Lang Resources & Evaluation (2019) 53:677–706
https://doi.org/10.1007/s10579-018-09441-5

ORIGINAL PAPER

Token-based spelling variant detection in Middle Low German texts

Fabian Barteld[1] · **Chris Biemann**[2] · **Heike Zinsmeister**[1]

Published online: 9 February 2019

Abstract In this paper we present a pipeline for the detection of spelling variants, i.e., different spellings that represent the same word, in non-standard texts. For example, in Middle Low German texts *in* and *ihn* (among others) are potential spellings of a single word, the personal pronoun 'him'. Spelling variation is usually addressed by normalization, in which non-standard variants are mapped to a corresponding standard variant, e.g. the Modern German word *ihn* in the case of *in*. However, the approach to spelling variant detection presented here does not need such a reference to a standard variant and can therefore be applied to data for which a standard variant is missing. The pipeline we present first generates spelling variants for a given word using rewrite rules and surface similarity. Afterwards, the generated types are filtered. We present a new filter that works on the token level, i.e., taking the context of a word into account. Through this mechanism ambiguities on the type level can be resolved. For instance, the Middle Low German word *in* can not only be the personal pronoun 'him', but also the preposition 'in', and each of these has different variants. The detected spelling variants can be used in two settings for Digital Humanities research: On the one hand, they can be used to facilitate searching in non-standard texts. On the other hand, they can be used to improve the performance of natural language processing tools on the data by reducing the number of

✉ Fabian Barteld
fabian.barteld@uni-hamburg.de

Chris Biemann
biemann@informatik.uni-hamburg.de

Heike Zinsmeister
heike.zinsmeister@uni-hamburg.de

[1] Institut für Germanistik, Universität Hamburg, Überseering 35, Postfach #15, 22297 Hamburg, Germany

[2] Department of Informatics, Language Technology Group, Universität Hamburg, Vogt-Kölln-Straße 30, 22527 Hamburg, Germany

unknown words. To evaluate the utility of the pipeline in both applications, we present two evaluation settings and evaluate the pipeline on Middle Low German texts. We were able to improve the F1 score compared with previous work from 0.39 to 0.52 for the search setting and from 0.23 to 0.30 when detecting spelling variants of unknown words.

Keywords Spelling variation · Non-standard language · Historical texts · Information retrieval

1 Introduction

Digital humanities (DH) has brought the challenges posed for language processing tools when used on non-standard language varieties such as diachronic texts into the focus of computational linguistics (Dipper et al. 2013).[1] This paper is concerned with a particular property of such non-standard texts, namely spelling variation—the phenomenon in which one and the same word is spelled in different ways. This variation is characteristic of historical texts that predate a standardized orthography. We present experiments with texts from Middle Low German (GML),[2] a group of historical dialects of German from the 15th to the 17th century, cf. example (1).[3]

(1) a. in dem seuenden *steit*
in the seventh is_indicated
b. JN dem elenboghen *steyt*
in the elbow is_indicated
(Brem. Ssp.)

A typical way of dealing with spelling variation is normalization, also called standardization or canonicalization (Piotrowski 2012). Here, every given type of a non-standard language is mapped to an equivalent type in a corresponding standard language. In example (1), the types *steit* and *steyt* could be mapped to the corresponding word in Modern Standard German, *steht*. This mapping serves two purposes: First, NLP tools developed for the standard variant will have a decent performance on the non-standard data as well. For instance, a part of speech (POS) tagger developed for Modern Standard German will most likely assign the correct POS tag to *steht* while *steyt* probably is a previously unseen word for the tagger, so that the tagger has to guess and is more likely to assign an incorrect POS tag. Second, when searching in non-standard texts or when training statistical NLP tools on such texts themselves, normalization reduces variation in the data and provides

[1] See also the workshop series on Language Technology for Cultural Heritage, Social Sciences, and Humanities (LaTeCH) by ACL SIGHUM (https://sighum.wordpress.com).

[2] Following ISO 639-3, we use GML as the abbreviation for Middle Low German in this paper.

[3] In the whole paper, we will ignore differences in capitalization. All types are lowercased before applying and evaluating our pipeline.

more robust results. For instance, the spelling variants *in* and *jn* (ignoring the difference in capitalization) from example (1) would both be mapped to the same form that could be used when searching for or training on the preposition *in*.

Note that for searching in non-standard data or for training an NLP tool on it, it is not required that a standard form exists. It is sufficient that variant sets can be identified. An illustrative example for this is given by the NLP application described in Hogenboom et al. (2015). As a preprocessing step for polarity classification, emojis are mapped to "groups of emoticons denoting the same emotion". In an analogous manner, *steit* and *steyt* would be mapped to a set of spelling variants $\{steit, steyt, \ldots\}$.

In this paper, we follow such an approach and explore techniques to deal with spelling variation without reference to a standard variant. This makes the approach usable for non-standard languages that do not have a closely related standard variant. The main contribution of this paper is to introduce a new pipeline for detecting spelling variants in a data-driven way by first generating a set of spelling variant candidates and then filtering this candidate set. The filtering is done on the type level and on the token level, i.e., taking context information into account. This is the first approach for spelling variant detection in the absence of a standard target language that works on the token level.

For candidate generation, we compare different techniques of measuring the distance between candidates based on a modified Levenshtein distance and combine them with a rule-based approach to reduce spelling variation. We employ Support Vector Machine (SVM) classifiers for type-level filtering and introduce a neural network architecture for token-level filtering. The distinction between generation, type-level, and token-level filters, allows one to choose between recall and precision: one can either use all the generated candidates for a high recall or apply one or both filters for higher precision at the cost of recall. This allows a researcher in the humanities to adapt the pipeline for her needs by simply leaving out some of the steps.

We approach the effects of spelling variation from two different perspectives that are relevant for digital humanities: From an *information retrieval* (IR) perspective, spelling variation leads to a low recall when searching for a specific word unless the user knows the possible variants when searching. This makes searching in non-standard texts challenging. The presented pipeline can facilitate searching by generating spelling variants for the query term. From a *natural language processing* (NLP) perspective, spelling variation leads to a situation in which training information that actually belongs to one and the same word is distributed over its different spelling variants, e.g. *steit* and *steyt* in (1). Furthermore—and this is more relevant in the context of this paper—spelling variation also leads to a high number of unknown words when applying a trained supervised machine learning 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 for unknown words when applying an NLP tool. To estimate the usefulness of our pipeline for both applications, we introduce and use two evaluation settings that capture relevant aspects of these tasks. We use a corpus of GML texts to evaluate the techniques and report micro-averaged precision, recall, and the harmonic mean between both values (F1) on the token level.

First, we present a definition of spelling variation and the different approaches to deal with it (Sect. 2) and a discussion of related work (Sect. 3). After that we describe the data sets (Sect. 4) and the evaluation methods (Sect. 5) in detail. In Sect. 6, we will present our pipeline to detect spelling variants, which will be further detailed in the following sections. For the pipeline we explore the generation of spelling variant candidates by simple lookup (Sect. 7) and by surface similarity (Sect. 8). The generated candidates are subsequently filtered first on the type level and then on the token level with a supervised machine learning setup (Sect. 9). We conclude with an overview of the results, some error analysis, and an outlook for further work (Sects. 10 and 11).

2 Defining and quantifying spelling variation

Spelling variation is usually approached in the context of an existing standard. As mentioned before, normalization is the predominant approach for dealing with non-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). 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 query term.

If no closely related standard language is available—as it is the case with Middle Low German, as Modern Low German does not have a standardized orthography either—these approaches cannot be applied in a straightforward manner. The alternative that is pursued in this paper is to generate the set of variants for a given token instead of a canonical cognate such that no reference to a standard is needed. To clarify the similarities and differences between normalization and historical variant detection on the one hand and spelling variant detection on the other hand, we present formal definitions for spelling variation and the related tasks in this section.

To define what constitutes a standard language,[4] we distinguish between *types* and *morphological words* (Barteld 2017): *Types* are the surface forms of a language, i.e., elements of Σ^* for some given alphabet Σ, *morphological words* are the abstract entities that are instantiated by types. They can be seen as combinations of a lemma (word-sense disambiguated), a POS, and a morphological description, e.g. *{Personal Pronoun, Nominative Sg., I}*.

Given the language $L \subseteq \Sigma^*$ and a set of morphological words L_{morph}, we define the left- and right-total binary relation $S \subseteq L_{\text{morph}} \times L$ that relates *morphological words* with their potential spellings. If S is a function, i.e., for each *morphological*

[4] In this paper, we are only concerned with spelling variation and therefore ignore other aspects of standard and non-standard languages.

word there exists only one *type* as a potential spelling, we say that L is a standard language, otherwise we call L a non-standard language. In general such a function is not injective due to ambiguities, e.g. case syncretisms.

Given a corpus $C = (t_i)_1^n$ in which each token is annotated with POS, morphological information and lemma, i.e., each t_i is a tuple (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 $(w, p, m, l) \in C$. In general S' will only approximate S. We will discuss implications of this in Sect. 5.

Given a set of morphological words L_{morph}, a non-standard language L and a standard language L_s with the corresponding spelling relations S_L and S_{L_s}, we can define (type-based) normalization, non-standard variant and spelling variant detection more precisely:

– Normalization is the function $n : L \to \mathcal{P}(L_s)$ 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 $e : L_s \to \mathcal{P}(L)$ 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 $s : L \to \mathcal{P}(L)$ with:

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

Normalization and non-standard variant detection are closely related, as the following definitions demonstrate. They show how a non-standard variant detection can be induced given a normalization and the other way round:

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

Furthermore, a normalization induces a spelling variant detection:

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

However, while normalization and non-standard variant detection need to refer to a language L_s, spelling variant detection does not need such a reference as can be seen easily with the definitions given above.

So far, we have only defined type-based variants of the three approaches. The token-based variants operate on tokens, i.e., 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. For non-standard and spelling variant detection, the token-based versions will return subsets of the type-based version, as some of the potential variants of the given type can be excluded given a specific context.

S can also be used to quantify the amount of spelling variation in L, e.g. simply as the number of *morphological words* that have more than one spelling:[5]

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

We want to use this quantification of spelling variance to make another distinction. For many applications it is not necessary that the target language is an existing standard language, it suffices if L_s exhibits less variation than the non-standard language. Just such an operation is commonplace for Twitter data: the removal of repeated characters as a preprocessing step (Han et al. 2013). This reduces variation because, for example, *loool* and *looooool* are both mapped to *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*, *looooool*, and *lol* to *laughing out loud*.

More generally, a mapping r from a language L to a language L^* is a simplification if it reduces variation.[6] In this sense, normalization is a special kind of simplification where L^* is an existing standard language.[7] As mentioned above, 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, this 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 search results for all types from L that are mapped to $r(s)$ as well, i.e., $r^{-1}(r(s))$, where r^{-1} denotes the preimage of r, improving the recall of the search. From the viewpoint of natural language processing (NLP), L^* will have less variation which results in less data sparsity and unknown words. Like normalizations, simplifications also induce spelling variant detections. As simplifications do not need the reference to a standard language, this is one way to achieve spelling variant detection in the absence of a standard language.

3 Related work

In this section we present several approaches for automatic normalization, non-standard variant detection, and spelling variant detection that share similarities to the techniques that we apply in this paper. While these approaches can be grouped

[5] When L, L_{morph}, and S are induced from a corpus, we have used a slightly different definition (Barteld 2017): v can be given as a ratio, as L_{morph} is finite. For this, we excluded morphological words, that are instantiated only once in the corpus as they cannot exhibit possible variance.

[6] Simplification does not mean that the resulting types are simpler in the sense that they are shorter. The addition of a h after every g that is not already followed by one would also be an example of a simplification.

[7] 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.

by their underlying concept of normalization (e.g. normalization as error correction vs. translation, Kobus et al. 2008), we present the selected approaches according to their basic methods. First, there are rule-based approaches. The rules can either be defined manually (Jurish 2010a; Pettersson et al. 2012) or learned from pairs from the source and the target language (Bollmann et al. 2011). Second, there are similarity-based approaches, using surface similarity (Pettersson et al. 2013a) or context similarity (Costa Bertaglia and Volpe Nunes 2016). Since similarity measures often overgenerate and yield similar types that are not spelling variants, Barteld (2017) trained a supervised classifier to filter such results that were obtained using similarity measures when detecting spelling variants. Rule-based and similarity-based approaches are often combined, e.g. by using the similarity-based approach as a backoff for the rule-based approach (Bollmann 2012).

Other approaches to normalization that have less in common with the approaches employed in this paper include tagging approaches, where each character of a non-standard word is tagged either with a sequence of edit operations (Chrupała 2014) or with the corresponding characters in the standard word (Bollmann and Søgaard 2016), and character-based machine translation, either statistical (Pettersson et al. 2013) or neural (Bollmann et al. 2017).

Another important aspect that distinguishes different normalization approaches is whether they are working on the type or on the token level. Approaches that work on the type level will always treat one type the same regardless of its actual usage in the text, while approaches that operate on the token level can treat the same type differently based on the context. For example, in GML the token *in* can be the preposition 'in' as in example (1) but it can also be the personal pronoun 'him', which should not be conflated with the preposition. While the approaches presented above all work on the type level, Jurish (2010b) proposes a Hidden Markov Model to select the best normalization from a set of normalization candidates for each token in a sequence of historical tokens. The candidates are generated by different methods (e.g. the token itself and rule-based transliteration) that work on the type level to generate normalization candidates. Ljubešić et al. (2016) compare character-based statistical machine translation, applied either to the the tokens individually, i.e., on the type level, or a whole sequence, i.e., token context is taken into account. In contrast to the token-based approach presented in this paper, these two approaches make use of a target language and a language model for this target language.

One benefit—and common use case—of normalization is that when a non-standard text is normalized, it can be processed by tools developed for the standard language variety (Tjong Kim Sang et al. 2017). Bollmann (2013) and Derczynski et al. (2013) among others have demonstrated the utility of normalization for annotation. However, lately the limits of this approach have been emphasized (Yang and Eisenstein 2016; van der Goot et al. 2017): While normalization helps to achieve better results when applying NLP tools, not all phenomena of the non-standard data can be covered—therefore other approaches like domain adaptation should be combined with normalization to improve the tagging accuracy. Furthermore, Koleva et al. (2017) report that a tagger trained on Modern Standard German tagged every GML token as "foreign material" despite normalization.

Hence for this data, training a tagger on GML texts is better suited. Along this line, another branch of research has looked at training taggers directly on non-standard data and using information about spelling variants when training and/or applying these NLP tools (Kestemont et al. 2010; Logačev et al. 2014; Barteld et al. 2015, 2016; Adesam and Bouma 2016). While Adesam and Bouma (2016) use a simplified form for tagging, the other approaches rely only on the sets of spelling variants to improve tagging and lemmatization accuracy, respectively. Therefore, these approaches can make use of a spelling variant detection tool such as the one presented in this paper.

Another use-case for normalization is searching in historical texts (Jurish et al. 2014). By normalizing, searching the texts is simplified, especially for non-expert users who are able to query the data using contemporary language. However, only knowing the spelling variants of a given query term is enough to improve the recall when searching documents that contain spelling variation. Pilz et al. (2006), Ernst-Gerlach and Fuhr (2006), Hauser and Schulz (2007) present approaches to search in historical texts by generating non-standard variants for a given search term. The systems are evaluated by testing the recall for given Modern Standard German search terms. Similarly, the pipeline presented in this paper can be used in this case as well.

4 Description of the data

For our experiments with historical data, we use the third pre-release of the Reference Corpus Middle Low German/Low Rhenish (1200–1650) (ReN 0.3).[8] This version 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. 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 3991 types, 1959 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 3063 types, 1681 of them do not appear in the training set, and again 623 have spelling variants in the training set. This means that by mapping all unknown or OOV (out of vocabulary) 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.

5 Evaluation

In this section, we introduce two evaluation settings that we will employ throughout the rest of the paper. They are inspired by the needs of *information retrieval* and *natural language processing* as discussed in Sect. 1.

[8] Version 0.3. Publication date 2017-06-15. http://hdl.handle.net/11022/0000-0006-473B-9.

Before we illustrate the two settings with an example, we need to clarify the input and output of our pipeline for spelling variant detection: Given a lexicon L_T, which contains the types that appear in textual data T, and some token t, i.e., a type which may or may not occur in T and its context, the pipeline produces a list of all types of L_T that are spelling variants for t.

For our example, the GML data is split into training, development and test sets as described in the previous section. From these sets, we extract three lexicons L_{train}, L_{devel}, and L_{test}, each containing the types that appear in the respective set of the data.

Both evaluation settings evaluate the pipeline on tokens from the test (respectively the development) set. Which lexicon is used for producing 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 1 shows their distribution over the data sets. L_{GML}, the unknown complete dictionary of GML, is approximated by the complete dataset, $L_{train} \cup L_{develop} \cup L_{test}$. The figure illustrates the situation when applying the pipeline on the test set, therefore the development set is not marked explicitly.

Detection of spelling variants in texts (text-eval) The first evaluation setting evaluates, for each token in the test set, the identification of the spelling variants in the same set, i.e., in L_{test}. It is motivated by the search scenario for which we assume

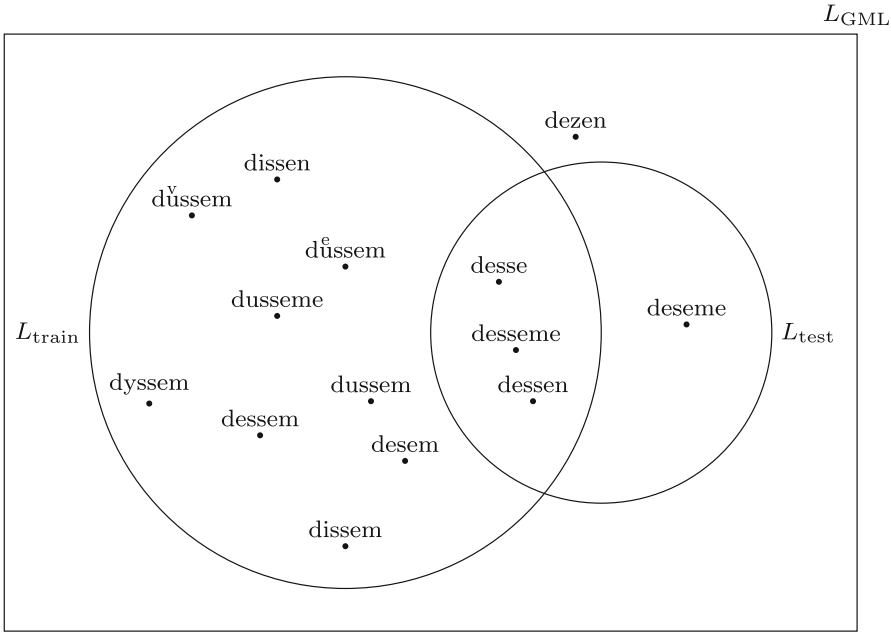

Fig. 1 Spelling variants of *desse, dit* in ReN 0.3

that the aim is to search in data that is distinct from the data that was used to train the pipeline. In particular, it determines for each occurrence of *desse*, *desseme*, *dessen*, and *deseme* in the test set the spelling variants that appear in this set (i.e., a subset of *desse*, *desseme*, *dessen*, and *deseme*).

Detection of known spelling variants for unknown types (OOV-eval) The second evaluation setting evaluates the mapping of OOV words in the test data 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 *deseme* as a subset of *dissen*, $d\overset{v}{u}ssem$, $d\overset{e}{u}ssem$, *dusseme*, *dyssem*, *dussem*, *dessem*, *desem*, *dissem*, *desse*, *desseme*, and *dessen*. This setting is similar to the evaluation in our previous work (Barteld 2017) with the difference that we evaluate on the token level and not on the type level.

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

A crucial requisite for evaluating which tokens should be considered as spelling variants is a proper definition of what a 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 corpus we employ a strict definition which is based on the lexical and syntactic annotation of the corpus: Two tokens with the same part-of-speech label, the same morphology, and the same lemma[9] are considered to be instances of the same word—and consequently, if the spelling differs, spelling variants.[10]

This definition has a data sparsity issue when used for evaluation: Infrequent but ambiguous types might not appear as instances of every possible word that they can stand for. 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') *gehystlich* ('clerical'), and *frage* ('(the/to) question') in Table 1.[11]

[9] The lemmatization in the corpus includes word-sense disambiguation such that homonyms are distinguished.

[10] This definition leads to a broad definition of spelling variation, as words 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.

[11] The abbreviations follow the Leipzig glossing rules (https://www.eva.mpg.de/lingua/resources/glossing-rules.php).

Table 1 Examples for spelling variants that do not appear with the same annotations in ReN 0.3

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

All of the types in one of the sections of the table are considered spelling variants by experts. However, only *vraghe* and *frage*, printed in bold in the table, are part of the extracted spelling variant relation as they are the only two types that appear with the same annotations in the texts. Though e.g. *pilatus* could appear in the nominative case, due to data sparsity it only appears in the genitive case and is therefore not considered a spelling variant of *pylatus* by our data-driven definition of spelling variants. This problem is obviously connected to low-frequency types: If they are ambiguous, they are unlikely to overlap in their grammatical function. Therefore, the actual precision of the pipeline will be higher than measured on this data set.

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 as this is the only data source that is available when training. For evaluating, we use the whole data set in order to get closer to the real spelling variant relation. This leads to a situation in which 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*]. The full corpus, however, also contains *noch*, *unnde*, and *vnn* as further variants.

The evaluation is performed on the token, not on the type level. This 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 2 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

Table 2 Ambiguity of *koning* and *koninge* in ReN 0.3

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

the latter appears in dative or—as in example (2b)—in accusative case but not in nominative case as in example (2a).

(2) 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 3 shows micro-averaged precision (P), recall (R), F1 values, and the average number of spelling variant candidates (C) in the training data when, for each token, all types that appear as spelling variants of the respective type are added. The recall is 1, however, 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).

6 A pipeline for spelling variant detection

In the following sections, we present a pipeline for spelling variant detection. This pipeline extends previous work (Barteld 2017). The general approach consists of two steps: in the first step, spelling variant candidates for a given type are generated. This candidate set is then filtered in a second step using a supervised classifier. Barteld (2017) obtained types with a low Levenshtein distance (Levenshtein 1966) to the given type from the lexicon in the generation step and used a SVM with the set of character n-grams taken from the aligned types (Ciobanu and Dinu 2014) and the cosine similarity of vector representations of the words as features for the filtering.

Table 3 Precision, recall, F1, and average number of candidates for the lookup approach (training set)

P	R	F1	C
0.32	1.00	0.49	14.38 ± 16.31

In this paper, we present an extended version of this general approach. Figure 2 gives an overview over the whole pipeline as it is presented in this paper. For a given type, spelling variant candidates are generated in two ways on the type level: By a simple lexicon lookup (Sect. 7) on the one hand and by surface similarity (Sect. 8) combined with a SVM filter on the other hand. For the surface similarity, we experiment with modified versions of the Levenhstein distance (Sect. 8.1). Furthermore, we combine this similarity-based generation with a rule-based simplification (Sect. 8.2) by first applying a simplifaction on all types before using the distance-based generation. This addresses one major issue with our previous approach, which is rooted in the low recall of the generation process: While a Levenshtein distance of 1 has a relatively low recall, it still offers a better balance between precision and recall than higher distances and led to the best overall results. For the SVM filter, we substitute the standard SVM classifier with a Bagging-SVM (BSVM) (Mordelet and Vert 2014), which is better suited for the training data (Sect. 9.1). A further addition to the pipeline presented in this paper is that candidates obtained by these two generators are combined and subsequently filtered (at the token level) by applying a CNN filter (Sect. 9.2).

An implementation of the pipeline using Python is available at https://github.com/fab-bar/SpellVarDetection.

The runtime of the candidate generation step is governed by searching in the lexicon for candidates with a Levenshtein distance lower than or equal to the given threshold. For this, we use Levenshtein automata which have a runtime that is linear with regard to the size of the dictionary (Schulz and Mihov 2002). This runtime can also be improved by incorporating filtering techniques (Mihov and Schulz 2004). For candidate filtering, the trained classifiers (SVM and CNN) have to be applied to each of the candidates which can be done in parallel.

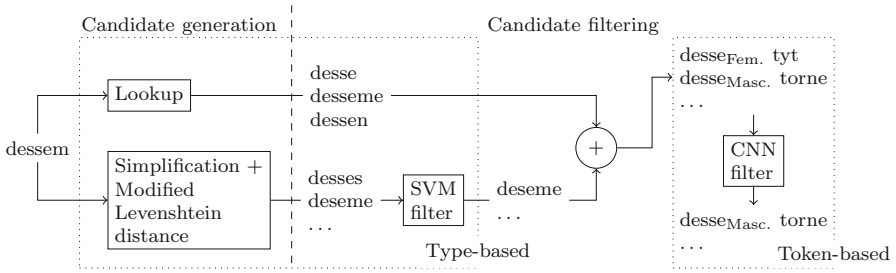

Fig. 2 Overview over the spelling variant generation pipeline

Table 4 Precision, recall, F1, and average number of candidates for the lookup approach (development set)

text-eval				OOV-eval			
P	R	F1	C	P	R	F1	C
0.36	0.74	0.49	5.68 ± 7.90	1.00	0.00	0.00	0.00 ± 0.00

The intermediate results presented in the following sections are obtained from the development set. Section 10 contains final results and an error analysis on the test set.

7 Lexicon-based variant generation

A simple approach for spelling variant detection is a lexicon-based lookup of spelling variants. The lexicon lists all known variants for each known type and is extracted from the training data. As described in Sect. 5, the approach is evaluated in two settings text-eval and OOV-eval. Table 4 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[12] and a recall of 0.

For the text-eval setting, the lexicon-based lookup is well suited for short but 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 corpus: [*ghy*, *gi*, *gij*, *gv*, *gy*, *je*]. The variant *ghy* will be especially hard to identify with an approach that is based on surface similarity, since its Levenshtein distance of 3 is greater than the length of the type itself.

All of the following approaches are combined with this lexicon-based lookup, so we will show how well these approaches generalize to spelling variants not covered by a simple lookup.

8 Candidate generation using surface similarity

8.1 Modified Levenshtein distance

As noted in Sect. 6 above, the simple Levenshtein distance is too coarse-grained and can only serve as a first approximation to spelling variant detection (Bollmann 2012). While recall is not very high with a Levenshtein distance of 1, when we increase the Levenshtein distance to 2 precision is already much lower, as can be

[12] We define precision to be 1 when there are no candidates generated as there are no falsely generated candidates.

Table 5 Precision, recall, F1, and average number of candidates for different Levenshtein distances (development set)

Dist	text-eval				OOV-eval			
	P	R	F1	C	P	R	F1	C
1	**0.22**	0.85	**0.35**	10.61 ± 10.84	**0.20**	0.27	**0.23**	2.10 ± 3.22
2	0.04	0.95	0.08	65.43 ± 56.47	0.04	0.64	0.07	25.86 ± 47.71
3	0.01	**0.97**	0.02	300.77 ± 212.50	0.01	**0.87**	0.01	216.36 ± 368.62

seen in Table 5.[13] Example (3) illustrates how the number of candidates increases dramatically with a higher Levenshtein distance. 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).

(3) *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 our pipeline, the missing granularity of the Levenshtein distance is addressed by the filters that are applied afterwards. However, previous experiments (Barteld 2017) have shown that even after applying the filter, the drop in precision when going from distance 1 to distance 2 leads to a decrease in F1 value despite the higher recall. In this section, we present experiments with two options to make the Levenshtein distance 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. The second option is to avoid a fixed distance for all types and to make the distance dependent on the type for which the candidates are obtained.

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

First, transpositions (T), *ab* → *ba*, are sometimes added to the atomic edit operations that have a cost of 1 (Damerau 1964). A second modification is the

[13] The given distances are always an upper bound. For brevity, we write distance 2 instead of the more precise ≤ 2.

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 for example in the pair *ij* and *y*.

As a third modification, we introduce a new variant of the Levenshtein distance. This variant is inspired by a common preprocessing step for social media data, i.e., the deletion of repeated characters (Han et al. 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 modified Levenshtein distance is no longer a metric in the mathematical sense as for example $d(lol, looool) = 0$.

Searching the set of all types for the neighbors of a given type is done using Levenshtein automata (Mihov and Schulz 2004). T and MS are added as described by Schulz and Mihov (2001) to the automata. 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 6 gives the results of using the modified Levenshtein 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 a reasonable 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. Hence adding transpositions and repetitions leads to the best tradeoff between recall and precision in the OOV-eval setting and a still very good tradeoff in the text-eval setting.

Regarding the usage of different distance thresholds depending on the type, one option is to normalize the Levenshtein distance by the length of one of the types (Yujian and Bo 2007). Setting a threshold $0 < t$ for this ratio between the length of

Table 6 Precision, recall, F1, and average number of candidates for different variants of the Levenshtein distances (development set)

T	MS	Re	text-eval				OOV-eval			
			P	R	F1	C	P	R	F1	C
–	–	–	**0.22**	0.85	**0.35**	10.61 ± 10.84	**0.20**	0.27	**0.23**	2.10 ± 3.22
x	–	–	**0.22**	0.85	**0.35**	10.66 ± 10.86	**0.20**	0.28	**0.23**	2.14 ± 3.29
–	x	–	0.11	0.90	0.19	23.21 ± 23.15	0.10	0.35	0.16	5.27 ± 9.66
–	–	x	0.21	0.88	0.33	11.80 ± 11.96	0.18	0.34	**0.23**	2.96 ± 5.28
x	x	–	0.11	0.90	0.19	23.27 ± 23.18	0.10	0.36	0.16	5.31 ± 9.71
x	–	x	0.21	0.88	0.33	11.84 ± 11.97	0.18	0.34	**0.23**	2.99 ± 5.34
–	x	x	0.10	**0.91**	0.18	25.03 ± 24.12	0.09	**0.41**	0.15	7.12 ± 13.49
x	x	x	0.10	**0.91**	0.18	25.08 ± 24.15	0.09	**0.41**	0.15	7.15 ± 13.54

the type and the Levenshtein distance, the maximum Levenshtein distance d to find spelling variant candidates for a given type T with length l_T can be computed using the following formula:

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

In order to generate candidates for short types as well, we use a minimum Levenshtein distance of 1.

This results in a small improvement in recall with a smaller loss in precision than using a Levenshtein distance > 1 for all types (cf. Table 7). 0.2 leads to the best F1 value in the OOV setting and one of the best F1 values in the text-eval setting. We use this threshold for further experiments.

Table 8 shows the comparison between Levenshtein distances 1, 2, and the Levenshtein distance with the modifications presented above (Mod), i.e., the combination of allowing transpositions and repetitions and using a different distance based on the length of type with a threshold of 0.2. For the following experiments, this combination is used to generate candidates (Lev(mod)).

The modifications of the Levenshtein distance presented so far are independent of the data set and require no training. A commonly used addition of the basic Levenshtein distance is to assign different costs to different substitutions or other edit operations (Bollmann 2012 and Pettersson et al. 2013a, for normalization of historical texts, Gomes and Pereira Lopes 2011, for cognate identification). These weights are trained on the specific data. In the next section, we will present a similar approach that combines rule-based simplification with the Levenshtein distance. For this, a set of rules is applied to all types of the lexicon to map them to a simplified lexicon. Then, all types whose simplified version is below the given Levenshtein threshold are seen as spelling variant candidates. Since differences between types that are covered by rules are removed before the Levenshtein distance is calculated, these differences do not contribute to the distance. Therefore, this approach is similar to assigning a cost of 0 to the edit operations encoded by the rules.

Table 7 Precision, recall, F1, and average number of candidates for different thresholds for the normalized Levenshtein distance (development set)

t	text-eval				OOV-eval			
	P	R	F1	C	P	R	F1	C
0.1	**0.22**	0.85	**0.35**	10.61 ± 10.84	**0.20**	0.27	0.23	2.10 ± 3.22
0.15	**0.22**	0.85	**0.35**	10.62 ± 10.83	**0.20**	0.27	0.23	2.13 ± 3.22
0.2	**0.22**	0.85	**0.35**	10.70 ± 10.78	**0.20**	0.31	**0.24**	2.39 ± 3.23
0.25	**0.22**	0.86	**0.35**	10.93 ± 10.67	0.16	0.39	0.22	3.78 ± 4.64$
0.3	0.21	0.86	0.34	11.39 ± 10.49	0.12	0.47	0.19	6.27 ± 7.13
0.35	0.18	0.87	0.30	13.52 ± 11.36	0.06	0.56	0.11	13.30 ± 17.93
0.4	0.13	**0.89**	0.22	19.49 ± 17.91	0.03	**0.68**	0.06	31.15 ± 36.82$

Table 8 Precision, recall, F1, and average number of candidates for the standard and the modified Levenshtein distance (development set)

Dist.	text-eval				OOV-eval			
	P	R	F1	C	P	R	F1	C
1	**0.22**	0.85	**0.35**	10.61 ± 10.84	**0.20**	0.27	0.23	2.10 ± 3.22
Mod	0.21	0.88	0.33	11.94 ± 11.91	0.17	0.38	**0.24**	3.33 ± 5.34
2	0.04	**0.95**	0.08	65.43 ± 56.47	0.04	**0.64**	0.07	25.86 ± 47.71

8.2 Simplification rules

In this section we present rules that remove variation from the texts. These are applied before the Levenshtein distance presented in the previous section is used to generate candidates. Therefore spelling variants with a higher Levenshtein distance than the specified threshold can be obtained. In the remainder of the section, we present three different methods of rule-based simplifications (Kol, Nie, and Sup) and evaluate them on GML data.

The first set of simpflication rules (Kol) consists of 26 rewrite rules that are applied to the types. These rules have been developed by linguists using three Middle Low German texts (Koleva et al. 2017). They use regular expressions with lookahead and lookbehind, such as in (4), which substitutes *g* with *gh* unless the *g* appears after an *n* or a *g* or before any of *g*, *h* or *t*.

(4) (?<![ng])g(?![ght]) → gh

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.

We compare this ruleset with two other rulesets (Nie and Sup) that are deliberately simpler and therefore easier to create. The rulesets are simply defined by a set of undirected correspondences between characters or one character and a character bigram as in (5a). This allows for the following edit operations: (1) deletion/insertion with left context (e.g. *gh* → *g*), (2) substitution (e.g. *i* → *j*), and (3) merges/splits (e.g. *ij* → *y*). This is similar to the rule pattern used by Pilz et al. (2006) with the difference that the rules are not weighted. Allowing for merges in the rules can capture some of the recall that was obtained by adding merges and splits to the Levenshtein distance in the previous section. Except for deletions, these operations are not context-sensitive. In our pipeline, the context of differences between two types will be considered in the SVM filter that is applied afterwards.

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 by length and then alphabetically such that the correspondences [‘g’, ‘gh’] and [‘gh’, ‘g’] both lead to the same rule, namely *gh* → *g*. 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. 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 (5a) is given in (5b).

(5) 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 (5b) 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 rules (Nie) with sets extracted from the training set (Sup). Nie is created on the basis of a list of graphemes and allographes in Middle Low German taken from Niebaum (2000).

For Sup, we take all the spelling variant pairs from the training set that have a Levenshtein distance (with the modification of including merges and splits) of 1 and extract the corresponding rules from them. In order to avoid extracting rules from errors (potential misspellings, transcription, and annotation errors, cf. Barteld 2017), we prune the ruleset by applying a frequency threshold to the rules, i.e., a minimal number of pairs that a rule appears with. We also use a precision threshold for the rules (Ernst-Gerlach and Fuhr 2006). The precision is calculated on the training set.

Table 9 shows the results for the different rulesets combined with the modified Levenshtein distance from the previous section. Nie, the ruleset based on the list of GML grapheme variants given in Niebaum (2000), shows the highest recall in both settings (0.92/0.77) indicating that the list is quite comprehensive. However, the aim of Niebaum (2000) is a presentation of possible grapheme variants and therefore does not take the precision of the variation into account. Consequently, the precision of the ruleset is quite low which makes the ruleset inappropriate for our purpose. Kol shows a better precision. This was expected as the ruleset from Koleva et al. (2017) applies the rules in a context-dependent fashion and was created with high precision in mind. The F1 value is therefore a lot better. The rulesets that are learned from the spelling variant pairs have 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).

Still, the precision for all rulesets is quite low as it can not be better than the precision of using only Lev(mod). This will be 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 Lev (mod). This combination leads to one of the rulesets where the precision is still at

Table 9 Precision, recall, F1, and average number of candidates for different rulesets (development set)

Freq.	Prec.	text-eval				OOV-eval			
		P	R	F1	C	P	R	F1	C
Lev(1)		**0.22**	0.85	**0.35**	10.61 ± 10.84	**0.20**	0.27	0.23	2.10 ± 3.22
Lev(mod)		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		0.06	**0.92**	0.11	42.45 ± 34.51	0.03	**0.77**	0.07	33.73 ± 55.79
10	0.7	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	0.8	0.20	0.88	0.32	12.59 ± 12.49	0.17	0.39	0.23	3.55 ± 5.82
10	0.85	0.20	0.88	0.33	12.12 ± 12.08	0.17	0.38	**0.24**	3.42 ± 5.59
10	0.9	0.21	0.88	0.33	11.94 ± 11.91	0.17	0.38	**0.24**	3.33 ± 5.34
20	0.7	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	0.8	0.20	0.88	0.33	12.28 ± 12.16	0.17	0.39	**0.24**	3.45 ± 5.51
20	0.85	0.20	0.88	0.33	12.05 ± 11.95	0.17	0.38	**0.24**	3.39 ± 5.45
20	0.9	0.21	0.88	0.33	11.94 ± 11.91	0.17	0.38	**0.24**	3.33 ± 5.34
30	0.7	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	0.8	0.20	0.88	0.33	12.27 ± 12.16	0.17	0.38	**0.24**	3.42 ± 5.46
30	0.85	0.20	0.88	0.33	12.04 ± 11.96	0.17	0.38	**0.24**	3.38 ± 5.43
30	0.9	0.21	0.88	0.33	11.94 ± 11.91	0.17	0.38	**0.24**	3.33 ± 5.34
40	0.7	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	0.8	0.20	0.88	0.33	12.27 ± 12.16	0.17	0.38	**0.24**	3.42 ± 5.46
40	0.85	0.20	0.88	0.33	12.04 ± 11.96	0.17	0.38	**0.24**	3.38 ± 5.43
40	0.9	0.21	0.88	0.33	11.94 ± 11.91	0.17	0.38	**0.24**	3.33 ± 5.34

0.24 while recall is improved from 0.38 to 0.61 in the OOV-eval setting. In the text-eval setting, this ruleset leads to one of the results with the second best recall (0.91) and the second best F1 value. The ruleset is given in (6).

(6) [[‘ij’, ‘i’], [‘s’, ‘z’], [‘i’, ‘j’], [‘ff’, ‘f’], [‘j’, ‘y’], [‘e̊’, ‘a’], [‘f’, ‘v’], [‘gh’, ‘g’], [‘u’, ‘v’], [‘ij’, ‘y’], [‘sz’, ‘s’], [‘th’, ’t’]]

By combining this ruleset with the Lev(mod) (S+Lev(mod)), we were 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 0.04/0.00, while using a distance of 2 leads to a recall of 0.95/0.64 with a loss in precision of 0.12 and 0.01, respectively. However, the precision is still very low (0.17/0.15).

9 Candidate filtering

In this section, we present approaches to improve the precision of the generation methods shown in the previous section by filtering candidate pairs using supervised machine learning. The first filter works on the type level and uses a SVM with radial-basis-function kernel to classify pairs of types as spelling variants or non–spelling variants. The second filter works on the token level and uses a convolutional neural network (CNN) to classify pairs of types with a left and a right context as spelling variants or non–spelling variants.

9.1 Type-level filter

The candidates generated in the previous steps are filtered using a SVM. The features are the same as described in our previous work (Barteld 2017). They comprise character n-grams from the aligned pairs and the cosine similarity between their two word embeddings. The embeddings are the same as we used in our previous work as well and are obtained using singular value decomposition with positive pointwise mutual information (SVD). In preliminary experiments, these embeddings outperformed alternative approaches using neural networks, e.g. using the skip-gram with negative-sampling method (SGNS) (Mikolov et al. 2013). This is in line with the results of Hamilton et al. (2016) who conclude that "SVD is more sensitive, as it performs well on detection tasks even when using a small dataset". SVD provided better results than SGNS, even when using subword information (Bojanowski et al. 2017).

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 data set has two characteristics that are relevant for training the classifier: First, 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). Second, the dataset is highly imbalanced.

In our previous work, these two issues of the training set were addressed by including only pairs of types where each type appeared at least 10 times in the data to get a set of reliable negative instances and used random undersampling to tackle the imbalance. We compare this approach with the use of 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. This is implemented using the Python libraries Scikit-learn (Pedregosa et al. 2011) and Imbalanced-learn (Lemaitre et al. 2017).

Table 10 shows the results of generating candidates with Levenshtein distance thresholds 1 and 2 or when generating the candidates with S+Lev(mod) as described in the previous sections along with the different SVM filters. For comparison, we repeat the results of using only lookup generation and no filtering. We also included the results for the perfect type-based spelling variant detection.

When comparing the simple SVM with the BSVM, the results are mixed: In the text-eval setting, the simple BSVM performs comparably to the simple SVM. But in

the OOV-eval setting it outperforms the simple SVM in terms of F1 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 S+Lev(mod) show comparable results in terms of F1 value. However, this F1 value is close to the upper bound for type-based spelling variant detection. This underlines the need for token-based spelling variant detection. In the OOV-eval setting using S+Lev(mod) leads to better results.

For the token-level filter, we use the combination of S+Lev(mod) with a BSVM filter. This combination performs comparably to using a Levenshtein distance of 1 and a simple SVM but with a better recall in the text-eval setting. In the OOV-eval setting, this combination leads to the best results overall.

9.2 Token-level filter

The approaches presented above work on the type level, i.e., candidates are generated and filtered regardless of the context. 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. We use a neural network that is similar to the CNN architecture proposed by Kim (2014) for sentence classification on the spelling variant candidate with n tokens before and after. The network is depicted in Fig. 3a. The words are on the one hand 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 et al. (2017), but instead of using LSTMs to obtain the surface (what they call syntactic) embedding, we create it using convolutional filters of length 2

Table 10 Precision, recall, F1, and average number of candidates for the SVM filter (development set)

Dist.	Filter	text-eval				OOV-eval			
		P	R	F1	C	P	R	F1	C
lookup		**0.36**	0.74	0.49	5.68 ± 7.90	1.00	0.00	0.00	0.00 ± 0.00
1	–	0.22	0.85	0.35	10.61 ± 10.84	0.20	0.27	0.23	2.10 ± 3.22
1	SVM	**0.36**	0.80	**0.50**	6.17 ± 7.91	0.48	0.20	0.29	0.64 ± 0.91
1	BSVM	0.35	0.81	0.49	6.49 ± 8.16	**0.51**	0.22	0.31	0.67 ± 0.94
Mod	–	0.17	0.91	0.29	14.93 ± 13.90	0.15	0.61	0.24	6.34 ± 10.65
Mod	SVM	0.33	0.87	0.47	7.47 ± 8.33	0.22	0.57	0.32	3.97 ± 6.66
Mod	BSVM	0.34	0.88	0.49	7.22 ± 8.46	0.40	0.50	**0.45**	1.91 ± 2.93
2	–	0.04	**0.95**	0.08	65.43 ± 56.47	0.04	**0.64**	0.07	25.86 ± 47.71
2	SVM	0.17	0.92	0.29	14.99 ± 14.38	0.10	0.63	0.18	9.37 ± 15.29
2	BSVM	0.17	0.91	0.29	14.51 ± 15.78	0.21	0.60	0.31	4.46 ± 6.97
gold		0.34	1.00	0.51	8.08 ± 10.32	0.78	1.00	0.87	1.97 ± 3.96

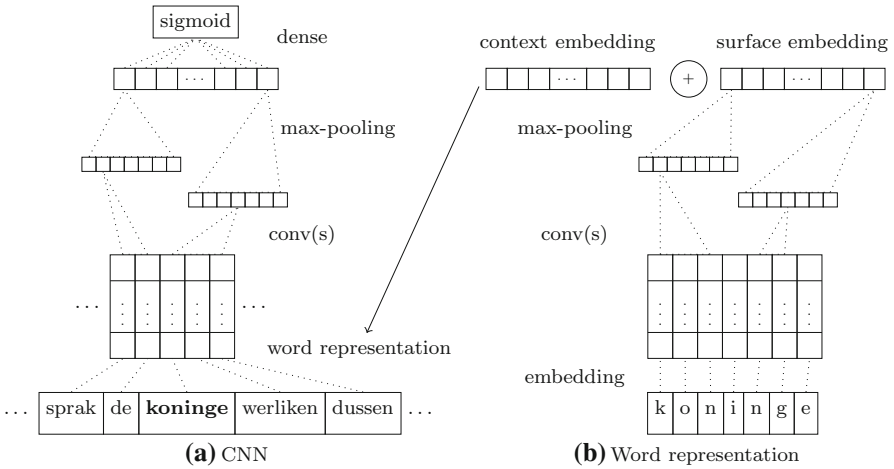

Fig. 3 Network architecture

and 3 that are applied to the (padded) sequence of characters for each word. This part of the network is depicted in Fig. 3b. The surface representation is not pre-trained.

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 combined with the target word. We apply 50 filters of each length on the character level and on the word level.

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 and contains actual spelling variants in the set of negative pairs. 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 (Kingma and Ba 2014). Rectified Linear Units are used for non-linearity. The architecture was implemented using Keras (Chollet et al. 2015).

Table 11 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 detect 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 F1 value as the loss in recall is not matched by the gain in precision. In the next section, we will look at possible reasons for this.

Table 11 Precision, recall, F1, and average number of candidates for the CNN filter (development set)

Embedd	Size	text-eval				OOV-eval			
		P	R	F1	C	P	R	F1	C
S+Mod-BSVM		0.34	**0.88**	0.49	7.22 ± 8.46	0.40	**0.50**	**0.45**	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

10 Results and error analysis

Table 12 presents the results on the test set. These show the same pattern as on the development set: With type-level generation and filtering, 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 F1 value. In the OOV-eval setting, where a lookup is not available, both recall and precision are improved, compared to candidate generation using the Levenshtein distance. In both settings the new generation approach with a filter on the type level (S+Lev(mod)-BSVM) leads to an improvement over our previous approach (Lev(1)-SVM).

Token-level filtering on the other hand is only helpful (in terms of F1 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 13, which

Table 12 Precision, recall, F1, and average number of candidates for different pipelines (test set)

Dist.	Filter	text-eval				OOV-eval			
		P	R	F1	C	P	R	F1	C
Lookup		0.29	0.69	0.41	5.62 ± 7.72	1.00	0.00	0.00	0.00 ± 0.00
1	–	0.16	0.85	0.27	12.17 ± 11.41	0.11	0.24	0.16	3.91 ± 5.55
Mod	–	0.14	**0.87**	0.25	14.26 ± 12.53	0.09	**0.40**	0.14	8.56 ± 12.68
1	SVM	0.25	0.83	0.39	7.72 ± 8.53	0.22	0.24	0.23	2.06 ± 2.58
Mod	BSVM	0.28	0.83	0.42	6.91 ± 7.96	0.27	0.34	**0.30**	2.33 ± 2.86
+ CNN (both, 1)		**0.47**	0.57	**0.52**	2.80 ± 3.36	**0.40**	0.18	0.25	0.84 ± 1.37
+ CNN (surf., 1)		0.45	0.51	0.48	2.65 ± 3.19	0.34	0.16	0.22	0.88 ± 1.44

shows the results calculated without the types that appear more than 10 times in the whole data set. Here, the token-based filter leads to a high drop in recall in both settings, resulting in a lower F1. 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 using 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: While we were able to improve the recall in the OOV-eval setting from 0.24 with Levenshtein distance 1 to 0.40 using the modified Levenshtein distance, this is still a very low recall.

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 the modified Levenshtein distance misses spelling variants for 588 types. We sampled 100 types from these and looked for patterns regarding the variants that were not generated.

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 modified Levenshtein distance. 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. ['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.

Many of the missing spelling variants involve prefixation and suffixation. One example of a prefix that appears often in spelling variant pairs is *g(h)e*: *richten*, *gerichten* '(to) sentence'; *coft*, *ghecoft* '(to) buy'; *nemet*, *genomen* '(to) take'. While in Standard Modern 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

Table 13 Precision, recall, F1, and average number of candidates for different pipelines on types with frequency lower 10 (test set)

Dist.	Filter	text-eval				OOV-eval			
		P	R	F1	C	P	R	F1	C
Lookup		**0.54**	0.03	0.05	0.03 ± 0.19	1.00	0.00	0.00	0.00 ± 0.00
1	–	0.21	0.51	0.30	1.29 ± 1.40	0.14	0.25	0.18	1.55 ± 2.35
Mod	–	0.17	**0.68**	0.27	2.08 ± 2.33	0.10	**0.51**	0.17	4.47 ± 6.59
1	SVM	0.29	0.51	0.37	0.93 ± 1.06	0.21	0.25	0.23	1.04 ± 1.51
Mod	BSVM	0.45	0.61	**0.51**	0.72 ± 0.92	0.33	0.45	**0.38**	1.24 ± 1.81
+ CNN (both, 1)		0.51	0.34	0.41	0.35 ± 0.68	**0.42**	0.26	0.32	0.55 ± 1.13
+ CNN (surf., 3)		0.51	0.36	0.42	0.37 ± 0.71	0.41	0.27	0.32	0.59 ± 1.17

variation. An example suffix from the data is *en*—or better the lack of the suffix *en*, as 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'.

11 Conclusion and further work

In this paper, we presented a pipeline for spelling variant detection in non-standard texts which is a challenge for many DH applications. Spelling variant detection is an alternative to normalization for improving the performance of NLP tools on non-standard data and facilitating searching in the texts. In contrast to normalization, this approach works without reference to a target language. Therefore, it is usable even when a target language does not exist or is not available. The pipeline presented in this paper is trainable on examples for spelling variants from the non-standard data.

Given a token, i.e., a type with a specific context, the pipeline outputs spelling variants for this type. This works by first generating spelling variant candidates. These candidates are then filtered on the type level and subsequently filtered on the token level. The pipeline was evaluated using precision, recall, and F1 on GML texts in two settings that simulate the two use-cases of (i) training and applying NLP tools on the non-standard data (OOV-eval) and (ii) searching in the data (text-eval).

By adding a rule-based simplfication step, using a modified Levenshtein distance and a Bagging-SVM instead of a standard SVM, we were able to improve the type-based spelling variant generation from our previous work. For the text-eval setting, which estimates the performance of the candidate generation for searching in GML texts, we were able to improve the F1 score from 0.39 to 0.42. This improvement is largely due to an improvement for infrequent types: For types that occur less than 10 times, the original pipeline achieved an F1 score of 0.37, while the presented pipeline achieved 0.51. Consequently the improvement in the OOV-eval setting that focuses on unknown and therefore often infrequent types is more pronounced: Here we were able to improve the F1 score from 0.23 to 0.30.

The main contribution of this work is the token-level filter. By applying this filter, the F1 value increases to 0.52 in the text-eval setting. In the OOV-eval setting, however, the token-level filter did not perform as well, leading to a drop in F1 value when applying the filter. The reason for this is that the filter does not perform well on infrequent types. Therefore, we plan to concentrate on such types in future work. One approach that we will look into is applying the rule-based simplification to the corpus before learning the context embeddings. This groups probable spelling variants together and could thus lead to better representations.

However, as the pipeline uses separate filters, digital humanities researchers using the spelling variant detection can choose between a higher recall or a higher precision by simply leaving out, e.g. the token-level filter. To assist the users with that, we plan to integrate the pipeline into an easy to use search application for DH researchers, which would facilitate searching collections of unannotated non-standard texts by suggesting and searching for spelling variants of the query. The separation of the different parts of the pipeline will allow researchers to have fine-grained control over the variants produced. Researchers could, for instance, remove

types that are generated by the Levenshtein generator or re-include types that are filtered either by the type- or the token-level filter.

Acknowledgements The first author was supported by the German Research Foundation (DFG), grant SCHR 999/5-2. We would like to thank the anonymous reviewers for their helpful remarks and Adam Roussel for improving our English. All remaining errors are ours.

References

Adesam, Y., & Bouma, G. (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*, (pp. 32–42). Berlin, Germany: Association for Computational Linguistics.

Barteld, F. (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*, (pp. 11–22). Valencia, Spain: Association for Computational Linguistics.

Barteld, F., Schröder, I., & Zinsmeister, H. (2015). Unsupervised regularization of historical texts for POS tagging. In *Proceedings of the workshop on corpus-based research in the humanities* (CRH), (pp. 3–12). Warsaw, Poland.

Barteld, F., Schröder, I., & Zinsmeister, H. (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*, (pp. 52–62). Berlin, Germany: Association for Computational Linguistics.

Bojanowski, P., Grave, E., Joulin, A., & Mikolov, T. (2017). Enriching word vectors with subword information. *Transactions of the Association for Computational Linguistics*, *5*, 135–146.

Bollmann, M. (2012). (Semi-)automatic normalization of historical texts using distance measures and the Norma tool. In Mambrini F, Passarotti M, Sporleder C (eds) *Proceedings of the second workshop on annotation of corpora for research in the humanities (ACRH-2)*, (pp. 3–14). Lisbon, Portugal.

Bollmann, M. (2013). POS tagging for historical texts with sparse training data. In *Proceedings of the 7th linguistic annotation workshop and interoperability with discourse*, (pp. 11–18). Sofia, Bulgaria: Association for Computational Linguistics.

Bollmann, M., & Søgaard, A. (2016). Improving historical spelling normalization with bi-directional LSTMs and multi-task learning. In *Proceedings of COLING 2016, the 26th international conference on computational linguistics: technical papers*, (pp. 131–139). Osaka, Japan.

Bollmann, M., Petran, F., & Dipper, S. (2011). Applying rule-based normalization to different types of historical texts—An evaluation. In *Proceedings of the 5th language & technology conference: humanlanguage technologies as a challenge for computer science and linguistics (LTC 2011)*, (pp. 339–344). Poznan, Poland.

Bollmann, M., Bingel, J., & Søgaard, A. (2017). Learning attention for historical text normalization by learning to pronounce. In *Proceedings of the 55th annual meeting of the association for computational linguistics (volume 1: long papers)*, (pp. 332–344). Vancouver, Canada : Association for Computational Linguistics

Chakrabarty, A., Pandit, O.A., & Garain, U. (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)*, (pp. 1481–1491). Vancouver, Canada: Association for Computational Linguistics.

Chollet F, et al. (2015). Keras. https://github.com/fchollet/keras

Chrupała, G. (2014). Normalizing tweets with edit scripts and recurrent neural embeddings. In *Proceedings of the 52nd annual meeting of the association for computational linguistics (volume 2: short papers)*, (pp. 680–686). Baltimore, Maryland, USA: Association for Computational Linguistics.

Ciobanu, M.A., & Dinu, L.P. (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)*, (pp. 99–105). Baltimore, Maryland, USA: Association for Computational Linguistics.

Costa Bertaglia, T.F., & Volpe Nunes, MdG. (2016). Exploring word embeddings for unsupervised textual user-generated content normalization. In *Proceedings of the 2nd workshop on noisy user-generated text (WNUT)*, (pp. 112–120). Osaka, Japan: The COLING 2016 Organizing Committee.

Damerau, F. J. (1964). A technique for computer detection and correction of spelling errors. *Communications of the ACM*, *7*(3), 171–176.

Derczynski, L., Ritter, A., Clark, S., & Bontcheva, K. (2013). Twitter part-of-speech tagging for all: Overcoming sparse and noisy data. In *Proceedings of the international conference recent advances in natural language processing (RANLP 2013)*, (pp. 198–206). Hissar, Bulgaria.

Dipper, S., Lüdeling, A., & Reznicek, M. (2013). NoSta-D: A corpus of German Non-Standard Varieties. In M. Zampieri, S. Diwersy (eds) *Non-standard data sources in corpus-based research*, ZSM-Studien, vol 5, Shaker, (pp. 69–76).

Ernst-Gerlach, A., & Fuhr, N. (2006). Generating search term variants for text collections with historic spellings. In M. Lalmas, A. MacFarlane, S. Rüger, A. Tombros, T. Tsikrika & A. Yavlinsky (Eds.), *Advances in Information Retrieval. ECIR 2006. Lecture Notes in Computer Science* (Vol 3936, pp. 49–60). Berlin: Springer.

Galar, M., Fernandez, A., Barrenechea, E., Bustince, H., & Herrera, F. (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.

Gomes, L., & Pereira Lopes, J.G. (2011). Measuring spelling similarity for cognate identification. In L. Antunes, H. S. Pinto (Eds.), *Progress in artificial intelligence: 15th Portuguese conference on artificial intelligence. EPIA 2011. Lecture Notes in Computer Science* (Vol 7026, pp. 624–633). Berlin: Springer.

Hamilton, W.L., Leskovec, J., & Jurafsky, D. (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)*, (pp. 1489–1501). Berlin, Germany: Association for Computational Linguistics.

Han, B., Cook, P., & Baldwin, T. (2013). Lexical normalization for social media text. *ACM Transactions on Intelligent Systems and Technology*, *4*(1), 5:1–5:27.

Hauser, A.W., & Schulz, K.U. (2007). Unsupervised learning of edit distance weights for retrieving historical spelling variations. In *Proceedings of the first workshop on finite-state techniques and approximate search*, (pp. 1–6). Borovets, Bulgaria.

Hogenboom, A., Bal, D., Frasincar, F., Bal, M., De Jong, F., & Kaymak, U. (2015). Exploiting emoticons in polarity classification of text. *Journal of Web Engineering*, *14*(1–2), 22–40.

Jurish, B. (2010a). Comparing canonicalizations of historical German text. In *Proceedings of the 11th Meeting of the ACL Special Interest Group on Computational Morphology and Phonology*, (pp. 72–77). Uppsala, Sweden: Association for Computational Linguistics.

Jurish, B. (2010b). More than words: Using token context to improve canonicalization of historical German. *Journal for Language Technology and Computational Linguistics (JLCL)*, *25*(1), 23–39.

Jurish, B. (2011). *Finite-state canonicalization techniques for historical German*. Ph.D. thesis, University of Potsdam, Germany.

Jurish, B., Thomas, C., & Wiegand, F. (2014). Querying the Deutsches Textarchiv. In U. Kruschwitz , F. Hopfgartner & C. Gurrin (eds) *MindTheGap2014* (pp. 25–30). Berlin, Germany.

Kestemont, M., Daelemans, W., & Pauw, G. D. (2010). Weigh your words–memory-based lemmatization for Middle Dutch. *Literary and Linguistic Computing*, *25*(3), 287–301.

Kim, Y. (2014). Convolutional neural networks for sentence classification. In *Proceedings of the 2014 conference on empirical methods in natural language processing (EMNLP)*, (pp. 1746–1751). Doha, Qatar: Association for Computational Linguistics.

Kingma, D.P., & Ba, J. (2014). Adam: A method for stochastic optimization. CoRR abs/1412.6980.

Kobus, C., Yvon, F., & Damnati, G. (2008). Normalizing SMS: Are two metaphors better than one? In *Proceedings of the 22nd international conference on computational linguistics (Coling 2008)*, (pp. 441–448). Manchester, United Kingdom.

Koleva, M., Farasyn, M., Desmet, B., Breitbarth, A., & Hoste, V. (2017). An automatic part-of-speech tagger for Middle Low German. *International Journal of Corpus Linguistics*, *22*(1), 107–140.

Lemaitre, G., Nogueira, F., & Aridas, C. K. (2017). Imbalanced-learn: A Python toolbox to tackle the curse of imbalanced datasets in machine learning. *Journal of Machine Learning Research*, *18*(17), 1–5.

Levenshtein, V. (1966). Binary codes capable of correcting deletions, insertions, and reversals. *Soviet Physics Doklady*, *10*(8), 707–710.

Li X. L., Liu B. (2005). Learning from positive and unlabeled examples with different data distributions. In J. Gama, R. Camacho, P. B. Brazdil, A. M. Jorge & L. Torgo (Eds.), *Machine Learning: ECML 2005. Lecture Notes in Computer Science* (Vol 3720, pp. 218–229). Berlin: Springer.

Ljubešić, N., Zupan, K., Fišer, D., & Erjavec, T. (2016). Normalising Slovene data: historical texts vs. user-generated content. In *Proceedings of the 13th Conference on Natural Language Processing (KONVENS 2016)*, (pp. 146–155). Bochum, Germany.

Logačev, P., Goldschmidt, K., & Demske, U. (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)*, (pp. 103–112). Tübingen, Germany.

Mihov, S., & Schulz, K. U. (2004). Fast approximate search in large dictionaries. *Computational Linguistics*, *30*(4), 451–477.

Mikolov, T., Chen, K., Corrado, G., & Dean, J. (2013). Efficient estimation of word representations in vector space. CoRR abs/1301.3781.

Mordelet, F., & Vert, J.P. (2014). A bagging SVM to learn from positive and unlabeled examples. *Pattern Recognition Letters 37(Supplement C)*:201–209.

Niebaum, H. (2000). Phonetik und Phonologie, Graphetik und Graphemik des Mittelniederdeutschen. In *Sprachgeschichte: Ein Handbuch zur Geschichte der deutschen Sprache und ihrer Erforschung, 2nd edn* (pp. 1422–1430). Berlin, Boston: DeGruyter.

Pedregosa, F., Varoquaux, G., Gramfort, A., Michel, V., Thirion, B., Grisel, O., Blondel, M., Prettenhofer, P., Weiss, R., Dubourg, V., Vanderplas, J., Passos, A., Cournapeau, D., Brucher, M., Perrot, M., & Duchesnay, É. (2011). Scikit-learn: Machine learning in Python. *Journal of Machine Learning Research* 12(Oct):2825–2830.

Pettersson, E., Megyesi, B., & Nivre, J. (2012). Rule-based normalisation of historical text — A diachronic study. In *Proceedings of KONVENS 2012 (LThist 2012 workshop)*, (pp. 333–341). Vienna, Austria.

Pettersson, E., Megyesi, B., & Nivre, J. (2013a). 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)*, (pp. 163–179). Oslo, Norway.

Pettersson, E., Megyesi, B., & Tiedemann, J. (2013). An SMT approach to automatic annotation of historical text. *Proceedings of the Workshop on Computational Historical Linguistics at NODALIDA, 2013*, (pp. 54–69). Oslo, Norway.

Pilz, T., Luther, W., Fuhr, N., & Ammon, U. (2006). Rule-based search in text databases with nonstandard orthography. *Literary and Linguistic Computing*, *21*(2), 179–186.

Piotrowski, M. (2012). *Natural language processing for historical texts. Synthesis lectures on human language technologies* 17, Morgan & Claypool Publishers.

Schulz, K., & Mihov, S. (2001). *Fast string correction with Levenshtein-automata*. CIS-Report 01-127. Tech. rep., CIS, University of Munich.

Schulz, K., & Mihov, S. (2002). Fast string correction with Levenshtein-automata. *International Journal on Document Analysis and Recognition*, *5*, 67–85.

Tjong Kim Sang, E., Bollmann, M., Boschker, R., Casacuberta, F., Dietz, F., Dipper, S., 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.

van der Goot, R., Plank, B., & Nissim, M. (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*, (pp. 31–39). Copenhagen, Denmark: Association for Computational Linguistics.

Weissweiler L. & Fraser A. (2018). Developing a Stemmer for German Based on a Comparative Analysis of Publicly Available Stemmers. In G. Rehm, T. Declerck (Eds.), *Language Technologies for the Challenges of the Digital Age. GSCL 2017. Lecture Notes in Computer Science* (Vol 10713, pp. 81–94). Cham: Springer.

Yang, Y., & Eisenstein, J. (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 (NAACL-HLT 2016)*, (pp. 1318–1328). San Diego, California, USA: Association for Computational Linguistics

Yujian, L., & Bo, L. (2007). A normalized Levenshtein distance metric. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, *29*(6), 1091–1095.