Dealing with word-internal modification and spelling variation in data-driven lemmatization

Fabian Barteld **Ingrid Schröder** **Heike Zinsmeister**

Institut für Germanistik
Universität Hamburg
firstname.lastname@uni-hamburg.de

Abstract

This paper describes our contribution to two challenges in data-driven lemmatization. We approach lemmatization in the framework of a two-stage process, where first lemma candidates are generated and afterwards a ranker chooses the most probable lemma from these candidates. The first challenge is that languages with rich morphology like Modern German can feature morphological changes of different kinds, in particular word-internal modification. This makes the generation of the correct lemma a harder task than just removing suffixes (stemming). The second challenge that we address is spelling variation as it appears in non-standard texts. We experiment with different generators that are specifically tailored to deal with these two challenges. We show in an oracle setting that there is a possible increase in lemmatization accuracy of 14% with our methods to generate lemma candidates on Middle Low German, a group of historical dialects of German (1200–1650 AD). Using a log-linear model to choose the correct lemma from the set, we obtain an actual increase of 5.56%.

1 Introduction

Lemmatization is the task of finding the lemma or base form for a given word token. It is used as a preprocessing step for information retrieval and other NLP applications for languages with rich morphology and has been shown to outperform stemming for some tasks (Korenius et al., 2004). Lemmatization can be formalized as a string transduction task where for an input sequence of tokens $t_1 \dots t_n$ an output sequence of lemmas $l_1 \dots l_n$ is produced. This task has been approached in a variety of ways, e.g. by combining morphological rules with dictionary lookups (Sennrich and Kunz, 2014). Chrupała (2006) introduced a sequence-labeling approach to lemmatization in which a token is labeled with a rule that transforms it to its lemma. The set of rules from which the labels are chosen are induced automatically from the training data. Müller and Schütze (2015) use a similar setting, but, instead of choosing a rule to apply, they apply all possible rules and afterwards use a ranker to select the lemma. Conceptually, this is a two-stage approach towards lemmatization – first generating lemma candidates for a given type, i.e. an inflected word form, and then choosing the best of these candidates for the token. We follow this approach and present generators that increase the number of correct lemma candidates that can be generated for out-of-vocabulary (OOV) words in the case of word-internal modification and spelling variation:

(i) Word-internal modifications like the *umlaut* (*Schl**ä**ge* - *Schl**a**g* "strikes - (the) strike") and infixation (*auf**geg**essen* - *aufessen* "eaten up - eat up") in Modern German (DEU)[1] pose special problems to lemma candidate generation. In order to improve the generalization capabilities of the rules induced from the training data, we substitute the edit trees (ET) (Chrupała, 2008) used by Müller and Schütze (2015) with lexical correspondences (LC) (Fulop and Neuvel, 2013).

(ii) Spelling variation as it appears in historical language or computer-mediated communication results in an increase in data sparsity and therefore a large number of OOV words. We add a generator that returns the lemma candidates for the most similar in-vocabulary (IV) word(s). Thereby, the lemmatization can be made more ro-

[1] Abbreviations for language names follow the ISO 639-3 codes.

Proceedings of the 10th SIGHUM Workshop on Language Technology for Cultural Heritage, Social Sciences, and Humanities (LaTeCH), pages 52–62,
Berlin, Germany, August 11, 2016.

bust against simple misspellings or spelling variations, since the correct lemma can be returned even in these cases.

We test our approach on Middle Low German (GML) texts. GML is a group of historical dialects of German (1200–1650 AD), which – like DEU – features word-internal modification. Also, as a historical language, GML exhibits spelling variation. In order to see how hard these features make the task of lemmatizing GML, we compare it with lemmatizing DEU newswire texts.

2 Related work

The methods presented here are general in nature and can be incorporated into different lemmatization approaches. We apply our methods with LEMMING (Müller and Schütze, 2015), a state-of-the-art lemmatizer which performs lemmatization with a log-linear model that combines candidate generation with a probabilistic ranker.[2] LEMMING can be used to do lemmatization independently from morphological tagging or to combine both tasks with a joint model. As we are interested in lemmatization, we only use the independent lemmatization model.[3] The generator used for lemma candidate generation can be of any kind. The original version of LEMMING uses a deterministic rule-based generator and learns the set of rules from the training data. Following Chrupała (2008), the standard generator in LEMMING uses edit trees (ET). In ETs – unlike in the shortest edit scripts on reversed strings that are used by Chrupała (2006) – the positions of edits are not all indexed from one end of the string but either from the beginning or the end. This is similar to prefix and suffix replacement rules (Gesmundo and Samardžić, 2012). Therefore prefixes and suffixes are handled independently from the length of the word. However, as Jongejan and Dalianis (2009) point out, languages like German and Dutch also allow word-internal modifications, which are not covered independently of the word length by rules which index the position of the change relative to either the beginning or the end of the word. This is illustrated with ETs in Figure (1a) and (1b). The numbers at the nodes of the ETs denote the position of the substring this node represents measured from the beginning and the end of the type. In the case of *Bäume* and *Baum* the longest common substring *um* starts after the second character in *Bäume* and ends before the last character. Therefore, the first node is indexed with 2 and 1. See Chrupała (2008) for a detailed description of edit trees.[4] As can be seen from Figure (1a) and (1b) the word-internal umlaut leads to different indices. Our contribution to this challenge is to test rules that model word-internal modifications independently of the word length. We use *lexical correspondences* that have been proposed in Whole Word Morphology by Fulop and Neuvel (2013) and have been used in morphological learning (Neuvel and Fulop, 2002).

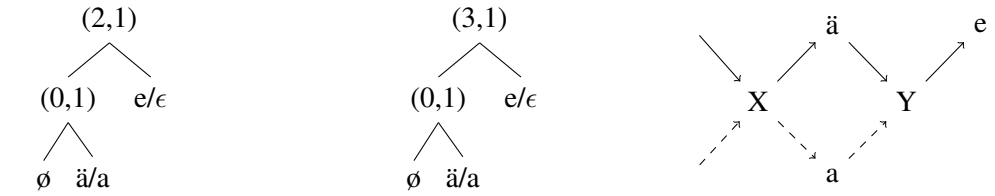
Our second addition to the generator addresses spelling variation. Spelling variation is often dealt with in a preprocessing step called normalization before taggers or lemmatizers are applied to the data (Eisenstein, 2013). Formally, such a normalization is a string transduction task like lemmatization. Therefore, LEMMING directly deals with spelling variation by learning rules that generate the lemma, simultaneously removing inflection and normalizing variation. The rules inferred from the training data, however, will only deal with specific combinations of inflection and spelling variation.

A way of dealing with spelling variation independently of inflection is to identify possible spelling variants and use them for the lemmatization. In several approaches spelling variation patterns are learned from the training data exploiting the annotation (Kestemont et al., 2010; van Halteren and Rem, 2013; Logačev et al., 2014). Apart from using these patterns to expand the training data by creating probable spelling variants, Kestemont et al. (2010) produce IV words that have a high probability to be a spelling variants for OOV words and use their lemmatization to predict the lemma of OOV words. We adopt a similar approach for lemma candidate generation: We determine probable spelling variants for all OOV types in the set of IV types and generate lemma candidates based on these. We experiment with different similarity measures for detecting the probable spelling variants.

[2] The original version of LEMMING is available at `http://cistern.cis.lmu.de/lemming/`. A version containing the generators described in this paper is available at `https://github.com/fab-bar/cistern`.

[3] This is referred to as LEMMING-P in Müller and Schütze (2015).

[4] Note that for a given pair of type and lemma more than one ETs might exist, see Appendix A.1 for details. The appendices can be found at `https://github.com/fab-bar/paper-LaTeCH2016`.



(a) Bäume → Baum "trees - (the) tree"

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

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

Figure 1: Edit trees and lexical correspondence for the a-umlaut + -e inflection pattern

3 Dealing with word internal modification

As mentioned above, edit trees (ET) do not generalize over word-internal modifications. An ET learned from the pair *Bäume*, *Baum* "trees – (the) tree" cannot predict the lemma *Traum* "(the) dream" for *Träume* "dreams". In order to allow for such generalizations, we use lexical correspondences for lemma candidate generation instead.

We define a *lexical correspondence* (LC) for two words over some alphabet $w_1, w_2 \in \Sigma^*$ as the tuple $\langle \mathcal{T}, \mathcal{L} \rangle$[5] where $\mathcal{T}$ and $\mathcal{L}$ are two sequences of constants and variables with the requirement that the same variables appear in both sequences.[6] Constants are elements of $\Sigma^* \setminus \{\epsilon\}$, i.e. the possible words over the alphabet Σ with the exception of the empty word ϵ. Variables are placeholders that can be replaced with constants. Note that we do not allow empty strings as constants. Therefore, a variable must be replaced with at least one letter. Figure (1c) depicts the LC for the ETs in Figures (1a) and (1b). This shows that LCs are able to generalize over pairs of type and lemma that ETs do not generalize over.

The solid arrows in Figure (1c) represent the sequence $\mathcal{T}$ (representing e.g. *Bäume*) and the dashed arrows the sequence $\mathcal{L}$ (representing e.g. *Baum*). The variables X and Y are depicted in the middle. By replacing the latter with the missing parts of the words, e.g. $\{X/\mathrm{B}, Y/\mathrm{um}\}$, type and lemma can be read off the sequences.

In order to create a lemma given a type and a LC, the constants in sequence $\mathcal{T}$ are matched with characters in the type and the variables are replaced with the remaining substrings. Then the lemma can be read off the second sequence $\mathcal{L}$. For instance, given the type *Träume* and the LC in Figure (1c) the constants match with *ä* and the final *e* in *Träume*, creating the replacements $\{X/\mathrm{Tr}, Y/\mathrm{um}\}$. Using these replacements in the second sequence creates the lemma *Traum*.

An ET can be unambiguously transformed into a LC. Hence, type-lemma pairs from which the same ET is induced lead to the same LC. This proves that LCs generalize over all cases over which ETs generalize.[7] On the other hand, as Fulop and Neuvel (2013) pointed out, lexical correspondences with more than one variable cannot always be unambiguously applied. One example is the type *Säbelschläge* "(the) blows with a saber" and the LC from Figure (1c). The *ä* can be matched with two positions in the type leading to two different replacements and corresponding lemma candidates: $\{X/\mathrm{S}, Y/\mathrm{belschläg}\}$ (lemma candidate: **Sabelschläg*) and $\{X/\mathrm{Säbelschl}, Y/\mathrm{g}\}$ (lemma candidate: *Säbelschlag*).

In the context of LEMMING, we can generate both variants and let the ranker decide. However, this leads to a trade-off between better generalization of the rules and possible indeterminacy, i.e. bigger candidate sets.[8] The latter increases the computational costs for training and applying the ranker. It also introduces new possible errors for the ranker. Consequently, we are looking for a way to restrict the overgeneralization. One source of overgeneralization are LCs of the form $\langle [X, Y], [X, \text{'constant'}, Y] \rangle$. An LC like this is applicable to all types. It creates a lemma candidate by inserting the constant and does this between all

[5] Note that the order of the sequences is fixed as we always use lexical correspondences between type and lemma in that order.

[6] Our definition of lexical correspondence is less strict than the original definition given by Fulop and Neuvel (2013). Their definition also takes syntactic categories and semantic relations into account.

[7] See Appendix A.2 for details.

[8] Note that the ET in Figure (1a) learned from *Bäume*, *Baum* will be applicable to *Säbelschläge* as well and creates the lemma candidate *Sabelschläg*. Therefore, in this case, it is unlikely that the LC generates more false lemma candidates than created with using ETs.

the letters. One example for a similar LC results from the German pair *sind*, *sein* "(they) are – (to) be" which leads to $\langle [X, Y, \text{'d'}], [X, \text{'e'}, Y] \rangle$. This LC is applicable to all types ending with *d* and allows the insertion of *e* at multiple positions.

In order to avoid this kind of overgeneration insertions are anchored by their character offset either from the beginning or the end of the type. This position can be read off directly from an ET.[9] Therefore, these lexical correspondences with anchored insertions (LC-AI) have a generalization capacity that is reduced in comparison to unanchored LCs but is still higher when compared to ETs.

4 Dealing with spelling variation

In this section, we present a method for allowing the lemmatizer to deal with spelling variation – namely generating lemma candidates from similar in-vocabulary (IV) types. Using this approach, the correct lemma candidate can be generated for out-of-vocabulary (OOV) types that are spelling variants of IV types. We restrict this additional generation of lemma candidates to OOV types to avoid overgeneration.

An upper bound for improving the coverage by this method is given by generating the lemma candidates for all of the training instances and add them to the lemma candidate set for OOV types. The problem with this approach is the large number of lemma candidates for OOV words. This makes it hard for the ranker to find the correct lemma. Therefore, our aim is to select an appropriate subset from the training data that contains possible spelling variants and only add the candidates generated on this subset to the lemma candidates. There are distance measures explicitly proposed for finding spelling variants (Kempken, 2005; Pilz, 2009; Bollmann, 2012) that could be used for this task. However, these measures need to be trained on pairs of spelling variant and standard spelling which are not available in our case. Consequently, we use string similarity measures that can be used without training data of this kind. Kestemont et al. (2010) use the Levenshtein distance (Levenshtein, 1966) and Dice's coefficient (Dice, 1945) to detect spelling variants in Middle Dutch texts. While they report a better performance of 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.[10] This weighted version is given by Equation 1.

$$JaccardIndex_w(f(t_1), f(t_2)) = \frac{\sum_{\text{f} \in f(t_1) \cap f(t_2)} w(\text{f})}{\sum_{\text{f} \in f(t_1) \cup f(t_2)} w(\text{f})} \quad (1)$$

Here, $f(t) \subseteq F$ is the set of similarity features for a type t and $w : F \rightarrow \mathbb{R}$ is a weight function. Both can be chosen differently allowing to fine-tune the measure for specific data. For normalizing Twitter data, Jin uses bigrams, skip-1-bigrams and sets the weight for each feature to 1.

Barteld et al. (2015) use yet another similarity measure, Proxinette (Hathout, 2014), for spelling variant detection. Similar to the Jaccard Index, Proxinette uses similarity features to compute the similarity of two types. Differing from the Jaccard Index, in Proxinette the similarity score is obtained by the probability of a random walk in a bipartite graph with types and similarity features as vertices. This leads to a weight for the features of $\frac{1}{deg(f)}$, where $deg(f)$ is the degree of the vertex f, i.e. the number of types having this similarity feature. Since in this form, the weights are dependent 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.[11] The relative frequency is estimated based on a training corpus (in our case the training corpus for the lemmatizer) leading to a weight of 0 for all features that appear in every type of the corpus, and 1 for features that did not appear in the corpus.

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, a problem that

[9] See Appendix A.3 for a detailed description.

[10] 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. Therefore, we restrict ourselves to the Jaccard Index.

[11] 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.

has been noted by Jin (2015) for the Jaccard Index with character *n*-grams of fixed lengths.

Even when no training data for spelling variation in the form of variant and standard form is available, variation patterns can be learned approximately using data annotated with POS tags and/ or lemmas (Kestemont et al., 2010; van Halteren and Rem, 2013; Logačev et al., 2014). We follow Kestemont et al. (2010), who use word forms annotated with the same lemma that have a Levenshtein distance of 1 as proxies for pairs of spelling variants. They train a memory-based learner (MBL) on the typical differences between those spelling variants and use it to rerank Levenshtein neighbours according to these variation patterns. As using a MBL is slow at tagging time, we estimate the probability of two types being spelling variants directly, following an approach similar to Logačev et al. (2014). Given an edit operation e, we estimate its probability of leading to a spelling variant, $P(e)$, by

$$\sum_{(t_i,t_j)\in tr(e)} min(1, |l(t_i) \cap l(t_j)|) * \frac{1}{|tr(e)|} \quad (2)$$

where $tr(e) = \{(t_i, t_j) | t_i \xrightarrow{e} t_j\}$, i.e. the set of all pairs of types (t_i, t_j) from the training data, such that t_i can be transformed into t_j by applying e and $l(t)$ is the set of all lemmas type t appears with in the training data. The $P(e)$ for an edit operation e that does not appear in the training data is set to 1 – thereby the probabilities capture negative evidence against the assumption that an edit operation leads to a spelling variant.

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 reduced by removing those for which the probability is below a given threshold.

We will apply these different similarity measures to extract types more similar to a given OOV type than a threshold from the IV types. These extracted types will then be used to generate lemma candidates for the OOV type. All possible generators are usable for this. We only use the lemmas that occured with the selected IV types in the training data and combine these with the lemma candidates generated by a rule-based generator (using LCs or ETs) from the OOV type.

5 Experiments

In this section we evaluate the effects of using lexical correspondences and the generation of lemma candidates from similar IV types. We test our approach on Middle Low German (GML). The data comes from the 'Reference Corpus Middle Low German/ Low Rhenish (1200-1650)' (ReN) (Peters and Nagel, 2014).[12] We use two texts: Johannes (19,641 tokens) as training data and Griseldis (9,057 tokens), that we split into two nearly equal parts, as development set (4,505 tokens) and test set (4,552 tokens). Full bibliographical information is given in the bibliography.[13] To assess the difficulty of lemmatizing GML, we compare our results with the accuracy on Modern German (DEU) newswire texts. For this, we use the TIGER corpus (Release 2.2) (Brants et al., 2004) with the same splits as Müller and Schütze (2015). In order to make the tasks on GML and DEU more comparable, we limit the training data to roughly 20,000 tokens and lowercase all types and lemmas in both datasets.

Examples for word-internal modification in DEU have been given in Figure (1). An example for spelling variation in GML is the pair of types *vigenbome* and *vighenbome* "(the) fig tree.SG.DAT" (Johannes). The corresponding lemma *vîgenbôm* also illustrates a special convention in the lemmatization of the GML texts: diacritics are added. In this case they denote the length of the vowels. These diacritics have the same effect as word-internal modification for the lemmatization.[14]

We evaluated the effects of different parameter settings on the development set.[15] The numbers in this section report the performance of selected settings on the test set measured on tokens.

[12] Note that the corpus is still under construction. The tokenization and the annotations used are prefinal. Therefore, the size of the texts might deviate from the numbers given elsewhere.

[13] We do not train and evaluate the lemmatization accuracy on splits of the same text as we are interested in the performance of the lemmatization in the situation were a set of lemmatized texts exists for training and the obtained model is applied to a new text.

[14] The lemmas also contain numbers to disambiguate meanings. Since this adds a word-sense-disambiguation task to the lemmatization, they have been removed for the experiments.

[15] The results can be found in Appendix B.

5.1 Coverage experiments

First, we look at the coverage of the different generators described in the previous sections, i.e. the number of tokens for which the generated set of lemma candidates contains the correct lemma, in other words, the accuracy given an oracle that chooses the correct lemma. We train the generators on the training set. In addition to the lemma candidates generated by the rules induced from the training data, we always extend the set of lemma candidates by all the lemmas that an IV type appears with in the training data.

Coverage should be increased because it sets an upper bound to the lemmatization accuracy. At the same time, the average size of the candidate sets should be kept as small as possible, to make the task of the ranker easier, i.e. choosing the correct lemma in real-life settings without an oracle.

We experimented with using only rules that appear at least n times with type-lemma pairs in the training data. In addition, we tested whether a POS-tag dependent application of the rules would limit the amount of overgeneration. We found that using all rules POS-tag dependently gave the best trade-off between coverage and average candidate set size on the development set.[16]

Table 1 and 2 give the results from the experiments on the test data. Table 1 contains a comparison between DEU and GML. The results show that lemmatizing GML is harder than lemmatizing DEU. Given a similar amount of training data (about 20,000 tokens), there is a difference of about 24.5% in the coverage between the two languages – using ETs the coverage drops from 98.92% for DEU to 74.3% for GML. While for DEU all of the generators reach a high coverage with a small average number of lemma candidates ($\varnothing$ cand.) the coverage for GML is significantly lower with an average size of the candidate sets that is more than three times larger than for DEU. The reason for the drop in coverage and the increase in the average number of lemma candidates might be more word internal modifications and the existence of spelling variation in GML. Following, we present the improvements coming from the methods we introduced to deal with word-internal modification and spelling variation.

[16]We used gold tags for the training and evaluation. When using predicted POS tags, the performance of POS-tag dependent candidate generation depends on the quality of the predictions.

Word-internal modification. Next to the results for ETs, Table 1 gives the coverage results for lexical correspondences (LC) and lexical correspondences with anchored insertions (LC-AI, see section 3) on the test data. The improvements in coverage coming from the better modeling of word-internal modifications are the same for LCs and LC-AIs for both languages while LC-AIs effectively reduce the number of wrongly generated lemma candidates compared with pure LCs. This is especially visible for GML.

For DEU, using LC(-AI)s leads to a small improvement for OOV words of 0.41% which is an error reduction of about 10%. Given the homogeneity of data in the TIGER corpus, this only leads to an overall improvement of the coverage of 0.12%. These numbers decrease further when more training material is used. Using about 100,000 tokens from the TIGER corpus for training, the coverage goes up to 99.50% (OOV: 97.84%) with ETs and 99.54% (OOV: 98%) with LC-AIs. However, the numbers indicate that even languages with moderate word-internal modification can benefit from the usage of LCs, especially when the amount of training data is limited and the lemmatizer has to deal with a large number of OOV types. As has been expected, the effect of using LC(-AI)s is bigger for GML, leading to an overall increase of 1.25% points. However, the gap in performance between DEU and GML remains huge.

Spelling Variation. The additional complexity of the task on GML is at least partially due to spelling variation. To deal with this, we carried out experiments with a regularized[17] version of the data and compare it with the generation of lemma candidates from similar IV types described in Section 4. The regularized version of the data is created using a rule-based approach with 26 handcrafted rewrite rules (in the form of regular expressions and substitutions), which was created by experts on GML for the purpose of reducing the spelling variation.[18] Like before, lemma candi-

[17]We follow Barteld et al. (2015) by using the term regularization, as normalization is usually used to describe a mapping to a standardized or modern variety of the language which is not the case here.

[18]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)'.

		all		oov	
Data	**Generator**	**Coverage (%)**	**⌀ cand.**	**Coverage (%)**	**⌀ cand.**
DEU	ET	98.92	2.79	96.65	3.69
	LC-AI	99.04	3.20	97.06	4.64
	LC	99.04	4.20	97.06	7.04
GML	ET	74.30	10.73	24.06	14.78
	LC-AI	75.55	14.12	27.60	22.45
	LC	75.55	25.17	27.60	47.53

Table 1: Coverage statistics
Coverage (%): number of tokens for which the candidate set contains the correct lemma; ⌀ cand.: the average size of the candidate sets.

dates are generated using all LC-AIs learned from the training data POS-tag dependently. For IV types, all their lemmas from the training data are added as well.

Firstly, we determined an upper bound by adding all lemmas that appeared in the training data to the candidate sets. Table 2 shows that the coverage increases from 74.3% to 88.31% using our method. This is a potential gain of about 14% – 7.58% more than with regularization (80.73%). However, using all lemma candidates for OOV types increases the average candidate set size to 302.46.

Secondly, we tested the trade-off between coverage and set size for our method of generating lemma candidates only from similar IV types (cf. Section 4). Table 2 gives examplary results on the test set.

We explored the effects of different paramater settings on the development set. For the Levenshtein distance, we used the maximal distance as parameter and Levenshtein automata (Schulz and Mihov, 2002) for finding candidates efficiently.[19] For the Jaccard Index, we varied the minimal similarity (between 0 and 0.7 in steps of 0.1, adding 0.25 and a smaller step size of 0.11 between 0.1 and 0.2), the minimal size of character *n*-grams ($\{1, 2, 3, 4\}$), the maximal size of *n*-grams ($\{2, 3, 4, \infty\}$) and the maximal size of skips ($\{0, 1, 2, 3, 4\}$). Furthermore, we optionally applied the frequency-based weighting on the similarity features.

To improve the precision, we calculated the probability of the possible spelling variants returned by the best parameter settings for different thresholds on the set size ($\{15, 16, 17, 18, 19, 20\}$), using the product of the $P(e)$ estimated by Equation 2. We tested different thresholds on the probability below which the pair was excluded (between 0.5 and 0.2, decreasing by 0.1 and decreasing by 0.025 between 0.2 and 0).

The Levenshtein distance is an easy to use method, leading to good results with a distance of 1 or 2. Using a distance of 1 already leads to a better coverage than the regularization with only a small increase in the average set size. The Jaccard Index has more parameters. With tuning them, it is possible to reach better coverage for any given upper bound on the average set size than with Levenshtein. In sum, we get best results by using the Jaccard Index with a small similarity threshold, *n*-grams up to the length of the type, allowing skips in the *n*-grams, and weighting the features by their inverse frequency. In addition, using the probabilities for edit operations to exclude unlikely pairs helped to improve precision.

5.2 Lemmatization accuracy

In contrast to the oracle setting in the previous section, we present the actual accuracy gain for lemmatization in this section. For evaluation we use the log-linear model described in Müller and Schütze (2015) to select the best lemma candidates from the sets. The authors report state-of-the-art results for a couple of languages among them DEU with this model. We use all the features described there.[20] The only exception is Wikipedia data for GML as this does not exist. We train the

[19] We used the implementation from `https://github.com/universal-automata/liblevenshtein-java`.

[20] We also include morphological tags as we train and lemmatize using gold tags. When using predicted tags, using this feature might hurt the performance as described by Müller and Schütze (2015). For Wikipedia, we use the dump available at `http://cistern.cis.lmu.de/marmot/naacl2015/` (Müller et al., 2015).

Spelling variation handling	all Coverage (%)	all ⌀ cand.	oov Coverage (%)	oov ⌀ cand.
None (ET)	74.30	10.73	24.06	14.78
None (LC-AI)	75.55	14.12	27.60	22.45
Regularization	80.73	13.92	29.58	24.45
Upper	88.31	302.46	68.72	951.33
Levenshtein(1)	83.79	14.52	54.14	23.73
Levenshtein(2)	86.27	17.83	62.14	34.41
Jaccard(0.25,2-∞,0)	81.00	14.46	45.15	23.55
Jaccard(0.25,2-∞,3)	84.73	15.09	57.18	25.59
Jaccard-weighted(0.25,2-∞,3)	84.29	14.75	55.77	24.47
Jaccard-weighted(0.25,2-∞,3), $P \geq 0.05$	84.14	14.58	55.27	23.94

Table 2: Coverage for lemma generation with handling of spelling variation on GML Parameter for Levenshtein: maximal distance; Parameters for Jaccard: minimal similarity, minimal and maximal size of character n-grams, maximal size of skips; P denotes the product of the P(e).

model using the implementation of L-BFGS (Liu and Nocedal, 1989) from MALLET (McCallum, 2002).

For the rule-based generators we compare ETs as they are used in the original version of LEMMING and our LC-AIs both with all rules induced from the training data. In contrast to the experiments in Müller and Schütze (2015), we apply the rules POS-tag dependently.

For the variation handling we tested the generators with the best coverage below different thresholds in candidate set size (increasing by 0.1) on the development set. The accuracy first increases but starts to decrease when the average set size becomes larger than 14.6. This shows that this specific log-linear model cannot exploit the potential of our generators, because it is tailored to the usage of ETs as generators. We selected the generator that led to the best results on the development set.

Table 3 shows the results of the best models. For spelling variation handling, we compare our approach with the rule-based regularization. The oracle experiment has shown that the rule-based regularization does not remove all of the spelling variation. Therefore, we applied our approach to spelling variation handling to the regularized data as well, again choosing the best parameters settings on the development set.

The results show that using LCs to generate candidates leads to better results. As expected from the coverage data (Section 5.1), the DEU data shows only a small but statistically significant increase in accuracy ($\chi^2_1 = 11.40$, $p < 0.001$).[21] GML profits more from using LCs (1.23%; $\chi^2_1 = 52.16$, $p < 0.001$). The handling of spelling variation has a bigger impact than modeling word-internal modification. The ranker cannot exploit the full potential of the generator and performs best with parameter settings that lead to a small increase of the average set size. The best performing model used LC-AIs to generate lemma candidates. The total increase in accuracy with this method is 5.56% ($\chi^2_1 = 34.88$, $p < 0.001$) above the baseline model, i.e. generating lemma candidates using ETs without handling spelling variation. This is comparable to the increase obtained by regularizing the texts before applying the lemmatization with LCs as generator (5.87%). The difference between both methods for handling the spelling variation is not significant ($\chi^2_1 = 0.2$, $p = 0.66$). Combining regularization and handling of spelling variation during lemmatization results in an additional increase of 1.6% ($\chi^2_1 = 21.87$, $p < 0.001$) over the model using LC-AIs with regularized texts, leading to a total improvement of 7.38% over the baseline.

6 Conclusion

We presented two methods for dealing with word-internal modification and spelling variation in lemma candidate generation. Both were implemented and tested in the context of data-driven lemmatization with the program LEMMING.

The experiments showed that a better modeling

[21] Significance has been tested using McNemar's test (McNemar, 1947) with continuity correction (Edwards, 1948).

Data	Generator	Spelling variation handling	correct (%) all	oov
DEU	ET	-	97.76	93.35
	LC-AI	-	97.83	93.57
GML	ET	-	71.86	23.78
	LC-AI	-	73.09	27.25
	ET	Regularization	76.58	36.16
	LC-AI	Regularization	77.64	39.35
	LC-AI	Generator	77.42	41.19
	LC-AI	Regularization+Generator	79.24	44.52

Table 3: Lemmatization accuracy

of word-internal modification leads to small improvements for a language like Modern German that features a moderate amount of word-internal modification (0.22% on OOV types). On a homogenous resource like the TIGER corpus, the overall effect of better coverage of OOV types on the lemmatization accuracy is small (0.07%). However, for languages with more word-internal modification and data with more OOV types the gain is higher. This was shown with a Middle Low German corpus. Here, using lexical correspondences (LC) leads to an increase in accuracy of 1.23%.

For the historical Middle Low German texts handling spelling variation is another important factor in lemma candidate generation. Our language-independent approach to generate lemma candidates from potential IV spelling variants for OOV types leads to an increase of 5.56% in accuracy. In comparison, limiting the spelling variation by preprocessing the data with rewrite rules created manually by language experts leads to an improvement of 5.87%. Combining both methods lead to a total increase of 7.38%.

While these are good improvements of the accuracy, the potential accuracy in terms of coverage is even higher for our data-driven method. However, the actual ranker used in our experiments was not able to exploit this potential. Consequently, there are two possible ways for further research: Firstly, adapting the ranker to our modified generators, or, secondly, to improve the precision of the generators. We plan to concentrate on the second strand for further research.

An alternative solution for the problem of restricting the generative capacity of LCs (see Section 3) might be an anchoring by lexicalization, i.e., adding letters before or/and after insertion as a constant. For instance German *sind*, *sein* "(they) are – (to) be" would lead to the less permissive LC $\langle[\text{'s'}, Y, \text{'d'}], [\text{'se'}, Y]\rangle$. This strategy is similar to adding context to lemmatization rules used by Loponen and Järvelin (2010).

The distance measures for detecting possible spelling variants used in this paper only use string similarities of the types ignoring their distribution in the texts. Barteld et al. (2015) also took context similarity into account by filtering the subset obtained from the string similarity with Brown clusters (Brown et al., 1992), keeping only those IV types which are in the same cluster as the OOV type. This – or other methods to include contextual similarity in the selection of potential spelling variants – is a promising way to improve the precision of the measures.

7 Resources

The paper is created reproducibly using org-mode (`http://orgmode.org`). The org-file and the scripts that where used to run the experiments are available at github (`https://github.com/fab-bar/paper-LaTeCH2016`). This version also includes the appendices.

With this paper, we also release our additions to LEMMING including the generators described in this paper. They are available at github as well (`https://github.com/fab-bar/cistern`).

Acknowledgements

The work of the first and second authors has been funded by the DFG. We would like to thank the anonymous reviewers for their helpful remarks and Andrew Fassett for improving our English. All remaining errors are ours.

Primary data

Johannes *Buxtehuder Evangeliar*. GML manuscript from about 1480. Transcribed in the DFG-funded project "Referenzkorpus Mittelniederdeutsch / Niederrheinisch (1200-1650)".

Griseldis *Griseldis / Sigismunda und Guiscardus*. GML print of two tales from 1502. Transcribed in the DFG-funded project "Referenzkorpus Mittelniederdeutsch / Niederrheinisch (1200-1650)".

References

Fabian Barteld, Ingrid Schröder, and Heike Zinsmeister. 2015. Unsupervised regularization of historical texts for POS tagging. *Proceedings of the Workshop on Corpus-Based Research in the Humanities (CRH)*, pages 3–12.

Marcel Bollmann. 2012. (Semi-)automatic normalization of historical texts using distance measures and the Norma tool. In Francesco Mambrini, Marco Passarotti, and Caroline Sporleder, editors, *Proceedings of the Second Workshop on Annotation of Corpora for Research in the Humanities (ACRH-2)*, pages 3–14.

Sabine Brants, Stefanie Dipper, Peter Eisenberg, Silvia Hansen-Schirra, Esther König, Wolfgang Lezius, Christian Rohrer, George Smith, and Hans Uszkoreit. 2004. TIGER: Linguistic interpretation of a German corpus. *Research on Language and Computation*, 2(4):597–620.

Peter F. Brown, 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.

Grzegorz Chrupała. 2006. Simple data-driven contextsensitive lemmatization. *Procesamiento del Lenguaje Natural*, 37:121–127.

Grzegorz Chrupała. 2008. *Towards a machine-learning architecture for lexical functional grammar parsing*. Ph.D. thesis, Dublin City University.

Lee R. Dice. 1945. Measures of the amount of ecologic association between species. *Ecology*, 26(3):297–302.

Allen L. Edwards. 1948. Note on the "correction for continuity" in testing the significance of the difference between correlated proportions. *Psychometrika*, 13(3):185–187.

Leo Egghe. 2010. Good properties of similarity measures and their complementarity. *Journal of the American Society for Information Science and Technology*, 61(10):2151–2160.

Jacob Eisenstein. 2013. What to do about bad language on the internet. In *Proceedings of the North American Chapter of the Association for Computational Linguistics (NAACL)*, pages 359–369.

Sean A. Fulop and Sylvain Neuvel. 2013. Networks of morphological relations. In *Proceedings of the International Symposium on Artificial Intelligence and Mathematics (ISAIM-2014)*.

Andrea Gesmundo and Tanja Samardžić. 2012. Lemmatisation as a tagging task. In *Proceedings of the 50th Annual Meeting of the Association for Computational Linguistics: Short Papers-Volume 2*, pages 368–372.

Nabil Hathout. 2014. Phonotactics in morphological similarity metrics. *Language Sciences*, 46, Part A:71–83.

Ning Jin. 2015. NCSU-SAS-Ning: Candidate generation and feature engineering for supervised lexical normalization. In *Proceedings of the Workshop on Noisy User-generated Text*, pages 87–92.

Bart Jongejan 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: Volume 1*, pages 145–153.

Sebastian Kempken. 2005. *Bewertung historischer und regionaler Schreibvarianten mit Hilfe von Abstandsmaßen*. Ph.D. thesis, Universität Duisburg-Essen.

Mike Kestemont, Walter Daelemans, and Guy De Pauw. 2010. Weigh your words—memory-based lemmatization for Middle Dutch. *Literary and Linguistic Computing*, 25(3):287–301.

Tuomo Korenius, Jorma Laurikkala, Kalervo Järvelin, and Martti Juhola. 2004. Stemming and lemmatization in the clustering of Finnish text documents. In *Proceedings of the Thirteenth ACM International Conference on Information and Knowledge Management*.

Michael Levandowsky and David Winter. 1971. Distance between sets. *Nature*, 234(5323):34–35.

Vladimir Levenshtein. 1966. Binary codes capable of correcting deletions, insertions, and reversals. *Soviet Physics Doklady*, 10:707–710.

Dong C. Liu and Jorge Nocedal. 1989. On the limited memory BFGS method for large scale optimization. *Mathematical Programming*, 45(1-3):503–528.

Pavel Logačev, 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)*, pages 103–112.

Aki Loponen and Kalervo Järvelin. 2010. A dictionary-and corpus-independent statistical lemmatizer for information retrieval in low resource languages. In Maristella Agosti, Nicola Ferro, Carol Peters, Maarten de Rijke, and Alan Smeaton, editors, *Multilingual and Multimodal Information Access Evaluation*, pages 3–14. Springer, Berlin and Heidelberg.

Andrew K. McCallum. 2002. MALLET: A machine learning for language toolkit. (http://mallet.cs.umass.edu).

Quinn McNemar. 1947. Note on the sampling error of the difference between correlated proportions or percentages. *Psychometrika*, 12(2):153–157.

Thomas Müller and Hinrich Schütze. 2015. Robust morphological tagging with word representations. In *Proceedings of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies (NAACL: HLT)*, pages 526–536.

Thomas Müller, 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)*, pages 2268–2274.

Sylvain Neuvel and Sean A. Fulop. 2002. Unsupervised learning of morphology without morphemes. In *Proceedings of the ACL-02 workshop on Morphological and phonological learning-Volume 6*, pages 31–40.

Robert Peters and Norbert Nagel. 2014. Das digitale ‚Referenzkorpus Mittelniederdeutsch / Niederrheinisch (ReN)'. *Jahrbuch für Germanistische Sprachgeschichte*, 5(1):165–175.

Thomas Pilz. 2009. *Nichtstandardisierte Rechtschreibung - Variationsmodellierung und rechnergestützte Variationsverarbeitung*. Ph.D. thesis, Universität Duisburg-Essen.

Klaus Schulz and Stoyan Mihov. 2002. Fast string correction with Levenshtein-automata. *International Journal of Document Analysis and Recognition*, 5:67–85.

Rico Sennrich and Beat Kunz. 2014. Zmorge: A German morphological lexicon extracted from Wiktionary. In *Proceedings of the 9th International Conference on Language Resources and Evaluation (LREC 2014)*, pages 1063–1067.

Hans van Halteren 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.