2013. "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,* 359–369. Atlanta, Georgia: Association for Computational Linguistics, June. Accessed October 2, 2021. http://aclweb.org/anthology/N13-1037.

Elmentaler, Michael. 2018. *Historische Graphematik des Deutschen Eine Einführung.* narr Studienbücher. Tübingen: Narr Francke Attempto Verlag GmbH + Co. KG. Accessed October 2, 2021. https://elibrary.narr.digital/book/99.125005/9783823379270.

Ernst-Gerlach, Andrea, and Norbert Fuhr. 2006. "Generating Search Term Variants for Text Collections with Historic Spellings." In *Advances in Information Retrieval: 28th European Conference on IR Research, ECIR 2006, London, UK, April 10-12, 2006. Proceedings,* edited by Mounia Lalmas, Andy MacFarlane, Stefan Rüger, Anastasios Tombros, Theodora Tsikrika, and Alexei Yavlinsky, 49–60. Lecture Notes in Computer Science 3936. Berlin, Heidelberg: Springer. doi:10.1007/11735106_6.

Flick, Johanna. 2019. "„Alte" Daten, neue Methoden." In "Historische Korpuslinguistik," edited by Renata Szczepaniak, Stefan Hartmann, and Lisa Dücker, *Jahrbuch für Germanistische Sprachgeschichte* 10 (1): 151–175. doi:10.1515/jbgsg-2019-0010.

Fulop, Sean A., and Sylvain Neuvel. 2013. "Networks of Morphological Relations." In *Proceedings of the International Symposium on Artificial Intelligence and Mathematics (ISAIM-2014).* Fort Lauderdale, Florida, USA, January. Accessed October 2, 2021. https://www.cs.uic.edu/pub/Isaim2014/WebPreferences/ISAIM2014_NLP_Fulop_Neuvel.pdf.

Galar, M., A. Fernandez, E. Barrenechea, H. Bustince, and F. Herrera. 2012. "A Review on Ensembles for the Class Imbalance Problem: Bagging-, Boosting-, and Hybrid-Based Approaches." *IEEE Transactions on Systems, Man, and Cybernetics, Part C (Applications and Reviews)* 42 (4): 463–484. doi:10.1109/TSMCC.2011.2161285.

Giesbrecht, Eugenie, and Stefan Evert. 2009. "Is Part-of-Speech Tagging a Solved Task? An Evaluation of Pos Taggers for the German Web as Corpus." In *Proceedings of the 5th Web as Corpus Workshop (WAC5),* 27–35. San Sebastian, Spain, September. Accessed October 2, 2021. https://www.sigwac.org.uk/attachment/wiki/WAC5/WAC5_proceedings.pdf.

Giménez, Jesús, and L. Marquez. 2012. *SVMTool - Technical Manual v1. 4.* Technical report. TALP Research Center, Universitat Politécnica de Catalunya.

Giménez, Jesús, and Lluís Màrquez. 2004. "SVMTool: A General POS Tagger Generator Based on Support Vector Machines." In *Proceedings of the Fourth International Conference on Language Resources and Evaluation (LREC'04),* 43–46. Lisbon, Portugal: European Language Resources Association (ELRA), May. Accessed October 2, 2021. http://www.lrec-conf.org/proceedings/lrec2004/pdf/597.pdf.

Gimpel, Kevin, Nathan Schneider, Brendan O'Connor, Dipanjan Das, Daniel Mills, Jacob Eisenstein, Michael Heilman, Dani Yogatama, Jeffrey Flanigan, and Noah A. Smith. 2011. "Part-of-Speech Tagging for Twitter: Annotation, Features, and Experiments." In *Proceedings of the 49th Annual Meeting of the Association for Computational Linguistics: Human Language Technologies,* 42–47. Portland, Oregon, USA: Association for Computational Linguistics, June. Accessed October 2, 2021. http://aclweb.org/anthology/P11-2008.

Goldberg, Yoav. 2017. *Neural Network Methods in Natural Language Processing.* Synthesis Lectures on Human Language Technologies. San Rafael: Morgan & Claypool Publishers. doi:10.2200/S00762ED1V01Y201703HLT037.

Gomes, Luís, and José Gabriel P. Lopes. 2011. "Measuring Spelling Similarity for Cognate Identification." In *Progress in Artificial Intelligence: 15th Portuguese Conference on Artificial Intelligence, EPIA 2011, Lisbon, Portugal, October 10-13, 2011. Proceedings,* edited by Luis Antunes and H. Sofia Pinto, 624–633. Lecture Notes in Computer Science 7026. Berlin, Heidelberg: Springer. doi:10.1007/978-3-642-24769-9_45.

Goodfellow, Ian, Yoshua Bengio, and Aaron Courville. 2016. *Deep Learning.* Adaptive Computation and Machine Learning. Cambridge, MA, USA: MIT Press. Accessed October 2, 2021. http://www.deeplearningbook.org.

Goot, Rob van der, Barbara Plank, and Malvina Nissim. 2017. "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,* 31–39. Copenhagen, Denmark: Association for Computational Linguistics, September. doi:10.18653/v1/W17-4404.

Guthrie, David, Ben Allison, Wei Liu, Louise Guthrie, and Yorick Wilks. 2006. "A Closer Look at Skip-Gram Modelling." In *Proceedings of the Fifth International Conference on Language Resources and Evaluation (LREC'06),* 1222–1225. Genoa, Italy: European Language Resources Association (ELRA), May. Accessed October 2, 2021. http://www.lrec-conf.org/proceedings/lrec2006/pdf/357_pdf.pdf.

Halácsy, Péter, András Kornai, and Csaba Oravecz. 2007. "HunPos – an Open Source Trigram Tagger." In *Proceedings of the 45th Annual Meeting of the Association for Computational Linguistics Companion Volume Proceedings of the Demo and Poster Sessions,* 209–212. Prague, Czech Republic: Association for Computational Linguistics, June. Accessed October 2, 2021. https://aclanthology.org/P07-2053.

Hamilton, William L., Jure Leskovec, and Dan Jurafsky. 2016. "Diachronic Word Embeddings Reveal Statistical Laws of Semantic Change." In *Proceedings of the 54th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers),* 1489–1501. Berlin, Germany: Association for Computational Linguistics, August. doi:10.18653/v1/P16-1141.

Han, Bo, and Timothy Baldwin. 2011. "Lexical Normalisation of Short Text Messages: Makn Sens a #twitter." In *Proceedings of the 49th Annual Meeting of the Association for Computational Linguistics: Human Language Technologies,* 368–378. Portland, Oregon, USA: Association for Computational Linguistics, June. Accessed October 2, 2021. http://aclweb.org/anthology/P11-1038.

Han, Bo, Paul Cook, and Timothy Baldwin. 2013. "Lexical Normalization for Social Media Text." *ACM Transactions on Intelligent Systems and Technology* 4 (1): 1–27. doi:10.1145/2414425.2414430.

Harris, Zellig S. 1954. "Distributional Structure." *Word* 10 (2-3): 146–162. doi:10.1080/00437956.1954.11659520.

Hastie, Trevor, Robert Tibshirani, and Jerome Friedman. 2009. *The Elements of Statistical Learning. Data Mining, Inference, and Prediction.* 2nd ed. Springer Series in Statistics. New York: Springer. doi:10.1007/978-0-387-84858-7.

Hathout, Nabil. 2009. "Acquisition of Morphological Families and Derivational Series from a Machine Readable Dictionary." In *Selected Proceedings of the 6th Décembrettes: Morphology in Bordeaux, December 2008,* edited by Fabio Montermini, Gilles Boyé, and Jesse Tseng, 166–180. Somerville, MA: Cascadilla Proceedings Project. Accessed October 2, 2021. http://www.lingref.com/cpp/decemb/6/paper2244.pdf.

———. 2014. "Phonotactics in Morphological Similarity Metrics." In "Theoretical and Empirical Approaches to Phonotactics and Morphonotactics," edited by Basilio Calderone, Chiara Celata, and Bernard Laks, *Language Sciences* 46 (Part A): 71–83. doi:10.1016/j.langsci.2014.06.008.

Hauser, Andreas W., and Klaus U. Schulz. 2007. "Unsupervised Learning of Edit Distance Weights for Retrieving Historical Spelling Variations." In *Proceedings of the Workshop on Finite-State Techniques and Approximate Search,* 1–6. Borovets, Bulgaria, September. Accessed October 2, 2021. https://acl-bg.org/proceedings/2007/RANLP_W8%202007.pdf.

Huang, Chang-Ning, and Hai Zhao. 2006. "Which Is Essential for Chinese Word Segmentation: Character versus Word." In *Proceedings of the 20th Pacific Asia Conference on Language, Information and Computation,* 1–12. Huazhong Normal University, Wuhan, China: Tsinghua University Press, November. Accessed October 2, 2021. https://aclanthology.org/Y06-1001.

Ihden, Sarah. 2020. *Relativsätze im Mittelniederdeutschen. Korpuslinguistische Untersuchungen zu Struktur und Gebrauch.* Lingua Historica Germanica 23. Berlin, Boston: de Gruyter. doi:10.1515/9783110678468.

Ihden, Sarah, and Ingrid Schröder. 2021. "Mittelniederdeutsche Grammatik: Konzeption und erste Analysen." *Niederdeutsches Jahrbuch. Jahrbuch des Vereins für niederdeutsche Sprachforschung* 144:79–104.

Jin, Ning. 2015. "NCSU-SAS-Ning: Candidate Generation and Feature Engineering for Supervised Lexical Normalization." In *Proceedings of the Workshop on Noisy User-generated Text,* 87–92. Beijing, China: Association for Computational Linguistics, July. doi:10.18653/v1/W15-4313.

Jongejan, Bart, and Hercules Dalianis. 2009. "Automatic Training of Lemmatization Rules that Handle Morphological Changes in Pre-, In- and Suffixes Alike." 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,* 145–153. Suntec, Singapore: Association for Computational Linguistics, August. Accessed October 2, 2021. https://aclanthology.org/P09-1017.

Jurish, Bryan. 2010a. "Comparing Canonicalizations of Historical German Text." In *Proceedings of the 11th Meeting of the ACL Special Interest Group on Computational Morphology and Phonology,* 72–77. Uppsala, Sweden: Association for Computational Linguistics, July. Accessed October 2, 2021. http://aclweb.org/anthology/W10-2209.

———. 2010b. "More than Words: Using Token Context to Improve Canonicalization of Historical German." Edited by Lother Lemnitzer. *Journal for Language Technology and Computational Linguistics (JLCL)* 25 (1): 23–39. Accessed October 2, 2021. https://jlcl.org/content/2-allissues/15-Heft1-2010/bryan_jurish.pdf.

———. 2011. "Finite-State Canonicalization Techniques for Historical German." PhD diss., Universität Potsdam. Accessed October 2, 2021. https://publishup.uni-potsdam.de/frontdoor/index/index/docId/5562.

Kestemont, Mike, Walter Daelemans, and Guy de Pauw. 2010. "Weigh Your Words—Memory-Based Lemmatization for Middle Dutch." *Literary and Linguistic Computing* 25 (3): 287–301. doi:10.1093/llc/fqq011.

Kim, Yoon. 2014. "Convolutional Neural Networks for Sentence Classification." In *Proceedings of the 2014 Conference on Empirical Methods in Natural Language Processing (EMNLP),* 1746–1751. Doha, Qatar: Association for Computational Linguistics, October. doi:10.3115/v1/D14-1181.

Kingma, Diederik P., and Jimmy Ba. 2015. "Adam: A Method for Stochastic Optimization." In *Proceedings of the 3rd International Conference on Learning Representations (ICLR) 2015, Conference Track.* San Diego, USA, May. Accessed October 2, 2021. https://arxiv.org/abs/1412.6980.

Kingma, Diederik P., and Max Welling. 2014. "Auto-Encoding Variational Bayes." In *Proceedings of the 2nd International Conference on Learning Representations (ICLR) 2014, Conference Track.* Banff, AB, Canada, April. Accessed October 2, 2021. http://arxiv.org/abs/1312.6114.

Klein, Thomas, and Stefanie Dipper. 2016. *Handbuch zum Referenzkorpus Mittelhochdeutsch.* Bochumer Linguistische Arbeitsbereichte 19. Bochum: Sprachwissenschaftliches Institut, Ruhr-Universität. Accessed October 2, 2021. https://www.linguistics.rub.de/forschung/arbeitsberichte/19.pdf.

Kobus, Catherine, François Yvon, and Géraldine Damnati. 2008. "Normalizing SMS: Are Two Metaphors Better Than One?" In *Proceedings of the 22nd International Conference on Computational Linguistics (Coling 2008),* 441–448. Manchester, United Kingdom: Coling 2008 Organizing Committee, August. Accessed October 2, 2021. http://aclweb.org/anthology/C08-1056.

Kolachina, Prasanth, Martin Riedl, and Chris Biemann. 2017. "Replacing OOV Words For Dependency Parsing With Distributional Semantics." In *Proceedings of the 21st Nordic Conference on Computational Linguistics,* 11–19. Gothenburg, Sweden: Association for Computational Linguistics, May. Accessed October 2, 2021. http://aclweb.org/anthology/W17-0202.

Koleva, Mariya, Melissa Farasyn, Bart Desmet, Anne Breitbarth, and Véronique Hoste. 2017. "An Automatic Part-of-Speech Tagger for Middle Low German." *International Journal of Corpus Linguistics* 22 (1): 107–140. doi:10.1075/ijcl.22.1.05kol.

König, Werner, Stephan Elspaß, and Robert Möller. 2005. *dtv-Atlas Deutsche Sprache.* 19., überarbeitete und korrigierte Auflage. dtv-Atlas. München: Deutscher Taschenbuch Verlag.

Kragl, Florian. 2015. "Normalmittelhochdeutsch. Theorieentwurf einer gelebten Praxis." *Zeitschrift für deutsches Altertum und deutsche Literatur* 144 (1): 1–27. Accessed October 2, 2021. http://www.jstor.org/stable/43672053.

Krause, Thomas, and Amir Zeldes. 2016. "ANNIS3: A New Architecture for Generic Corpus Query and Visualization." *Digital Scholarship in the Humanities* 31 (1): 118–139. doi:10.1093/llc/fqu057.

Kumar, Ankit, Piyush Makhija, and Anuj Gupta. 2020. "Noisy Text Data: Achilles' Heel of BERT." In *Proceedings of the Sixth Workshop on Noisy User-generated Text (W-NUT 2020),* 16–21. Online: Association for Computational Linguistics, November. doi:10.18653/v1/2020.wnut-1.3.

Lasch, Agathe. 2011. *Mittelniederdeutsche Grammatik.* Sammlung kurzer Grammatiken germanischer Dialekte. A: Hauptreihe 9. Berlin, New York: Max Niemeyer Verlag. doi:10.1515/9783111393124.

Lemnitzer, Lothar, and Heike Zinsmeister. 2015. *Korpuslinguistik. Eine Einführung.* 3rd ed. narr Studienbücher. Tübingen: Narr Francke Attempto Verlag GmbH + Co. KG. Accessed October 2, 2021. https://elibrary.narr.digital/book/99.125005/9783823378860.

Levandowsky, Michael, and David Winter. 1971. "Distance between Sets." *Nature* 234:34–35. doi:10.1038/234034a0.

Levenshtein, Vladimir. 1966. "Binary Codes Capable of Correcting Deletions, Insertions, and Reversals." *Soviet Physics Doklady* 10 (8): 707–710.

Levy, Omer, Yoav Goldberg, and Ido Dagan. 2015. "Improving Distributional Similarity with Lessons Learned from Word Embeddings." *Transactions of the Association for Computational Linguistics* 3:211–225. doi:10.1162/tacl_a_00134.

Li, Xiao-Li, and Bing Liu. 2005. "Learning from Positive and Unlabeled Examples with Different Data Distributions." In *Machine Learning: ECML 200,: 16th European Conference on Machine Learning, Porto, Portugal, October 3-7, 2005. Proceedings,* edited by João Gama, Rui Camacho, Pavel B. Brazdil, Alípio Mário Jorge, and Luís Torgo, 218–229. Lecture Notes in Computer Science 3720. Berlin, Heidelberg: Springer. doi:10.1007/11564096_24.

Liang, Percy. 2005. "Semi-Supervised Learning for Natural Language." Master's thesis, Massachusetts Institute of Technology. Accessed October 2, 2020. http://hdl.handle.net/1721.1/33296.

Littell, Patrick, Shobhana Chelliah, and Gina-Anne Levow. 2016. "Text Normalization for Endangered Languages: A Shared Task Challenge." In *The 2nd Workshop on Noisy User-generated Text (W-NUT). Abstracts.* Osaka, Japan, December. Accessed October 2, 2021. `http://noisy-text.github.io/2016/pdf/WNUT32abstract.pdf`.

Ljubešić, Nikola, Katja Zupan, Darja Fišer, and Tomaž Erjavec. 2016. "Normalising Slovene Data: Historical Texts vs. User-Generated Content." In *Proceedings of the 13th Conference on Natural Language Processing (KONVENS 2016),* 146–155. Bochumer Linguistische Arbeitsberichte 16. Bochum, Germany, September. Accessed October 2, 2021. `https://www.linguistics.rub.de/konvens16/pub/19_konvensproc.pdf`.

Logačev, Pavel, Katrin Goldschmidt, and Ulrike Demske. 2014. "POS-Tagging Historical Corpora: The Case of Early New High German." In *Proceedings of the Thirteenth International Workshop on Treebanks and Linguistic Theories (TLT-13),* 103–112. Tübingen, Germany, December. Accessed October 2, 2021. `http://tlt13.sfs.uni-tuebingen.de/tlt13-proceedings.pdf`.

Lusetti, Massimo, Tatyana Ruzsics, Anne Göhring, Tanja Samardžić, and Elisabeth Stark. 2018. "Encoder-Decoder Methods for Text Normalization." In *Proceedings of the Fifth Workshop on NLP for Similar Languages, Varieties and Dialects (VarDial 2018),* 18–28. Santa Fe, New Mexico, USA: Association for Computational Linguistics, August. Accessed October 2, 2021. `https://aclanthology.org/W18-3902`.

Makarov, Peter, and Simon Clematide. 2020. "Semi-Supervised Contextual Historical Text Normalization." In *Proceedings of the 58th Annual Meeting of the Association for Computational Linguistics,* 7284–7295. Online: Association for Computational Linguistics, July. doi:`10.18653/v1/2020.acl-main.650`.

Manjavacas, Enrique, Ákos Kádár, and Mike Kestemont. 2019. "Improving Lemmatization of Non-Standard Languages with Joint Learning." In *Proceedings of the 2019 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, Volume 1 (Long and Short Papers),* 1493–1503. Minneapolis, Minnesota: Association for Computational Linguistics, June. doi:`10.18653/v1/N19-1153`.

Manning, Christopher D., and Hinrich Schütze. 1999. *Foundations of Statistical Natural Language Processing.* Cambridge, MA, USA: MIT Press.

Marcus, Mitchell P., Beatrice Santorini, and Mary Ann Marcinkiewicz. 1993. "Building a Large Annotated Corpus of English: The Penn Treebank." In "Special Issue on Using Large Corpora: II," *Computational Linguistics* 19 (2): 313–330. Accessed October 2, 2021. https://www.aclweb.org/anthology/J93-2004.

McNemar, Quinn. 1947. "Note on the Sampling Error of the Difference between Correlated Proportions or Percentages." *Psychometrika* 12 (2): 153–157. doi:10.1007/BF02295996.

Meeks, Elijah, and Scott B. Weingart. 2012. "The Digital Humanities Contribution to Topic Modeling." *Journal of Digital Humanities* 2 (1): 2–6. Accessed October 2, 2021. http://journalofdigitalhumanities.org/2-1/dh-contribution-to-topic-modeling/.

Mihov, Stoyan, and Klaus U. Schulz. 2004. "Fast Approximate Search in Large Dictionaries." *Computational Linguistics* 30 (4): 451–477. doi:10.1162/0891201042544938.

Mikolov, Tomas, Kai Chen, Greg Corrado, and Jeffrey Dean. 2013. "Efficient Estimation of Word Representations in Vector Space." In *Proceedings of the International Conference on Learning Representations (ICLR) 2013, Workshop Track*. Scottsdale, Arizona, USA, May. Accessed October 2, 2021. http://arxiv.org/abs/1301.3781.

Mikolov, Tomas, Wen-tau Yih, and Geoffrey Zweig. 2013. "Linguistic Regularities in Continuous Space Word Representations." In *Proceedings of the 2013 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies,* 746–751. Atlanta, Georgia: Association for Computational Linguistics, June. Accessed October 2, 2021. https://aclanthology.org/N13-1090.

Miller, Scott, Jethran Guinness, and Alex Zamanian. 2004. "Name Tagging with Word Clusters and Discriminative Training." In *Proceedings of the Human Language Technology Conference of the North American Chapter of the Association for Computational Linguistics: HLT-NAACL 2004,* 337–342. Boston, Massachusetts, USA: Association for Computational Linguistics, May. Accessed October 2, 2021. https://www.aclweb.org/anthology/N04-1043.

Mintz, Mike, Steven Bills, Rion Snow, and Daniel Jurafsky. 2009. "Distant Supervision for Relation Extraction without Labeled Data." 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,* 1003–1011. Suntec, Singapore: Association for Computational Linguistics, August. Accessed October 2, 2021. https://www.aclweb.org/anthology/P09-1113.

Mitchell, Tom Michael. 1997. *Machine Learning.* McGraw-Hill Series in Computer Science. New York: McGraw-Hill.

Mordelet, F., and J.-P. Vert. 2014. "A Bagging SVM to Learn from Positive and Unlabeled Examples." In "Partially Supervised Learning for Pattern Recognition," edited by Friedhelm Schwenker and Edmondo Trentin, *Pattern Recognition Letters* 37:201–209. doi:10.1016/j.patrec.2013.06.010.

Müller, Thomas, Ryan Cotterell, Alexander Fraser, and Hinrich Schütze. 2015. "Joint Lemmatization and Morphological Tagging with Lemming." In *Proceedings of the 2015 Conference on Empirical Methods in Natural Language Processing (EMNLP),* 2268–2274. Lisbon, Portugal: Association for Computational Linguistics, September. doi:10.18653/v1/D15-1272.

Müller, Thomas, Helmut Schmid, and Hinrich Schütze. 2013. "Efficient Higher-Order CRFs for Morphological Tagging." In *Proceedings of the 2013 Conference on Empirical Methods in Natural Language Processing,* 322–332. Seattle, Washington, USA: Association for Computational Linguistics, October. Accessed October 2, 2021. https://aclanthology.org/D13-1032.

Müller, Thomas, and Hinrich Schütze. 2015. "Robust Morphological Tagging with Word Representations." In *Proceedings of the 2015 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies,* 526–536. Denver, Colorado, USA: Association for Computational Linguistics, May. doi:10.3115/v1/N15-1055.

Neuvel, Sylvain, and Sean A. Fulop. 2002. "Unsupervised Learning of Morphology Without Morphemes." In *Proceedings of the ACL-02 Workshop on Morphological and Phonological Learning,* 31–40. Association for Computational Linguistics, July. doi:10.3115/1118647.1118651.

Niebaum, Hermann. 2000. "Phonetik und Phonologie, Graphetik und Graphemik des Mittelniederdeutschen." In *Sprachgeschichte. Ein Handbuch zur Geschichte der deutschen Sprache und ihrer Erforschung,* 2. vollst. neubearb. und erw. Aufl., edited by Werner Besch, Anne Betten, Oskar Reichmann, and Stefan Sonderegger, bk. 2, 1422–1430. Handbücher zur Sprach- und Kommunikationswissenschaft. Berlin, Boston: De Gruyter.

Nübling, Damaris, Antje Dammel, Janet Duke, and Renata Szczepaniak. 2017. *Historische Sprachwissenschaft des Deutschen. Eine Einführung in die Prinzipien des Sprachwandels.* 5., aktualisiert Aufl. narr Studienbücher. Tübingen: Narr Francke Attempto Verlag GmbH + Co. KG. Accessed October 2, 2021. https://elibrary.narr.digital/book/99.125005/9783823390732.

Odebrecht, Carolin, Malte Belz, Amir Zeldes, Anke Lüdeling, and Thomas Krause. 2017. "RIDGES Herbology: Designing a Diachronic Multi-Layer Corpus." *Language Resources and Evaluation* 51 (3): 695–725. doi:10.1007/s10579-016-9374-3.

Peters, Robert, and Norbert Nagel. 2014. "Das Digitale ‚Referenzkorpus Mittelniederdeutsch / Niederrheinisch (ReN)'." In "Paradigmen der aktuellen Sprachgeschichtsforschung," edited by Hans Ulrich Schmid and Arne Ziegler, *Jahrbuch für Germanistische Sprachgeschichte* 5 (1): 165–175. doi:10.1515/jbgsg-2014-0012.

Petran, Florian, Marcel Bollmann, Stefanie Dipper, and Thomas Klein. 2016. "ReM: A Reference Corpus of Middle High German — Corpus Compilation, Annotation, and Access." Edited by Armin Hoenen, Alexander Mehler, and Jost Gippert. *Journal for Language Technology and Computational Linguistics (JLCL)* 31 (2): 1–15. Accessed October 2, 2021. https://jlcl.org/content/2-allissues/4-Heft2-2016/01Petran.pdf.

Pettersson, Eva, Beáta Megyesi, and Joakim Nivre. 2013. "Normalisation of Historical Text Using Context-Sensitive Weighted Levenshtein Distance and Compound Splitting." In *Proceedings of the 19th Nordic Conference of Computational Linguistics (NODALIDA 2013),* 163–179. Oslo, Norway: Linköping University Electronic Press, Sweden, May. Accessed October 2, 2021. https://aclanthology.org/W13-5617.

Pilz, Thomas, Wolfram Luther, Norbert Fuhr, and Ulrich Ammon. 2006. "Rule-Based Search in Text Databases with Nonstandard Orthography." *Literary and Linguistic Computing* 21 (2): 179–186. doi:10.1093/llc/fql020.

Piotrowski, Michael. 2012. *Natural Language Processing for Historical Texts.* Synthesis Lectures on Human Language Technologies. San Rafael: Morgan & Claypool Publishers. doi:10.2200/S00436ED1V01Y201207HLT017.

Plank, Barbara. 2016. "What to Do about Non-Standard (or *Non-Canonical*) Language in NLP." In *Proceedings of the 13th Conference on Natural Language Processing (KONVENS 2016),* 13–20. Bochumer Linguistische Arbeitsberichte 16. Bochum, Germany, September. Accessed October 2, 2021. https://www.linguistics.rub.de/konvens16/pub/2_konvensproc.pdf.

Plank, Barbara, Dirk Hovy, Ryan McDonald, and Anders Søgaard. 2014. "Adapting Taggers to Twitter with Not-so-Distant Supervision." In *Proceedings of COLING 2014, the 25th International Conference on Computational Linguistics: Technical Papers,* 1783–1792. Dublin, Ireland: Dublin City University / Association for Computational Linguistics, August. Accessed October 2, 2021. https://www.aclweb.org/anthology/C14-1168.

Rama, Taraka. 2016. "Siamese Convolutional Networks for Cognate Identification." In *Proceedings of COLING 2016, the 26th International Conference on Computational Linguistics: Technical Papers,* 1018–1027. Osaka, Japan: The COLING 2016 Organizing Committee, December. Accessed October 2, 2021. https://aclanthology.org/C16-1097.

Rat für deutsche Rechtschreibung, ed. 2018. *Deutsche Rechtschreibung. Regeln und Wörterverzeichnis: Aktualisierte Fassung des amtlichen Regelwerks entsprechend den Empfehlungen des Rats für deutsche Rechtschreibung 2016.* Mannheim. Accessed October 2, 2021. https://www.rechtschreibrat.com/DOX/rfdr_Regeln_2016_redigiert_2018.pdf.

Resnik, Philip, and Jimmy Lin. 2010. "Evaluation of NLP Systems." Chap. 11 in *The Handbook of Computational Linguistics and Natural Language Processing,* edited by Alexander Clark, Chris Fox, and Shalom Lappin, 271–295. Blackwell Handbooks in Linguistics. Chichester, West Sussex: Wiley-Blackwell. doi:10.1002/9781444324044.ch11.

Reynaert, Martin. 2004. "Text Induced Spelling Correction." In *COLING 2004: Proceedings of the 20th International Conference on Computational Linguistics,* 834–840. Geneva, Switzerland: COLING, August. Accessed October 2, 2021. https://aclanthology.org/C04-1120.

Robertson, Alexander, and Sharon Goldwater. 2018. "Evaluating Historical Text Normalization Systems: How Well Do They Generalize?" In *Proceedings of the 2018 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, Volume 2 (Short Papers),* 720–725. New Orleans, Louisiana: Association for Computational Linguistics, June. doi:10.18653/v1/N18-2113.

Sakr, Ali, and Mark Hasegawa-Johnson. 2013. "Topic Modeling of Phonetic Latin-Spelled Arabic for the Relative Analysis of Genre-Dependent and Dialect-Dependent Variation." *Revue de l'Information Scientifique et Technique* 21 (1): 63–71. Accessed October 2, 2021. https://www.asjp.cerist.dz/en/article/45937.

Sanders, Willy. 1982. *Sachsensprache, Hansesprache, Plattdeutsch. Sprachgeschichtliche Grundzüge des Niederdeutschen.* Göttingen: Vandenhoeck & Ruprecht.

Saussure, Ferdinand de. (1931) 2001. *Grundfragen der allgemeinen Sprachwissenschaft.* 3rd ed. Berlin and New York: Walter de Gruyter. doi:10.1515/9783110870183.

Scherrer, Yves, and Tomaž Erjavec. 2016. "Modernising Historical Slovene Words." *Natural Language Engineering* 22 (6): 881–905. doi:10.1017/S1351324915000236.

Schiller, A., S. Teufel, C. Stöckert, and C. Thielen. 1999. *Guidelines für das Tagging deutscher Textcorpora mit STTS (Kleines und großes Tagset).* Technical report. University of Stuttgart and University of Tübingen.

Schmid, Helmut. 1994. "Probabilistic Part-of-Speech Tagging Using Decision Trees." In *Proceedings of the International Conference on New Methods in Language Processing.* Manchester, UK. Accessed October 2, 2021. https://www.cis.uni-muenchen.de/~schmid/tools/TreeTagger/data/tree-tagger1.pdf.

———. 1995. "Improvements in Part-of-Speech Tagging with an Application to German." In *In Proceedings of the EACL SIGDAT-Workshop.* Dublin, Ireland, March. Accessed October 2, 2021. https://www.cis.uni-muenchen.de/~schmid/tools/TreeTagger/data/tree-tagger2.pdf.

———. 2019. "Deep Learning-Based Morphological Taggers and Lemmatizers for Annotating Historical Texts." In *Proceedings of the 3rd International Conference on Digital Access to Textual Cultural Heritage,* 133–137. Brussels, Belgium: Association for Computing Machinery, May. doi:10.1145/3322905.3322915.

Schmid, Helmut, and Florian Laws. 2008. "Estimation of Conditional Probabilities with Decision Trees and an Application to Fine-Grained POS Tagging." In *Proceedings of the 22nd International Conference on Computational Linguistics (Coling 2008),* 777–784. Manchester, UK: Coling 2008 Organizing Committee, August. Accessed October 2, 2021. https://aclanthology.org/C08-1098.

Schölkopf, Bernhard, and Alexander J. Smola. 2001. *Learning with Kernels: Support Vector Machines, Regularization, Optimization, and Beyond.* Adaptive Computation and Machine Learning. Cambridge, MA, USA: MIT Press.

Schröder, Ingrid. 2014. "Das Referenzkorpus: Neue Perspektiven für die mittelniederdeutsche Grammatikographie." In "Paradigmen der aktuellen Sprachgeschichtsforschung," edited by Hans Ulrich Schmid and Arne Ziegler, *Jahrbuch für Germanistische Sprachgeschichte* 5 (1): 150–164. doi:10.1515/jbgsg-2014-0011.

Schulz, Klaus, and Stoyan Mihov. 2001. *Fast String Correction with Levenshtein-Automata.* Technical report CIS-Report 01-127. CIS, Universität München. Accessed October 2, 2021. https://www.cis.uni-muenchen.de/publikationenall/cis_berichte/cis-01-127.html.

———. 2002. "Fast String Correction with Levenshtein-Automata." *International Journal of Document Analysis and Recognition* 5:67–85. doi:10.1007/s10032-002-0082-8.

Schulz, Sarah. 2018. "The Taming of the Shrew – Non-Standard Text Processing in the Digital Humanities." PhD diss., University of Stuttgart. doi:10.18419/opus-9685.

Schulz, Sarah, and Nora Ketschik. 2019. "From 0 to 10 Million Annotated Words: Part-of-Speech Tagging for Middle High German." In "Language Technology for Digital Humanities," edited by Erhard Hinrichs, Marie Hinrichs, Sandra Kübler, and Thorsten Trippel, *Language Resources and Evaluation* 53 (4): 837–863. doi:10.1007/s10579-019-09462-8.

Serban, Iulian Vlad, Alessandro Sordoni, Ryan Lowe, Laurent Charlin, Joelle Pineau, Aaron Courville, and Yoshua Bengio. 2017. "A Hierarchical Latent Variable Encoder-Decoder Model for Generating Dialogues." In *Proceedings of the Thirty-First AAAI Conference on Artificial Intelligence,* 3295–3301. San Francisco, California, USA: AAAI Press, February. Accessed October 2, 2021. http://aaai.org/ocs/index.php/AAAI/AAAI17/paper/download/14567/14219.

Siewert, Janine, Yves Scherrer, and Jorg Tiedemann. 2021. "Towards a Balanced Annotated Low Saxon Dataset for Diachronic Investigation of Dialectal Variation." In *Proceedings of the 17th Conference on Natural Language Processing (KONVENS 2021).* Düsseldorf, Germany, September. Accessed October 2, 2021. https://konvens2021.phil.hhu.de/wp-content/uploads/2021/09/2021.KONVENS-1.25.pdf.

Silfverberg, Miikka, Teemu Ruokolainen, Krister Lindén, and Mikko Kurimo. 2014. "Part-of-Speech Tagging using Conditional Random Fields: Exploiting Sub-Label Dependencies for Improved Accuracy." In *Proceedings of the 52nd Annual Meeting of the Association for Computational Linguistics (Volume 2: Short Papers),* 259–264. Baltimore, Maryland: Association for Computational Linguistics, June. doi:10.3115/v1/P14-2043.

Spadini, Elena. 2019. "Exercises in Modelling: Textual Variants." *International Journal of Digital Humanities* 1:289–307. doi:10.1007/s42803-019-00023-7.

Sutton, Charles, and Andrew McCallum. 2012. "An Introduction to Conditional Random Fields." *Foundations and Trends in Machine Learning* 4 (4): 267–373. doi:10.1561/2200000013.

Tjong Kim Sang, Erik, Marcel Bollmann, Remko Boschker, Francisco Casacuberta, Feike Dietz, Stefanie Dipper, Miguel Domingo, et al. 2017. "The CLIN27 Shared Task: Translating Historical Text to Contemporary Language for Improving Automatic Linguistic Annotation." *Computational Linguistics in the Netherlands Journal* 7:53–64. Accessed October 2, 2021. https://www.clinjournal.org/clinj/article/view/68.

Toutanova, Kristina, and Colin Cherry. 2009. "A Global Model for Joint Lemmatization and Part-of-Speech Prediction." 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,* 486–494. Suntec, Singapore: Association for Computational Linguistics, August. Accessed October 2, 2021. https://aclanthology.org/P09-1055.

van Halteren, Hans, and Margit Rem. 2013. "Dealing with Orthographic Variation in a Tagger-Lemmatizer for Fourteenth Century Dutch Charters." *Language Resources and Evaluation* 47 (4): 1233–1259. doi:10.1007/s10579-013-9236-1.

Wagner, Robert A., and Michael J. Fischer. 1974. "The String-to-String Correction Problem." *Journal of the ACM* 21 (1): 168–173. doi:10.1145/321796.321811.

Wegera, Klaus-Peter. 2000. "Grundlagenprobleme Einer Mittelhochdeutschen Grammatik." In *Sprachgeschichte. Ein Handbuch zur Geschichte der deutschen Sprache und ihrer Erforschung,* 2. vollst. neubearb. und erw. Aufl., edited by Werner Besch, Anne Betten, Oskar Reichmann, and Stefan Sonderegger, bk. 2, 1304–1320. Handbücher zur Sprach- und Kommunikationswissenschaft. Berlin, Boston: De Gruyter.

Wegera, Klaus-Peter, Sandra Waldenberger, and Ilka Lemke. 2018. *Deutsch Diachron. Eine Einführung in den Sprachwandel des Deutschen.* 2., neu bearbeitete Auflage. Grundlagen Der Germanistik 52. Berlin: Erich Schmidt Verlag.

Weißweiler, Leonie, and Alexander Fraser. 2018. "Developing a Stemmer for German Based on a Comparative Analysis of Publicly Available Stemmers." In *Language Technologies for the Challenges of the Digital Age: Proceedings of the International Conference of the German Society for Computational Linguistics and Language Technology, Berlin, Germany, September 2017,* edited by Georg Rehm and Thierry Declerck, 81–94. Lecture Notes in Computer Science 10713. Cham: Springer International Publishing. doi:10.1007/978-3-319-73706-5_8.

Witten, Ian H., Eibe Frank, Mark A. Hall, and Christopher J. Pal. 2017. *Data Mining: Practical Machine Learning Tools and Techniques.* 4th ed. Cambridge, MA, USA: Morgan Kaufmann. doi:10.1016/C2015-0-02071-8.

Yang, Yi, and Jacob Eisenstein. 2016. "Part-of-Speech Tagging for Historical English." In *Proceedings of the 2016 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies,* 1318–1328. San Diego, California: Association for Computational Linguistics, June. doi:10.18653/v1/N16-1157.

Yujian, Li, and Liu Bo. 2007. "A Normalized Levenshtein Distance Metric." *IEEE Transactions on Pattern Analysis and Machine Intelligence* 29 (6): 1091–1095. doi:10.1109/TPAMI.2007.1078.

Datasets

Klein, Thomas, Klaus-Peter Wegera, Stefanie Dipper, and Claudia Wich-Reif. 2016. *Referenzkorpus Mittelhochdeutsch (1050–1350), Version 1.0, https://www.linguistics.ruhr-uni-bochum.de/rem/.* ISLRN 332-536-136-099-5.

ReN-Team. 2016. *Referenzkorpus Mittelniederdeutsch/Niederrheinisch (1200–1650). Archived in Hamburger Zentrum für Sprachkorpora. Version 0.1. Publication date 2016-08-23.* http://hdl.handle.net/11022/0000-0001-B002-5.

———. 2017. *Referenzkorpus Mittelniederdeutsch/Niederrheinisch (1200–1650). Archived in Hamburger Zentrum für Sprachkorpora. Version 0.3. Publication date 2017-06-15.* http://hdl.handle.net/11022/0000-0006-473B-9.

———. 2018. *Referenzkorpus Mittelniederdeutsch/Niederrheinisch (1200–1650). Archived in Hamburger Zentrum für Sprachkorpora. Version 0.6. Publication date 2018-03-07.* http://hdl.handle.net/11022/0000-0007-C64C-5.

Appendix A.

Publication Index

This is a selection of publications I (co-)authored while working on this dissertation. They are all related to the topic of spelling variation and non-standard texts.

A.1. Main Publications

These are the publications this thesis is based on. The research described in these papers was carried out in its entirety by myself. The other authors of the publications acted as advisors.

Barteld, Fabian. 2017. "Detecting Spelling Variants in Non-Standard Texts." In *Proceedings of the Student Research Workshop at the 15th Conference of the European Chapter of the Association for Computational Linguistics,* 11–22. Valencia, Spain: Association for Computational Linguistics, April. Accessed October 2, 2021. https://aclanthology.org/E17-4002.

Barteld, Fabian, Chris Biemann, and Heike Zinsmeister. 2018. "Variations on the Theme of Variation: Dealing with Spelling Variation for Fine-Grained POS Tagging of Historical Texts." In *Proceedings of the 14th Conference on Natural Language Processing (KONVENS 2018),* edited by Hanno Biber, Christina Katsikadeli, and Manfred B. Sellner, 202–212. Vienna, Austria: Verlag der Österreichischen Akademie der Wissenschaften, September. doi:10.1553/0x003a12bd.

———. 2019. "Token-based Spelling Variant Detection in Middle Low German Texts." In "Language Technology for Digital Humanities," edited by Erhard Hinrichs, Marie Hinrichs, Sandra Kübler, and Thorsten Trippel, *Language Resources and Evaluation* 53 (4): 677–706. doi:10.1007/s10579-018-09441-5.

The following publications were published before the official start of my dissertation but contain research relevant to the dissertation:

Barteld, Fabian, Ingrid Schröder, and Heike Zinsmeister. 2015. "Unsupervised Regularization of Historical Texts for POS Tagging." In *Proceedings of the Workshop on Corpus-Based Research in the Humanities (CRH),* 3–12. Warsaw, Poland, December. Accessed October 2, 2021. `http://crh4.ipipan.waw.pl/files/9814/4973/5451/CRH4_proceedings.pdf`.

———. 2016. "Dealing with Word-Internal Modification and Spelling Variation in Data-Driven Lemmatization." In *Proceedings of the 10th SIGHUM Workshop on Language Technology for Cultural Heritage, Social Sciences, and Humanities,* 52–62. Berlin, Germany: Association for Computational Linguistics, August. doi:`10.18653/v1/W16-2106`.

A.2. Other Related Publications

These are publications about different aspects of creating the ReN that I co-authored while working on this dissertation.

Barteld, Fabian, Ingrid Schröder, and Heike Zinsmeister. 2016. "$_t\gamma$ – Inter-Annotator Agreement for Categorization with Simultaneous Segmentation and Transcription-Error Correction." In *Proceedings of the 13th Conference on Natural Language Processing (KONVENS 2016),* 27–37. Bochumer Linguistische Arbeitsberichte 16. Bochum, Germany, September. Accessed October 2, 2021. `https://www.linguistics.rub.de/konvens16/pub/4_konvensproc.pdf`.

Barteld, Fabian, Katharina Dreessen, Sarah Ihden, and Ingrid Schröder. 2017. "Das Referenzkorpus Mittelniederdeutsch/Niederrheinisch (1200–1650) – Korpusdesign, Korpuserstellung und Korpusnutzung." In "Mittelniederdeutsche Literatur," edited by Anja Becker and Albrecht Hausmann, *Mitteilungen des Deutschen Germanistenverbandes* 64 (3): 226–241. doi:`10.14220/mdge.2017.64.3.226`.

Schröder, Ingrid, Fabian Barteld, Katharina Dreessen, and Sarah Ihden. 2017. "Historische Sprachdaten als Herausforderung für die manuelle und automatische Annotation: Das Referenzkorpus Mittelniederdeutsch/Niederrheinisch (1200-1650)." *Niederdeutsches Jahrbuch. Jahrbuch des Vereins für niederdeutsche Sprachforschung* 140:43–57.

Barteld, Fabian, Sarah Ihden, Katharina Dreessen, and Ingrid Schröder. 2018. "HiNTS: A Tagset for Middle Low German." In *Proceedings of the Eleventh International Conference on Language Resources and Evaluation (LREC 2018),* 3940–3945. Miyazaki, Japan: European Language Resources Association (ELRA), May. Accessed October 2, 2021. https://www.aclweb.org/anthology/L18-1622.

Barteld, Fabian, Katharina Dreessen, Sarah Ihden, and Ingrid Schröder. 2019. "Analyse syntaktischer Phänomene mit dem Referenzkorpus Mittelniederdeutsch / Niederrheinisch (1200–1650)." In "Historische Korpuslinguistik," edited by Renata Szczepaniak, Stefan Hartmann, and Lisa Dücker, *Jahrbuch für Germanistische Sprachgeschichte* 10 (1): 261–281. doi:10.1515/jbgsg-2019-0015.

Appendix B.

SpellvarDetection

An open-source Python implementation of the different approaches for simplification and spelling variant detection presented in this thesis is available at `https://github.com/fab-bar/SpellvarDetection`. It allows users to create and apply pipelines for spelling variant detection using the different presented approaches for generating and filtering spelling variant candidates for a given type or token.

For the SVM-filter and bagging, the Python libraries Scikit-learn (Pedregosa et al. 2011) and Imbalanced-learn (Lemaitre, Nogueira, and Aridas 2017) are used. The CNN-filter is implemented using Keras (Chollet 2015).

After installing the package and its dependencies, type-based spelling variant detection can be performed using the command `spellvardetection generate`. This command takes a list of types for which to generate spelling variants as JSON[42] array, a description of the pipeline as JSON object, and—if needed by the pipeline—a dictionary of types from which to generate the candidates. The command returns a JSON object with the target types as keys and JSON arrays with generated spelling variant candidates as values.

A token-based filter can be applied with the command `spellvardetection filter_tokens`. This command takes a list of tokens as JSON array and a JSON object containing spelling variant candidates in the same format as the output of `spellvardetection generate`. A token is a JSON object with the keys *type*, *left_ context*, and *right_ context*, which contains the type as a string and the left respectively the right context as lists of strings. The object can contain additional information, e. g., the corpus from which the token is extracted. The output is the list of tokens with the type-based spelling variant candidates under the key *candidates* and the filtered spelling variant candidates under the key *filtered_ candidates*.

A more detailed description of how to use the tool can be found at `https://spellvardetection.readthedocs.io`. In this chapter, we give an overview of the JSON description of the different parts of the pipelines used for the experiments in Chapter 7.

42. `https://www.json.org`, last visited October 4, 2021.

B.1. Type-Based Pipelines

B.1.1. Generators

Edit Distance

The Levenshtein distance can be used with the following settings:

```
{
  "type": "levenshtein",
  "options": {
    "max_dist": 1
  }
}
```

The additional edit operations introduced in Section 7.1.2 can be used by adding the respective options to this generator. Adding transpositions and repetitions to the Levenshtein distance and using it with a distance of 1 for generating candidates can be done with the following options:

```
{
  "type": "levenshtein",
  "options": {
    "max_dist": 1,
    "transposition": true,
    "repetitions": true,
    "merge_split": false
  }
}
```

In order to use different distance thresholds depending on the length of the given type, use the following generator (here given with the options for $ed_{T,Re}$ with a maximum distance of 5 which speeds up computation for long types without losses in recall in our experiments):

```
{
  "type": "levenshtein_normalized",
  "options": {
    "dist_thresh": 0.2,
    "max_dist": 5,
    "no_zero_dist": true,
    "transposition": true,
    "repetitions": true,
    "merge_split": false
  }
}
```

N-Gram Similarity

The n-gram similarity measures described in Section 7.1.1 can be used by using either *jaccard* for the Jaccard Index, *proxinette* for Proxinette or *frequency_wjaccard* for the Jaccard Index with frequency weighting for the features as *type* of the generator .

The types of n-grams can be changed by adding an option *feature_extractor* which takes the description of the n-gram extractor, which allows setting parameters like the lengths of the n-grams and potential skips.

The best settings for the Jaccard Index with frequency-weighted features according to F_1 in the text-eval and the OOV-eval settings (cf. Table 7.5 on page 90) are given with the following two settings:

```
{
  "type": "frequency_wjaccard",
  "options": {
    "sim_thresh": 0.35,
    "feature_extractor": {
      "type": "ngram",
      "options": {
        "min_ngram_size": 2,
        "skip_size": 4,
        "pad_ngrams": true
      }
    }
  }
}
```

```
{
  "type": "frequency_wjaccard",
  "options": {
    "sim_thresh": 0.25,
    "feature_extractor": {
      "type": "ngram",
      "options": {
        "min_ngram_size": 4,
        "skip_size": 4,
        "pad_ngrams": true
      }
    }
  }
}
```

Lexicon-based lookup

The generator type *union* can be used to add known variants to the generated variants as described in Section 7.2.1. It takes a list of generators as an option. The following generator adds *unde* to the spelling variant candidates generated using a Levenshtein distance of 1:

```
{
  "type": "union",
  "options": {
    "generators": [
      {
        "type": "levenshtein",
          "options": {
            "max_dist": 1
          }
        },
        {
          "type": "lookup",
            "options": {
              "spellvar_dictionary": {"vnd": ["unde"] }
        }
      }
    ]
  }
}
```

Simplification

Rule-based simplification for generating spelling variant candidates as described in Section 7.2.1 can be used with the generators *gent_gml_simplification*—which implements KOL—and *simplification*—which allows specifying corresponding characters as described in Chapter 6 with the option *ruleset*. Both take the option *generator* to define a generator that is applied after simplifying all types. The following example uses a simplification in combination with a $\text{ed}_{T,Re}$ candidate generation which essentially leads to a weighted edit distance, where the substitutions $u \leftrightarrow v$ and $i \leftrightarrow j$ have a cost of 0:

```
{
  "type": "simplification",
  "options": {
    "ruleset": [["u", "v"], ["i", "j"]],
```

```
    "generator": {
      "type": "levenshtein_normalized",
      "options": {
        "dist_thresh": 0.2,
        "max_dist": 5,
        "no_zero_dist": true,
        "transposition": true,
        "repetitions": true,
        "merge_split": false
      }
    }
  }
}
```

The ruleset used for S+ed$_{T,Re}$ is given by the following correspondences:

```
[
  ["ij", "i"], ["s", "z"], ["i", "j"], ["ff", "f"],
  ["j", "y"], ["a\u0364", "a"], ["f", "v"], ["gh", "g"],
  ["u", "v"], ["ij", "y"], ["sz", "s"], ["th", "t"]
]
```

B.1.2. Filters

For the filters, the *type pipeline* has to be used. The filter is defined in the field *type_filter*. A generator has to be given in the field *generator*. The generator can either be defined directly or by giving the name of a file that contains the definition. Here, any filter as described above or even a full pipeline can be used.

The type filters using the undirected SPSim and the (B)SVM need to be trained using the command `spellvardetection train filter`, which takes the definition of the filter as well as negative and positive pairs and a filename for the resulting model.

Brown Filter

In order to use Brown clusters to filter generated candidates, the *type_filter* with the *type cluster* can be used. *cluster_ type brown* allows using the output of `https://github.com/percyliang/brown-cluster` (last visited October 4, 2021) to define the clusters.

```
{
  "type": "pipeline",
  "options": {
    "generator": "generator_definition.json",
    "type_filter": {
      "type": "cluster",
      "options": {
        "cluster_type": "brown",
        "cluster_file": "path/to/clusters/paths"
      }
    }
  }
}
```

UndirSPSim

Calling the command `spellvardetection train filter` with the filter type *uspsim* extracts the undirected SPs from the training data. The resulting file needs to be specified when applying the filter:

```
{
  "type": "pipeline",
  "options": {
    "generator": "generator_definition.json",
    "type_filter": {
      "type": "uspsim",
      "options": {
        "spsim_filename": "file_with_sps",
        "sim_thresh": 1
      }
    }
  }
}
```

Edit Probabilities

Edit probabilities can be learned from known spelling variants with the command `spellvardetection learn edit_probabilities`. The command takes the known spelling variants and outputs the edit probabilities. The resulting file needs to be specified when applying the filter:

```
{
  "type": "pipeline",
  "options": {
    "generator": "generator_definition.json",
    "type_filter": {
      "type": "edit_probabilities",
      "options": {
        "probabilities": "file_with_probabilities",
        "sim_thresh": 0.5
      }
    }
  }
}
```

SVM

The type filter *sklearn* allows to use models for binary classification trained with scikit-learn. It takes the classname of the classifier and settings for hyperparameters as options for training. For the (B)SVM filter, the special names *__svm__* and *__bagging_svm__* can be used. Using these options, the hyperparameters C and *gamma* are set to the best values from our experiments, 2 and 10^{-1}, by default.

The used features are set by defining feature extractors and setting the respective options. The type *surface* extracts alignment n-grams from the pairs of types, by default using the lengths 1, 2 and 3 and only n-grams around mismatches. The type *context* uses word embeddings to extract the contextual similarity for the types, it needs a file with embeddings as option. *vector_type hyperwords* allows to use the output of *hyperwords* which can be used to create embeddings using PPMI-SVD as well as SGNS.[43]

```
{
  "type": "sklearn",
  "options": {
    "classifier_clsname": "__bagging_svm__",
    "feature_extractors": [
      { "type": "surface" },
      { "type": "context",
        "options": {
          "vector_type": "hyperwords",
```

43. Note that *hyperwords* is no longer available at `http://bitbucket.org/omerlevy/hyperwords` (last visited October 4, 2021), the URL given in Levy, Goldberg, and Dagan (2015).

```
            "vectorfile_name": "file_with_embeddings"
          }
        }
      ]
    }
}
```

For applying the filter, only the *modelfile_name* has to be given as option.

B.2. Token-Based Filter

The token-based filter presented in Section 7.2.3 can be trained with the command `spellvardetection train token_filter` which takes the description of the filter, the name of the file in which the trained model is saved, training data in the form of a list of tokens and a spelling variant dictionary. The following description trains the CNN filter with the settings that have led to the best results for the text-eval settings:

```
{
  "type": "cnn",
  "options": {
    "left_context_len": 1,
    "right_context_len": 1,
    "vector_type": "hyperwords",
    "vectorfile_name": "file_with_embeddings"
  }
}
```

For applying the filter, the command `spellvardetection filter_tokens` has to be used, which takes a list of tokens, a dictionary with spelling variant candidates and the description of the filter as arguments. For the description of the filter only the *modelfile_name* has to be given.

References

Chollet, François, et al. 2015. *Keras.* Accessed October 2, 2021. `https://keras.io`.

Lemaitre, Guillaume, Fernando Nogueira, and Christos K. Aridas. 2017. "Imbalanced-learn: A Python Toolbox to Tackle the Curse of Imbalanced Datasets in Machine Learning." *The Journal of Machine Learning Research* 18 (1): 559–563. Accessed October 2, 2021. `https://dl.acm.org/doi/10.5555/3122009.3122026`.

Levy, Omer, Yoav Goldberg, and Ido Dagan. 2015. "Improving Distributional Similarity with Lessons Learned from Word Embeddings." *Transactions of the Association for Computational Linguistics* 3:211–225. doi:10.1162/tacl_a_00134.

Pedregosa, Fabian, Gaël Varoquaux, Alexandre Gramfort, Vincent Michel, Bertrand Thirion, Olivier Grisel, Mathieu Blondel, et al. 2011. "Scikit-learn: Machine Learning in Python." *The Journal of Machine Learning Research* 12:2825–2830. Accessed October 2, 2021. https://dl.acm.org/doi/10.5555/1953048.2078195.

Eidesstattliche Versicherung

Hiermit erkläre ich an Eides statt, dass ich die vorliegende Dissertationsschrift selbst verfasst und keine anderen als die angegebenen Quellen und Hilfsmittel benutzt habe.

Hamburg, den ....
(Unterschrift)