(Semi-)Automatic Normalization of Historical Texts using Distance Measures and the Norma tool

Marcel Bollmann

Department of Linguistics
Ruhr-Universität Bochum
E-mail: bollmann@linguistics.rub.de

Abstract

Historical texts typically show a high degree of variance in spelling. Normalization of variant word forms to their modern spellings can greatly benefit further processing of the data, e.g., POS tagging or lemmatization. This paper compares several approaches to normalization with a focus on methods based on string distance measures and evaluates them on two different types of historical texts. Furthermore, the Norma tool is introduced, an interactive normalization tool which is flexibly adaptable to different varieties of historical language data. It is shown that a combination of normalization methods produces the best results, achieving an accuracy between 74% and 94% depending on the type of text.

1 Introduction[1]

Historical language data poses a difficult challenge for natural language processing. One of the biggest problems is the lack of standardized writing conventions: the exact characters and symbols used to spell a word can depend on many different factors such as dialectal influences, space constraints on a page, or personal preferences of the writer. Consequently, while annotation tasks such as part-of-speech (POS) tagging are well explored for modern languages, with reported accuracies as high as 97% (Brants [5]), the same cannot be said for historical varieties. In a study done on Early New High German (Scheible et al. [12]), tagging accuracies using taggers trained on modern German were below 70%. A possible solution is a pre-processing step during which word forms are converted to their modern spellings, which the same study found to increase tagging accuracy by 10 percentage points. This process will be referred to here as 'normalization'.

[1]The research reported here was financed by Deutsche Forschungsgemeinschaft (DFG), Grant DI 1558/4-1.

When creating a corpus of historical texts, manually normalizing all data can be a tedious and time-consuming process. Furthermore, despite the many inconsistencies, spelling is typically not arbitrary, and many regularities can still be found. In Early New High German (ENHG), typical examples are *'v'* for (modern) *'u'*, as in *vnd – und* 'and', or *'y'* for (modern) *'i'*, as in *seyn – sein* 'be'. This leads to the question of how the process of normalization can be automated. Several methods for automatic normalization have been proposed (some of the more recent ones are, e.g., Baron et al. [3], Jurish [9], Bollmann et al. [4]), though no clear "best" method has been identified. On the contrary, it is unclear to which extent the results are even comparable, as the amount and degree of spelling variance can be expected to differ greatly between texts. At the same time, this variance suggests the need for normalization methods which are flexibly adaptable to different types of texts.

This paper addresses these issues in two ways: (1) by comparing different normalization methods that can be flexibly trained to handle different spelling varieties; and (2) by presenting the interactive normalization tool Norma, which provides a framework to combine, train, and compare different methods of normalization.

The structure of this paper is as follows. Sec. 2 presents a rule-based method for normalization, while Sec. 3 describes a normalization approach based on string distance measures. Sec. 4 introduces the Norma tool. In Sec. 5, all presented normalization methods are evaluated on two different types of historical texts. Sec. 6 discussed the comparable VARD tool, Sec. 7 presents related work and further conceivable approaches to the normalization task, and Sec. 8 concludes.

2 Rule-Based Method

Rule-based normalization will be used as a comparison to the distance measures described in Sec. 3. Its main concepts are only briefly summarized here; a detailed description can be found in Bollmann et al. [4].

The rule-based method applies a set of character rewrite rules to a historical input string in order to produce its normalized variant. Rewrite rules operate on one or more characters and take their immediate context into account; an example rule is shown in Ex. (1).

(1) `j → ih / # _ r`
('j' is replaced by 'ih' between a word boundary ('#') and 'r')

The rules are not meant to be specified manually, but are supposed to be learned from training data (using a modified Levenshtein algorithm). To normalize an input string, it is processed from left to right, with one rewrite rule being applied at each position. Typically, there will be several applicable rules at each given position; in this case, the frequency of a rule during training determines how likely it is to be used during the normalization process. In case there is no applicable rule, the

default is to leave the character at that position unchanged. Additionally, to prevent the generation of nonsense words, all normalization candidates are matched against a target lexicon. Therefore, it is possible that the rule-based method sometimes fails to find any normalization candidate at all.

3 Distance Measures

Levenshtein distance between two strings is defined as the number of substitutions, insertions, and deletions required to transform one string into the other (Levenshtein [11]). More generally, the term "distance measure" will be used here to describe any function that accepts a pair of strings and returns a positive, numeric value describing their distance.

Any distance measure can theoretically be used for the purpose of normalization. Given a (finite) lexicon of modern word forms, the distance of a historical input word form to each entry in the modern lexicon can be calculated; the word form with the lowest distance to the input string is then chosen as the normalization.[2]

3.1 FlexMetric

FlexMetric (Kempken [10]) is a measure originally designed for rating historical and dialectal spelling variants, which makes it appear ideal for the normalization task. It is supposed to be flexible in the sense that it can easily be adapted to different types of text and language varieties. This is achieved by using a variant of Levenshtein distance with weights, where instead of simply counting the number of edit operations, each such operation is assigned a weight w (also called "cost") with $0 < w \leq 1$. Weights can be individually defined; if $w = 1$ for all edit operations, this approach is identical to standard Levenshtein distance.

However, instead of allowing arbitrary weights, FlexMetric introduces the concept of character groups, which contain characters that are closely related to each other. Conceivable groupings for German are, e.g., *{s, z, ß}* or *{i, y}*. These groups are then assigned individual costs, which in turn define the cost of character substitutions within that same group. Substitutions of characters that do not belong to the same group are assigned a default cost of 1. Note that this approach induces a symmetry: the substitution $i \rightarrow y$ always costs the same as $y \rightarrow i$.

Additionally, in the extended version of the algorithm, which is used here, multiple occurrences of the same character are ignored—e.g., the transformation $n \rightarrow nn$ (and vice versa) always has a cost of 0.

The relative simplicity of this approach is deliberate; the intention was to allow for the creation of a good parametrization by manual inspection of a small portion of the input data and without too much effort (Kempken [10], p. 61). Again, this seems well-suited for normalization, as historical data typically shows a lot

[2] In practice, finding an efficient algorithm for this operation can be a limiting factor, though.

of variety between texts, requiring an easily adaptable normalization method, and training data is typically not available (or not in large amounts).

3.2 MultiWLD

In the Luther text used for evaluation (cf. Sec. 5), the substitution $v \rightarrow u$ is by far the most common one; however, the opposite direction, $u \rightarrow v$, is comparatively rare. This leads to the question whether costs should actually be defined in a symmetric way. Furthermore, it is questionable whether characters should be treated in isolation: e.g., *'ck'* often occurs for modern *'k'*, as in *werck – werk* 'work', and does not represent a different phoneme. Treating this example simply as a deletion of *'c'* would be a case of overgeneralization, though. Many similar examples for German can be found ($mb \rightarrow m$, $i \rightarrow ie$, $a \rightarrow ah$, etc.).

These considerations led to the MultiWLD[3] algorithm. It is supposed to be similar to FlexMetric in the sense that costs should be easily definable by intuition and/or manual inspection of a small input sample, but differs in the aspects mentioned above: (1) it is directional (instead of symmetric); and (2) it allows the definition of substitutions with more than one character on either side.

Although these changes draw nearer to the idea of the rule-based approach (cf. Sec. 2), there are two important differences to keep in mind. Firstly, the distance measures described here only assign a cost to actual modifications to the input string; i.e., it is always preferred that a character is left unchanged, which is not the case in the rule-based method. Secondly, while the definition of multi-character substitutions introduces a form of context-sensitivity (e.g., $a \rightarrow ah$ represents the insertion of *'h'* after *'a'*), it is up to the creator of the parametrisation how much and which context, if any, to include in each rule. In contrast, the rule-based approach always requires exactly one character on each side as context.

4 The Norma tool

Norma is a tool for (semi-)automatic normalization of historical language data. It can be used both interactively and non-interactively, and is intended to be flexible in the sense that it is not restricted to any particular method(s) of normalization. Normalization methods—or "normalizers"—are encapsulated in external modules for easy extensibility. Parameter sets of normalizers can be dynamically retrained during the normalization process if the normalizer supports it. This way, Norma supports an incremental learning approach to normalization.

Norma is written in Python 2.7.[4] At the time of writing, only a command-line interface is available.

[3] MultiWLD = Weighted Levenshtein Distance with multi-character substitutions

[4] `http://www.python.org/`

4.1 Modes of operation

Norma supports three modes of operation: batch, interactive, and training mode.

In batch mode, Norma takes an input file consisting of one historical word form per line and, for each line, outputs the suggested normalization of that word form.

Interactive mode works the same way as batch mode, but prompts the user with each generated normalization, which he or she can either confirm or correct. The resulting pair of historic and (confirmed) modern word form is then used to train the normalizers. Ideally, the number of corrections the user has to make should decrease over time as the normalizers are supplied with more training data and can adjust their set of parameters accordingly.

Finally, there is a training mode, which works similarly to interactive mode except that the confirmed modern word form is not entered by the user, but given in the input file itself. Apart from training normalizers from a set of manually normalized data, this mode can also be used to "simulate" the interactive normalization process, e.g. to evaluate the effect of incremental learning.

4.2 Normalizer modules

Norma itself does not implement any normalization or learning techniques, but rather depends on external normalizer modules to provide that functionality.

Normalizers exchange data with Norma via a certain, pre-defined interface of functions. The normalization function is given a historical word form as input and should return its normalized form along with a confidence score. Confidence scores are numerical values between 0 and 1; a score of 0 is taken to mean that no suitable normalization could be found at all. The training function is given a historical word form along with its modern counterpart and is expected to trigger the normalizer's training algorithm.

4.3 Normalization cycle

The normalization process in Norma can be described as a cycle consisting of the following steps:

1. fetching a historical input word form;
2. generating a normalization candidate for the historical word form using one or more normalizers;
3. validating the candidate word form against the correct normalization; and
4. training the normalizers with the historical word form and its correct normalization.

This cycle is repeated until all input word forms have been processed. If training is disabled or impossible, e.g. when batch-processing texts, the last two steps will do nothing. Fig. 1 shows a flow diagram visualizing the normalization cycle.

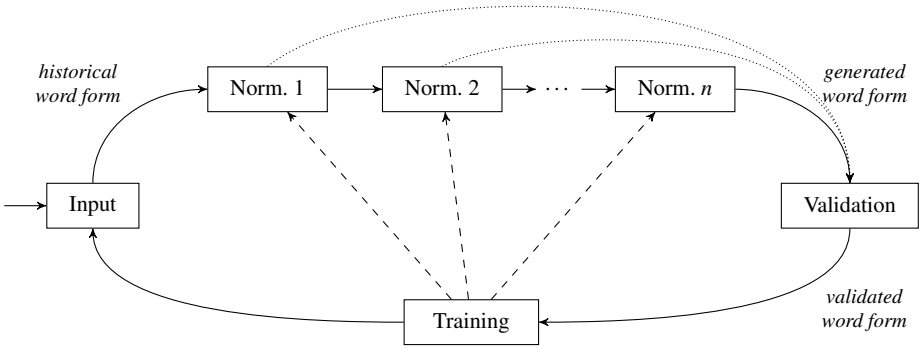

Figure 1: Flow diagram of a normalization cycle (with training) in Norma

The first step is the retrieving of a historical word form as input. Typically, this will be read from a text file, from which lines are processed in sequential order.

The second step in the cycle is the actual normalization process. It is modelled as a chain: at the beginning, the first normalizer in the chain is called and supplied with the historical word form; if it finds a suitable normalization with a confidence score above a pre-defined threshold, the chain is interrupted and processing continues with the next step (indicated in Fig. 1 by the dotted-line arrows). Otherwise, the next normalizer in the chain is called. If all normalizers fail to produce an acceptable result, the unmodified original word form is used as the normalization candidate.

Afterwards, the normalization candidate is validated against the correct normalization. In interactive mode, the user is prompted to either confirm the candidate form or correct it, while in training mode, the correct form is read directly from the input file.

As the last step of the cycle, the training function of all normalizers is called, with the historic word form and its (confirmed) modern equivalent as input. After the normalizers have been trained, the cycle starts anew.

5 Evaluation

5.1 Corpora

For the evaluation, two different texts from ENHG were used: (1) the Luther bible in its version from 1545 and a revised modern version of it; and (2) a manuscript of the Anselm corpus from the Eastern Upper Germany dialectal area[5] that was manually normalized by a student assistant. A more detailed description is given in Bollmann et al. [4].

[5]The depository of the manuscript is Melk, Austria.

		Luther		Anselm	
Training	Total tokens	218,504	—	500	—
Evaluation	Total tokens	109,258	(100.00%)	4,020	(100.00%)
	Identical tokens	71,163	(65.13%)	1,512	(37.61%)
	Maximum accuracy	103,409	(94.65%)	3,760	(93.53%)

Table 1: General corpus statistics, split up by training and evaluation part. "Identical tokens" are counted between the original text and its modern version; "maximum accuracy" gives the highest accuracy that can be achieved with context-free normalization methods.

Table 1 shows some general corpus statistics. The Luther bible is a comparatively large text, with 109,258 tokens in the evaluation part, and is relatively close to its modern version: 65% of all tokens in the historical version are already identical to their modern counterparts. This figure serves as the baseline for the evaluation. The Anselm text, on the other hand, is much smaller (4,020 tokens) and shows a lot more variance in spelling. While the size of the Luther text—and the training part in particular—makes it more suitable for training automatic normalization methods, the Anselm text is closer to actual research scenarios: texts with a high variance in spelling, and without a reasonable amount of training data available (if any).

It is also interesting to note the maximum achievable accuracy, which is around 94% for both texts. All methods evaluated in this paper are context-free, i.e., they operate on word forms in isolation. Consequently, source tokens of the same type will be assigned identical normalizations. However, the same type can be aligned with more than one target token: a typical example is the historical *jn* mapped to either *in* 'in' or *ihn* 'him', depending on context. In this case, it is impossible to achieve perfect accuracy using context-free methods. Therefore, the "maximum accuracy" in Table 1 represents the accuracy that would be achieved if each type was normalized to the token it is most often aligned with.

5.2 Evaluation procedure

Five normalization methods were evaluated: rule-based normalization (Sec. 2), standard Levenshtein distance (Sec. 3), FlexMetric (Sec. 3.1), MultiWLD (Sec. 3.2), and a wordlist mapper. The mapper simply learns word-to-word substitutions from training data without any notion of character distance. If a word form can be mapped to multiple modern word forms, the mapping with the highest frequency is used; in case of a tie, the mapping that was learned first is chosen.

Parametrization for the rule-based method and the wordlist mapper has been learned from separate training parts of the texts as shown in Table 1. For FlexMetric and MultiWLD, parametrization was created by hand after inspection of at most

500 tokens from the training part of each text. As an example, characters groups learned from Luther that were assigned the lowest costs are *{u, v}* and *{i, j}*, while the most common insertions/deletion were of *'e'* and *'h'*. This method is, of course, a highly subjective one and should only be seen as a first experiment on whether a reasonable parametrization can be created by manual inspection of a small text fragment alone.

For the target lexicon, I used the complete vocabulary of the modern Luther bible complemented by a full-form lexicon of all simplices generated by the German morphology DMOR (Schiller [13]).

In addition to the separate evaluation of each method, each possible chain combination (cf. p. 6) has been evaluated. Threshold for confidence scores was set to 0, i.e., unless the normalizer failed to find a normalization, any generated normalization candidate was accepted as the final result. As a consequence, chains in this test setup can consist of three elements at maximum, as the distance-based measures can never fail to find a normalization candidate.

All methods have been implemented as normalizer modules for Norma (cf. Sec. 4.2). Evaluation was done using Norma's training mode to produce parameter files for the rule-based method and the wordlist mapper, and batch mode to process the evaluation texts.

5.3 Results

Evaluation results are shown in Table 2. For the Luther text, the simple wordlist mapper performs best in isolation, achieving an accuracy of 92.60%, with the rule-based approach and MultiWLD being only slightly worse. This is a notable result for the adaptable (or "trained") distance measures in particular, as their parametrization was compiled manually and with much less training data. Combining the approaches yields an even better performance, up to a maximum of 93.72% when the mapper is used first, the rule-based approach second, and either MultiWLD or FlexMetric last. However, any chain that starts with the wordlist mapper results in a comparable score, making it hard to determine a single "best" combination here. It should be noted that these scores are very close to the maximum accuracy of 94.65%, showing that these approaches—and the chain configurations in particular—are almost optimal for the Luther text.

Results for the Anselm text are considerably worse than for Luther; again, this is to be expected, as the baseline is significantly lower (37.61%). Additionally, much less training data was used for the mapper and the rule-based method. Their scores fall below those of the trained distance measures, with MultiWLD now achieving the best accuracy (67.91%) and FlexMetric following close behind. This suggests that trained distance measures outperform both wordlist mappings and the rule-based approach when only a small amount of training data is available.

Combining methods results in an even bigger increase in accuracy with the Anselm text. The best result (73.63%) is achieved by using only the mapper and MultiWLD, although the configuration that was best for the Luther text also

	Luther		Anselm	
Baseline	71,163	(65.13%)	1,512	(37.61%)
Mapper	**101,170**	**(92.60%)**	2,448	(60.90%)
Rule-based	98,767	(90.40%)	2,530	(62.94%)
MultiWLD	96,510	(88.33%)	**2,730**	**(67.91%)**
FlexMetric	92,353	(84.53%)	2,655	(66.04%)
Levenshtein	87,619	(80.19%)	2,161	(53.76%)
Mapper → *Rule-based* → *MultiWLD*	**102,193**	**(93.53%)**	2,947	(73.31%)
Mapper → *Rule-based* → *FlexMetric*	**102,193**	**(93.53%)**	2,947	(73.31%)
Mapper → *Rule-based* → *Levenshtein*	102,160	(93.50%)	2,793	(69.48%)
Mapper → *MultiWLD*	102,131	(93.48%)	**2,960**	**(73.63%)**
Mapper → *FlexMetric*	102,118	(93.47%)	2,911	(72.41%)
Mapper → *Levenshtein*	101,867	(93.24%)	2,705	(67.29%)
Rule-based → *Mapper* → *MultiWLD*	98,890	(90.51%)	2,829	(70.37%)
Rule-based → *Mapper* → *FlexMetric*	98,890	(90.51%)	2,800	(69.25%)
Rule-based → *Mapper* → *Levenshtein*	98,857	(90.48%)	2,675	(66.54%)
Rule-based → *MultiWLD*	98,890	(90.51%)	2,815	(70.02%)
Rule-based → *FlexMetric*	98,890	(90.51%)	2,785	(69.28%)
Rule-based → *Levenshtein*	98,857	(90.48%)	2,660	(66.17%)

Table 2: Accuracies after normalization, with methods evaluated both separately and combined as a chain. "Baseline" gives the accuracy before normalization for comparison. Best results are highlighted in bold.

shows a similar performance here. Again, a clear "winner" cannot be determined. However, the tendency that the wordlist mapper should come first in the chain holds equally for the Anselm text as for Luther, despite the comparatively bad performance of the mapper on Anselm when used in isolation.

Untrained Levenshtein distance always performs worst, which is especially apparent in the separate evaluation. This suggests that it should probably never be used for the normalization task, or at least not in isolation, as a list of edit weights compiled from even a small input sample gives significant boosts in accuracy.

6 Comparison to VARD 2

A well known interactive tool for normalization of historical texts is VARD 2 (Baron and Rayson [2]). It was designed for use with Early Modern English texts and also combines different normalization methods. VARD is written in Java and is freely available for academic research.[6]

In contrast to Norma, VARD features a graphical user interface for interactive processing of texts. When a text is loaded into the tool, historical spelling variants are automatically detected and highlighted. Right-clicking on a variant word form

[6] http://www.comp.lancs.ac.uk/~barona/vard2/

shows a list of suggested normalization candidates; the user can choose to either normalize all occurrences of a word form or one particular instance only. VARD clearly offers much more convenience here. However, its approach is not functionally different, i.e., the same kind of user interface and functions could also be built on top of Norma.

The biggest difference between the tools, however, lies in the customizability of the normalization methods. VARD combines four different methods: 'known variants', which is similar to the wordlist mapper described in Sec. 5.2; 'letter replacement', which applies a list of letter replacement rules to a word; 'phonetic matching', which uses a modified Soundex algorithm to find modern word forms that are phonetically similar to the input word; and standard edit distance. Additionally, all candidate word forms are matched against a modern lexicon.

This set of normalization methods is fixed and can be neither reduced nor extended. Also, while some of these components can be modified, the phonetic matching algorithm is hard-coded to the phonetics of English and can neither be changed nor completely turned off. Consequently, when working with texts in other languages, this algorithm is most likely to produce incorrect results. Norma, on the other hand, places no restrictions on the use and combination of normalizers, which should enable an easier adaption to different languages.

Nevertheless, there has been work on adapting VARD to other languages, such as Portuguese (Hendrickx and Marquilhas [8]). To be able to compare VARD's performance with Norma's, a similar adaption has been tried for the German data. To this end, VARD has been given the same modern lexicon and the same letter replacement rules as the MultiWLD algorithm in the former evaluation (cf. Sec. 5.2). The training part of each text was then used to run VARD's training mode. Finally, batch mode with a threshold of 0%[7] was used to normalize the evaluation parts.

Final accuracies were 91.19% (Luther) and 69.38% (Anselm), both worse than the best scores in the evaluation with Norma (cf. Table 2). However, VARD performed better on the Anselm text than any normalization method in isolation. Interestingly, it performed worse on Luther than the wordlist mapper alone, despite having learned the same mappings from the training text.

7 Related Work and Outlook

Previous work on historical spelling variation tends to focus on the task of information retrieval (IR), which is basically reverse to normalization: finding historical variants that correspond to a modern input string (e.g., Baron et al. [3], Ernst-Gerlach and Fuhr [6], Hauser and Schulz [7]). It is unclear how well these approaches translate to the normalization task with a view to further processing of the data by NLP tools such as POS taggers.

[7] The threshold is the minimum F-score required for a normalization candidate to be used; a setting of 0% ensures that all variant word forms will be normalized, which is closest to the behaviour of Norma. Also, the default setting of 50% was found to perform significantly worse.

For future work, more distance measures could be included in the evaluation. Zobel and Dart [14] test different methods in the context of spelling correction and name matching, and find an n-gram string distance to be effective. Jurish [9] uses token context information and a combination of different methods via a hidden Markov model to normalize historical German, achieving high performance, though also evaluated as an IR task. Context information has not yet been considered for Norma, but is a possible line of future research.

Incremental learning is a core feature of Norma, but has so far been neglected in the evaluation. Parameter sets for FlexMetric and MultiWLD have been compiled manually with good results; it remains to be tested whether similar results can be achieved when learning parameters automatically from training data, and if so, how much training data is needed. Adesam et al. [1] describe a pilot study using automatically extracted replacement rules with promising results.

Finally, normalization also bears similarities to machine translation, in which words (to be normalized) correspond to sentences (to be translated) and characters correspond to words. This is especially apparent in the rule-based method (cf. Sec. 2) where character identities must be learned in the same way as substitutions, and where context information is taken into account, similar to the implementations of statistical translation models.

8 Conclusion

In this paper, several approaches to normalization were discussed and evaluated against two different historical texts. When used in isolation, the rule-based method and a simple wordlist mapper performed best on the Luther text, where a large amount of training data was available. Trained distance measures performed best on the Anselm text, where only a small amount of training data was used and which showed considerably more variation in spelling. In all cases, a combination of these methods proved to be best, achieving an accuracy of up to 93.72% for Luther and 73.63% for Anselm. In particular, the results showed that integrating a simple word-to-word mapper always increased accuracy.

Evaluation was performed using Norma, an interactive normalization tool which is easily extensible and more flexible than the comparable VARD tool. Norma was used here for its capability of employing, training, and combining different normalization methods, though the possibility of semi-automatic normalization using incremental learning techniques remains to be explored further.

References

[1] Yvonne Adesam, Malin Ahlberg, and Gerlof Bouma. *bokstaffua, bokstaffwa, bokstafwa, bokstaua, bokstawa...* Towards lexical link-up for a corpus of Old Swedish. In *Proceedings of KONVENS 2012 (LThist 2012 workshop)*, pages 365–369, Vienna, Austria, 2012.

[2] Alistair Baron and Paul Rayson. VARD 2: A tool for dealing with spelling variation in historical corpora. In *Proceedings of the Postgraduate Conference in Corpus Linguistics*, 2008.

[3] Alistair Baron, Paul Rayson, and Dawn Archer. Automatic standardization of spelling for historical text mining. In *Proceedings of Digital Humanities 2009*, Maryland, USA, 2009.

[4] Marcel Bollmann, Florian Petran, and Stefanie Dipper. Applying rule-based normalization to different types of historical texts — an evaluation. In *Proceedings of LTC 2011*, pages 339–344, Poznan, Poland, 2011.

[5] Thorsten Brants. TnT — a statistical part-of-speech tagger. In *Proceedings of ANLP 2000*, pages 224–231, Seattle, USA, 2000.

[6] Andrea Ernst-Gerlach and Norbert Fuhr. Generating search term variants for text collections with historic spellings. In *Proceedings of the 28th European Conference on Information Retrieval Research (ECIR 2006)*. Springer, 2006.

[7] Andreas W. Hauser and Klaus U. Schulz. Unsupervised learning of edit distance weights for retrieving historical spelling variations. In *Proceedings of FSTAS 2007*, pages 1–6, Borovets, Bulgaria, 2007.

[8] Iris Hendrickx and Rita Marquilhas. From old texts to modern spellings: an experiment in automatic normalisation. *JLCL*, 26(2):65–76, 2011.

[9] Bryan Jurish. More than words: using token context to improve canonicalization of historical German. *Journal for Language Technology and Computational Linguistics*, 25(1):23–39, 2010.

[10] Sebastian Kempken. *Bewertung historischer und regionaler Schreibvarianten mit Hilfe von Abstandsmaßen*. Diploma thesis, Universität Duisburg-Essen, 2005.

[11] Vladimir I. Levenshtein. Binary codes capable of correcting deletions, insertions, and reversals. *Soviet Physics Doklady*, 10(8):707–710, 1966.

[12] Silke Scheible, Richard J. Whitt, Martin Durrell, and Paul Bennett. Evaluating an 'off-the-shelf' POS-tagger on Early Modern German text. In *Proceedings of LaTeCH-2011*, pages 19–23, Portland, USA, 2011.

[13] Anne Schiller. Deutsche Flexions- und Kompositionsmorphologie mit PC-KIMMO. In Roland Hausser, editor, *Proceedings of the first Morpholympics*, Tübingen, 1996. Niemeyer.

[14] Justin Zobel and Philip Dart. Finding approximate matches in large lexicons. *Software: Practice and Experience*, 25(3):331–345, 1995.