Weigh your words—memory-based lemmatization for Middle Dutch

Mike Kestemont
Institute for the Study of Literature in the Netherlands (ISLN) and University of Antwerp, Belgium

Walter Daelemans and Guy De Pauw
CLiPS Computational Linguistics Group, University of Antwerp, Belgium

Abstract

This article deals with the lemmatization of Middle Dutch literature. This text collection—like any other medieval corpus—is characterized by an enormous spelling variation, which makes it difficult to perform a computational analysis of this kind of data. Lemmatization is therefore an essential preprocessing step in many applications, since it allows the abstraction from superficial textual variation, for instance in spelling. The data we will work with is the *Corpus-Gysseling*, containing all surviving Middle Dutch literary manuscripts dated before 1300 AD. In this article we shall present a language-independent system that can 'learn' intra-lemma spelling variation. We describe a series of experiments with this system, using Memory-Based Machine Learning and propose two solutions for the lemmatization of our data: the first procedure attempts to *generate* new spelling variants, the second one seeks to implement a novel string distance metric to better *detect* spelling variants. The latter system attempts to rerank candidates suggested by a classic Levenshtein distance, leading to a substantial gain in lemmatization accuracy. This research result is encouraging and means a substantial step forward in the computational study of Middle Dutch literature. Our techniques might be of interest to other research domains as well because of their language-independent nature.

Correspondence:
Mike Kestemont,
Universiteit Antwerpen,
Stadscampus, Prinsstraat 13,
Room D.118, 2000
Antwerpen, Belgium.
E-mail:
mike.kestemont@ua.ac.be

1 Spelling Variation in Middle Dutch

Middle Dutch is a typical example of a historical language displaying a considerable amount of spelling variation (Van der Voort van der Kleij, 2005; Ernst-Gerlach and Fuhr, 2006; Kestemont and Van Dalen-Oskam, 2009; Souvay and Pierrel, 2009). Especially before the advent of the printing press, there existed no standard language variety of Dutch, let alone a standard spelling. As such, medieval Dutch spelling was generally highly phonological and 'personal' in nature, since it would represent each writer's own dialectal pronunciation and local spelling habits. That is why even highly frequent words could be spelled in very different ways, reflecting the abundant variety of dialects and local substandards then found in the Low Countries (Fig. 1).

This spelling variation makes it difficult to process medieval texts in any computational application. For instance for authorship attribution, it

Literary and Linguistic Computing, Vol. 25, No. 3, 2010. © The Author 2010. Published by Oxford University Press on behalf of ALLC and ACH. All rights reserved. For Permissions, please email: journals.permissions@oxfordjournals.org
doi:10.1093/llc/fqq011 Advance Access published on 4 August 2010

Downloaded from http://llc.oxfordjournals.org/ at UB Leipzig on January 17, 2012

- *allographs*: certain graphemes (characters or combinations of them) could be used interchangeably without any difference in meaning (e.g. *vvas* :: *was* or *wiit* :: *wijt*)
- *dialectal differences*: writing would reflect a writer's dialect (e.g. *ben* :: *bem* ; *verslagen* :: *verslegen*)
- *clitics:* tokens would often contain clitic components or small affixes that nowadays would be considered another word (tsweerd :: dat sweerd)
- *spelling habits:* some texts display the (inconsequent) use of local or supra-regional spelling conventions (e.g. lengthening graphemes to express the longitude of a vowel: *daar* :: *dar* :: *daer* :: *dair*)

Fig. 1 Overview of some phenomena causing spelling variation in Middle Dutch texts

has been shown that spelling variation hinders our capturing of an original author's style (Van Dalen-Oskam and Van Zundert, 2007; Kestemont and Van Dalen-Oskam, 2009). Medieval manuscripts have often only survived in copies of the original author's text, which gave subsequent scribes the opportunity to freely adapt texts to their own wishes. Stylometric analyses that are restricted to superficial features (such as character *n*-grams) often prove to be affected by the scribe of a particular manuscript and convey little or no information on the authorship of the original.

For Middle Dutch literature, it has indeed been shown that authorial features have only been left intact on a deeper (e.g. lexical) level (Kestemont and Van Dalen-Oskam, 2009). Consequently, analyses should move away from the surface of the texts and focus on deeper textual features, such as lemmas, to yield robust results. Language technology for Middle Dutch is still in its infancy, although researchers in the field of computational Middle Dutch Studies are in great need of it (Van Dalen-Oskam and Van Zundert, 2007, p. 352). With this article, we hope to partially remedy this deficiency.

2 Lemmatization: Related Research

Lemmatization is an intuitively simple task: we all perform it whenever we look up a word in a dictionary. It refers to the process of assigning a label, 'lemma' or dictionary headword to a language token (Knowles and Mohd Don 2004, p. 70; Lyras *et al.*, 2008, p. 1043ff). This process enables the subsequent generalization 'about the behaviour of groups of words in cases where their individual differences are irrelevant' (Knowles and Mohd Don, 2004, p. 69). As such, lemmatization implies mapping tokens that only differ in 'inflection and/or spelling' (Francis and Kucera, 1982, p. 1) to the same 'abstract representation' that, as such, comes to subsume 'all the formal lexical variations which may apply' (Crystal, 1997).

Research so far has focused on modern languages with little or no spelling variation (recent overviews in Lyras *et al.*, 2008; Chrupała, 2009). Apart from early stemming techniques (such as Porter 1980), researchers have often turned to string distance metrics for the task: given a certain input token, a string distance measure is used to retrieve similar tokens or lemmas from a word list or lexicon in order to lemmatize the input token (e.g. Lyras *et al.*, 2008). It is characteristic of many of these approaches that they exploit the graphemic similarity between the input token and the desired lemma. Lyras *et al.* (2008), for instance, compared the performance of Levenshtein's edit distance (1966) and Dice's coefficient (1945) for this task. Machine learning techniques have been applied to the problem as well, in which case the task was often considered as a multi-label classification task (a.o. Chrupała 2009; Toutanova and Cherry 2009; Van den Bosch and Daelemans 1999).

Interestingly, what these researchers try to 'predict' for a given input token are rarely the lemmas

Downloaded from http://llc.oxfordjournals.org/ at UB Leipzig on January 17, 2012

themselves but rather an 'edit script' that allows the input token to be edited or transformed into the desired normalized form (Van den Bosch and Daelemans, 1999; Erjavec and Dzeroski, 2004; De Pauw and de Schryver, 2008; Groenewald, 2009; Daelemans *et al.*, 2009). A simple example can be borrowed from Daelemans *et al.*, 2009; to transform the Afrikaans form *geslaap* into the appropriate lemma SLAAP, the following script needed is: *delete initial ge-*. This 'scripts-as-label' approach is attractive. First, it considerably reduces the number of entities needed in classification (a small amount of edit operations versus all of the lemmas in a corpus). Secondly, this method can deal with unknown class labels or lemma labels that were not encountered during training. Even if a test token's 'gold lemma' hasn't been previously observed, chances exist that the algorithm will find it anyway through the prediction of the right edit script.

There has been some research into spelling normalization and lemmatization for historic languages and a way of dealing with *historical spelling variants* or HSVs (Reynaert, 2005; Kempken *et al.*, 2006). A recent reference for Old French is the paper by Souvay and Pierrel (2009), in which the authors propose a system for the lemmatization of Old French. For other Romance languages, research has been limited (Giusti *et al.*, 2007 for historical Portuguese). Most of the available research has been carried out for historical stages of English and German. Pilz *et al.* (2008) offer an interesting article in which two systems are compared that operate on historical English and German data (17th–19th century). For German, the article describes a procedure for the generation of spelling variants to enhance information retrieval in historical databases (Ernst-Gerlach and Fuhr, 2006). The generation of variants was based on manually aligned pairs of tokens in their historic and contemporary spelling.[1] For English Pilz *et al.* (2008) describe the work on the so-called VARD tool that uses similarity metrics to detect historical spelling variants of modern tokens. These two approaches deal with younger data than ours and have in common that they use the present-day stage of a language as a base outset to deal with the historic variants.

3 The Data

The data we will work on is a digitized version of the literary part of the *Corpus-Gysseling*, a corpus containing all literary Middle Dutch texts that were preserved in manuscripts dated before 1300 AD and after 1200 AD. This corpus has been digitized and annotated by the Institute for Dutch lexicology in Leiden (Van der Voort van der Kleij, 2005).[2] The corpus comprises material from 27 texts, all greatly varying in topic, age, spelling, etc. In the tokenized version of this corpus we counted 573,063 running tokens; 40,471 of them are distinct. These have been annotated with 14,892 distinct lemma-tags. If not explicitly stated otherwise, all our experiments will be based on a 10-fold cross validation, whereby each time one tenth of the available training was 'held out' as a test set, while the other 90% of the data served as training material. Each tenth of the corpus would consist out of one tenth of every single text: fold 1 would contain the first tenth of every text, fold 2 every second tenth of every text and so on. For the sake of clarity and comparison, each experiment was run on the same ten folds.

Interestingly, the *Corpus-Gysseling* has been annotated within the framework of a large-scale diachronic language database (the 'Integrated Language Database', Van der Voort van der Kleij, 2005), spanning many centuries of the history of the Dutch language. It is laudable that the data for each stage in the development of the language has been consistently annotated with the same structures. All texts, for instance, have been uniformly enriched with the very same part-of-speech tag set, regardless of their age, dialect or topic. This is of course a welcome opportunity for scholars to undertake diachronic studies into the various data contained in the corpus. However commendable this annotation uniformity might be, it also has its drawbacks: the Middle Dutch texts, for instance, were annotated with present-day lemmas, which is perfectly comprehensible from the point of view of diachronic corpus studies. But as the data is of a considerably high age, Middle Dutch word tokens often have extremely little in common with their present-day lemma counterparts.

Downloaded from http://llc.oxfordjournals.org/ at UB Leipzig on January 17, 2012

These issues have important consequences for our methodology and possibly that of other diachronic corpus studies too, since we can hardly assume any formal relationship between the input token and the present-day lemma we would like to assign to it (something previous research into lemmatization sometimes heavily relied on). Consider the previous example: editing Afrikaans *geslaap* into SLAAP can be done through the simple edit script, 'delete initial *ge-*'. It is quite obvious, however, that translating the Middle Dutch form *ylicke* into the lemma GELIJK or *tpeert* into DAT+PAARD requires much more difficult edit scripts. This effect requires us to abandon the popular *script-as-label* method and attempt a *lemma-as-label* approach.[3] In what follows, we will not attempt to predict an edit script that allows a token to be edited into its lemma; rather we will try to treat the lemma as a stand-alone class label that is formally unrelated to the tokens associated with it.

4 Some Baselines

If we conceive of lemmatization as a classification task whereby tokens in running text should be assigned a class label—the lemma in our case—that bears no formal relationship to those tokens, the task of lemmatization becomes surprisingly similar to another well-known task, namely part-of-speech tagging. This task too is a sequential classification task, whereby tokens in context are assigned a tag, using information sources such as the focus token itself and its surrounding context (Daelemans and Van den Bosch, 2005, p. 86ff). In this view, we can conceive of lemmatization as part-of-speech tagging, the only difference being that the tagger will have to account for (many) more labels than in traditional PoS tagging applications.

To get a grasp of our data, we performed some exploratory experiments in which we used a traditional *part of speech* tagger as a *lemma* tagger. The tagger's task was to predict lemma tags, in the same way as it traditionally would be asked to predict PoS tags. For these experiments, we used MBT (Daelemans *et al.*, 1996), a part-of-speech tagger based on the *Tilburg memory-based learner* or TiMBL (Daelemans and Van den Bosch, 2005; Daelemans *et al.*, 2007).[4] For the sake of comparison, we used the software 'out-of-the-box' with its default options. As with part-of-speech tagging, it could be expected that the main difficulties would lie with the so-called *unknown* words: tokens in the test corpus that were not verbatim encountered during training and are as such difficult to analyze. We therefore also performed experiments with two well known string metrics: the Levenshtein edit distance and the Dice coefficient, both implemented in the latest version of the TiMBL-software (v6.2). The set up was simple: for each unknown word, both algorithms had to choose the lemma of the training token that was closest to it. No context was taken into account so that in the case of ties, the lemma of the most frequent token was chosen. The results of these initial experiments will be considered as baseline values in the rest of this article.

Figure 2 shows that the main problem indeed lies with the unknown tokens, which on average make up about 5% of the test material in each fold. For the known words category, the results are acceptable considering the little training data available. The final accuracy score after cross validation is just under 94%, a score that could undoubtedly be improved on through parameter tuning (and more training data). The unknown tokens seem to pose a larger problem. First of all, there is a rather low upper bound score or 'ceiling': in all experiments

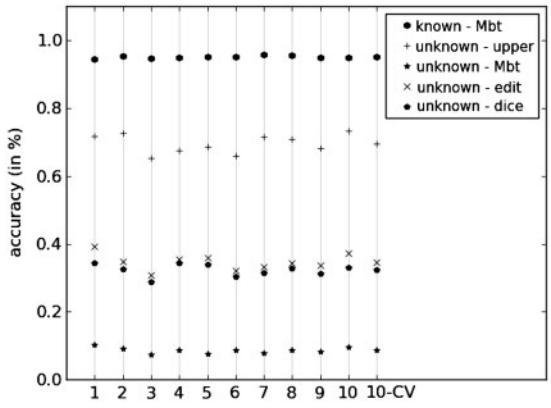

Fig. 2 Baseline accuracy results using the TiMBL and MBT software packages

Downloaded from http://llc.oxfordjournals.org/ at UB Leipzig on January 17, 2012

only about 70% of the unknown tokens has a lemma that could theoretically be predicted by the tagger, because the tagger encountered the lemma during training.[5] Subsequently, this ceiling value drops significantly in each fold. It is interesting to note that the classic Levenshtein distance does quite a good job and yields a rather high baseline: ±34%, meaning that this metric predicted nearly half of the words right that it could have gotten theoretically right. The Dice metric performed slightly worse, a remarkable result since recent research suggested that for modern languages, the Dice metric greatly outperforms the edit distance (Lyras *et al.*, 2009). For this particular Middle Dutch data, we were not able to reproduce this effect.

5 Learning Spelling Variation

Whatever procedure is developed in order to deal with HSVS, it should be able to account for the plausible variations that can occur between tokens within the same lemma category. Such intra-lemma variation will have to take into account inflectional and grammatical differences as well as spelling differences. In previous research, this knowledge was often 'handwritten', in the sense that a lemmatization algorithm would (at least initially) rely on some handcrafted rule-set (e.g. Souvay and Pierrel, 2009). In this section we describe a procedure that can automatically acquire this knowledge on the basis of a lemma-enriched text corpus.

Given some annotated training material, one is able to build a dictionary in which each entry is a lemma-label that is linked to the set of tokens that were associated with that lemma in the training data. For the lemma 'TONG' (English *tongue*) for instance, the set of tokens associated with it is listed in Fig. 3.

Subsequently, we are able to combine each token in that set with every other token in the set to form 'pairs' (cf. Pilz *et al.*, 2008). If the edit distance between the tokens of such a pair does not exceed 1 (i.e. at most one edit operation was needed to transform the first token into the second), we consider the pair to be a reliable indication of some kind of spelling variation in

TONG=[*tonghen*, *tonghe*, *tonge*, *tongen*, *tungen*, *tunge*]

Fig. 3 Representation of the 6 distinct tokens associated with the lemma TONG ('tongue') in the corpus

(**a**)	(**b**)	(**c**)
t – o – n – g – **h** – e – n	t – o – n – g – e – **n**	t – **o** – n – g – e
t – o – n – g – // – e – n	t – o – n – g – e – //	t – **u** – n – g – e

Fig. 4 Example of some Needleman-Wunsch alignments of pairs of tokens under the lemma TONG

Middle Dutch: e.g. *tonghen* : *tonghe*; *tonghen* : *tongen*; *tonge* : *tunge*.[6] We then align these two forms using the Needleman-Wunsch algorithm (Needleman and Wunsch, 1970), closely related to the Levenshtein algorithm. For the previous example, this process results in the three alignments displayed in Fig. 4.

We can then represent these alignments as a 'transliteration', a set of feature vectors representing how the first word could be transformed into the second. Each vector consists of a focus character, some left and right context characters, while the class label would constitute the transliteration character (group), as illustrated in Fig. 5.

The advantage of such a representation is that a machine learning algorithm can now be applied to the data, learning in which circumstances (=left and right character contexts) a given character (=focus character) can be replaced by another (group of) character(s) (=the class label). The learner will be able to build up a spelling model that is sensible to all sorts of mismatches between words, such as 'replacements' or 'insertions' in Levenshtein's (1966) original terms.

The above introduced software package, TIMBL, offers a fast implementation of the *k nearest neighbours* (henceforth *kNN*) algorithm. The *k nearest neighbours* algorithm is a lazy learning algorithm (Aha *et al.*, 1991): when training the learner on a particular data set, it simply stores a set of labelled examples in its memory, without performing any kind of 'abstraction, selection, or restructuring' on this data (Daelemans and Van den Bosch, 2005,

Downloaded from http://llc.oxfordjournals.org/ at UB Leipzig on January 17, 2012

(a)	(b)	(c)
From token 1 to token 2		
======**t**onghen:t =====**to**nghen=:o ====to**n**ghen==:n ===ton**g**hen===:g ==tong**h**en====:- =tongh**e**n=====:e tonghe**n**======:n	======**t**onge==:t =====to**n**ge===:o ====ton**g**e====:n ===tong**e**=====:g ==tong**e**======:en	======**t**onge==:t =====to**n**ge===:u ====ton**g**e====:n ===tong**e**=====:g ==tong**e**======:e
From token 2 to token 1		
======**t**ongen=:t =====to**n**gen==:o ====ton**g**en===:n ===tong**e**====:g ==tong**e**=====:he =tonge**n**======:n	======**t**ongen=:t =====to**n**gen==:o ====ton**g**en===:n ===tong**e**====:g ==tong**e**=====:e =tonge**n**======:-	======**t**unge==:t =====tu**n**ge===:o ====tun**g**e====:n ===tung**e**=====:g ==tung**e**======:e

Fig. 5 The representation in features vectors of the Needleman-Wunsch alignments for the tokens pairs introduced in Fig. 4

p. 26). When asked to classify a new, unseen instance, the memory-based system scans its memory for those instances that are most 'similar' to the new instance. The learner then places the unseen instance in the multidimensional space of training items in memory. Figuratively speaking, the algorithm 'draws a small circle' (the so-called 'hypersphere') around the test instance containing those training instances that are most similar to the test instance (Fig. 6). This (typically small) selection of highly similar instances is traditionally referred to as the set of *k nearest neighbours* for the test instance in question. Finally, the learner assigns a class label to the test instance by letting the test instance's k nearest neighbours perform a (e.g. majority) 'vote' on a class label for the unseen instance.

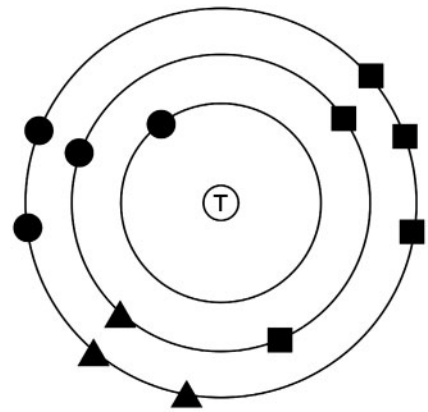

Fig. 6 Naive visualization of *kNN* classification. The circle in the middle represents the test instance (T). When assigned a position in the training space, the test instance has one nearest neighbour within its first hyperspere @k = 1. The *kNN* algorithm will assign the test instance the label of the black dots

6 Variant Generation: Principles

Now that we have a system that is able to acquire information about intra-lemma variation, the question remains how one can apply this knowledge. In what follows, we will present and compare two procedures for this. The first system we present turns to 'variant generation' to tackle the issue of lemmatizing unknown words. Following the work of Ernst-Gerlach and Fuhr (2006) and Van Halteren and Rem (2009) we can additively try to re-populate the instance space hoping that this data expansion will reduce the effects of data sparseness (i.e. the number of unknown words) and improve general classification results. The main problem involved in this procedure is finding a balance between the over-generation of instances (too noisy an instance space) and the under-generation of them (an instance space that is not representative enough).

Downloaded from http://llc.oxfordjournals.org/ at UB Leipzig on January 17, 2012

On the basis of the regularities observed in the training tokens, an algorithm suggests plausible new variants for the training tokens. As such, we hope for larger or more representative intersections between on the one hand, the (desired) instance areas in the training space and on the other hand the area assigned to unseen test input tokens. The idea of such a training data expansion is reminiscent of the Artificial Intelligence notion of 'Analysis through Synthesis'. What makes this technique interesting is that it puts more load and pressure on the training component, resulting in a simpler lookup procedure during the actual classification, namely lemmatization.

In order to generate new variants, we train a classifier on the character translation vectors presented in the previous section and subsequently apply ('test') the model on the very same data it was trained on. A crucial aspect in this stage is that we will force the learner into abstraction and generalization by increasing the *k* value. When applying the classifier on the same data it was trained on, it might tend to simply 'reproduce' the training data. It may tend to reproduce the focus character, in the sense that the predicted class label will be identical to the original focus character in the feature vector (since there are relatively more training instances in which the class is just equal to the focus character). In our case of 'variant generation', however, it only becomes interesting if we not simply consider the classifier's 'best pick' but also the 'second best pick', 'third best pick' and so on. If we need to generate a lot of plausible variants given a certain context, we need to consider many 'nearest neighbours' for each translation vector. With a *kNN* approach, we simulate this effect through varying/increasing the size of *k*.[7] In Fig. 7 we present the results of classifying the instances for the token *tonghen* at various *k* values.

Taking into account all nearest neighbours returned by the classifier, we can now construct a horizontal skeleton or matrix representing all possible variations for each character slot (Fig. 8). To generate new variants, we combine all options in all slots through the Cartesian product in the matrix by means of a permutation algorithm (Ernst-Gerlach and Fuhr, 2006).

Features	*Neighbours*		
	k = 1	*k = 2*	*k = 4*
= = = = = = t o n g h e n	{t}	{t, d}	{t, d}
= = = = = t o n g h e n =	{o}	{o}	{o, u}
= = = = t o n g h e n = =	{n}	{n}	{n}
= = = t o n g h e n = = =	{g}	{g}	{g}
= = t o n g h e n = = = =	{h , -}	{g, h}	{h, -}
= t o n g h e n = = = = =	{e}	{e}	{e, de}
t o n g h e n = = = = = =	{n, -}	{-, n, ne}	{r, -, n, ne}

Fig. 7 Representation of the nearest neighbours for each character in *tonghen*

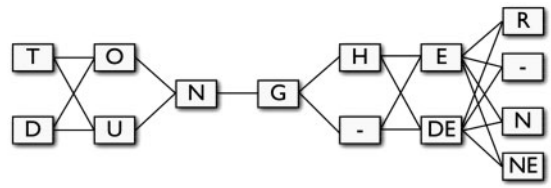

Fig. 8 A schematic representation of the possible combinations of the nearest neighbours represented in Fig. 7

The results for such an operation are presented in Fig. 9, with the resulting products at each value of *k*. The example used is the form *pecsuart* a token belonging to the lemma 'PEKZWART' ('pitch black', 'very black'). The token is unique in the corpus and furthermore spelled in a rather deviant way.

As can be expected, the larger the *k* value, the larger the number of highly plausible new variants. It is only at this increased level of abstraction, that the algorithm becomes sensible towards large-scale spelling variation phenomena. As such, the allograph *u* is replaced by its equivalent *w* in *pecswart* already at $k = 1$, while at $k = 3$ the vowel *a* gets replaced by its dialectal counterpart *e* in *pecswert*; both operations are furthermore combined in *pecswert*. A negative side-effect of creating these plausible new instances is the creation of less plausible or even nonsensical variants. These are mainly due to side effects that are difficult to remedy at this stage. The introduction of the final *–d*, for instance, instead of *–t* seems due to the influence of e.g. *swart* (SWORD). The apparition of final *–e* on the other hand, seems due to the influence of the frequent adjective *sware* (HEAVY). Nevertheless, the

Downloaded from http://llc.oxfordjournals.org/ at UB Leipzig on January 17, 2012

k-value	distinct products	generated variants	new highly plausible variants
1	2	*pecsuart*, pecswart	1
2	6	?pecsward, *pecsuart*, *pecsuare, pecswart ?pecsuard, *pecsware	1
3	23	?pecsward, ?pexsuart, pecsuert, ?pecsuard, ?pexswart, *pecsware, *pexswere, ?pecswerd, *pexsware, *pecsuare, ?pexsuert, ?pexswerd, *pecswere, ?pexsward, *pexsuare, pexsuere, ?pecsuerd, pecswart, *pecsuart,* ?pexsuard, ?pexsuerd, pexswert, *pecsuere, pecswert	4

Fig. 9 Result of the expansion procedure and subsequent permutation algorithm for the input word *pecsuart*. In the 'variant' row, the already existing variant is in italics; new variants, highly plausible to the expert eye, are underlined; new highly implausible variants are preceded by an asterisk. New variants whose plausibility is questionable, are preceded by a question mark

resulting 'bad' variants are always less plausible and as such do not introduce a (possibly confusing) overlap with the tokens of other lemmas. These are of course the results for a successful application. In the previously presented example of *tonghen* for instance, it is only at $k = 4$, that two new plausible variants are created, at the cost of creating 62 nonsensical variants.

7 Variant Generation: Experiments

We experimented with the concept of variant generation and evaluated the procedure in the following way: first we pre-process our training data in the manner presented in sections 5 and 6: we compile a lemma-tokens lexicon on the basis of the training data and align within each lemma the pairs of tokens that reliably seem to represent some kind of intra-lemma variation. Secondly, these alignments are presented as windowing vectors, representing the transformation needed to translate the tokens into each other. Subsequently, we train the memory-based learner on this data and apply it to its own training data, at various sizes of k. We consequently collect the variants suggested by this classification procedure, resulting in the automatic expansion of our available training material. Finally, we evaluate this procedure by extracting the unknown tokens from the test data and selecting their nearest neighbours in the *expanded* instance space on the basis of (1) an overlap metric (exact matches between the unknown tokens and the expanded set of training tokens) or (2) an edit distance (selecting the unknown tokens' closest neighbours in the expanded instance space according to the Levenshtein distance metric).

Table 1 and Fig. 10 present the results for these expansion experiments. In the left-column of Table 1 is the k value, indicating at which distances neighbours were allowed to take part in the generation of new variants. In the column 'training size', we list the new number of distinct tokens in the

Downloaded from http://llc.oxfordjournals.org/ at UB Leipzig on January 17, 2012

Table 1 Experimental results for the expansion procedure

k	Training size	Metrics				
		overlap			Levenshtein	
		Exacts	Among	Correct	Among	Correct
1	169.552	14.54%	11.24%	**08.67%**	46.40%	**30.58%**
2	309.552	26.20%	20.47%	**15.89%**	47.14%	**32.43%**
3	512.889	33.75%	26.26%	**19.96%**	47.42%	**32.84%**
4	772.631	38.60%	29.70%	**21.72%**	47.80%	**32.76%**
5	1.095.968	41.79%	31.97%	**22.75%**	48.12%	**32.56%**

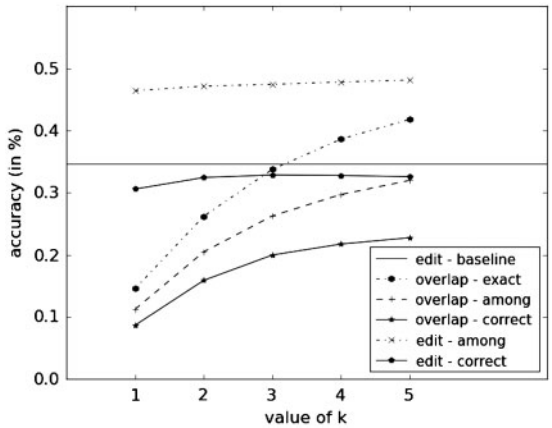

Fig. 10 Graph representing the results of the expansion procedure

training set after the expansion took place. Regarding the overlap metric, the 'exacts' column shows in how many cases a previously unknown word overlaps with a newly generated variant. The 'among' column, on the other hand, indicates how many of these unknown tokens with a new overlap, indeed overlapped with at least one token that was associated with the correct lemma. The 'correct' column indicates the final accuracy for the overlap score after a majority vote took place between the overlapping tokens. For the Levenshtein metric, the 'among' column indicates in how many cases at least one token with the correct lemma was among the unknown token's Levenshtein's closests. The Levenshtein 'correct' column displays the final accuracy score, in the case where tokens' Levenshtein's closests performed a majority vote on the lemma.

Figure 10 reveals interesting results. First of all, it is clear that the procedure is doing what it should do: increasing *k* leads to the generation of new variants, resulting in a steadily increasing overlap between the new variants and the (previously unknown) test tokens. However, the overlap generation is far from perfect: if an overlap is found for a test token, the overlapping set of training tokens will not always contain a token with the correct lemma. Moreover, the overlapping set of training tokens seems to contain a lot of 'false friends', since our final scores again significantly drop after a majority vote. With respect to Levenshtein, improvements can be made as well by increasing the *k* value. The results from this expansion approach, however, seem only interesting from a theoretical point of view and not from a practical one, since all final results ('correct' columns) are below the baselines we proposed: on the expanded corpus both overlap and edit distance perform worse than a plain Levenshtein distance procedure on the non-expanded corpus. From a theoretical point of view, the expansion algorithm proved interesting since it was actually able to significantly increase the overlap between the unknown tokens in the test set and the expanded training tokens, thus often successfully turning unknown tokens into known ones. From a practical point of view however, the expansion algorithm only yielded scanty results, since its final results were below that of the edit distance baseline we set out with.[8]

8 A Novel Voting Procedure

As presented in the previous sections, the Levenshtein distance so far proved the most efficient lemmatization algorithm for unknown words. Let us for a moment return to this approach: how does it work? To explain this we will use our model of *kNN* introduced in Section 5. With increasing Levenshtein distance, increasingly more nearest neighbours of a token will be considered. The problem is that the Levenshtein distance in its classic form is a 'blind' algorithm: it will consider

Downloaded from http://llc.oxfordjournals.org/ at UB Leipzig on January 17, 2012

the insertion of an 'x' equally plausible as the deletion of an 'e' and as such doesn't integrate any linguistic knowledge in its selection procedure. This procedure works fine for small hyperspheres: if nearest neighbours of an unknown token are found that only mildly differ from a particular set of training tokens (e.g. in the case of an edit distance equal to 1), the algorithm will have little problem predicting the right class label. It gets problematic when the unknown token deviates more from its nearest neighbours and does not have straightforward neighbours (e.g. an edit distance of 2). In those cases, the resulting hypersphere will be larger and can be expected to contain a lot of false friends. A plain, blind frequency vote will not suffice in these cases (Figs 11 and 12).

As previously mentioned, about 70% of the unknown tokens are tokens with a class label that the algorithm can theoretically predict. If we inspect the hyperspheres for the Levenshtein distance approach, we see that in roughly 60% of the cases at least one token with the right lemma is actually part of the set of nearest neighbours. Nevertheless, a plain majority voting procedure leads to a significant drop in accuracy and results in a correct classification of only 34% of the unknown tokens.

In what follows, we propose an alternative voting procedure that is able to finegrain this blind selection. In limiting ourselves to the Levenshtein-suggested hypersphere of tokens, our approach is comparable to the one proposed in Mitton (2009), as it does not focus on the actual selection of candidates but rather on *re-ranking* the already selected candidates. After the Levenshtein selection procedure, we have a set of training tokens associated with each test token. In 60% of the cases, this hypersphere actually contains a token with the desired class label. Therefore, this value becomes our new upperbound value.

The next step is to align the test token with every Levenshtein neighbour in its nearest neighbour hypersphere. These alignments can then be represented in the same way as we did above (Section 5). The example we will continue to work with is the unknown token *erkos* that should be assigned the lemma ERKIEZEN. The token apparently does not have any training neighbours at an edit

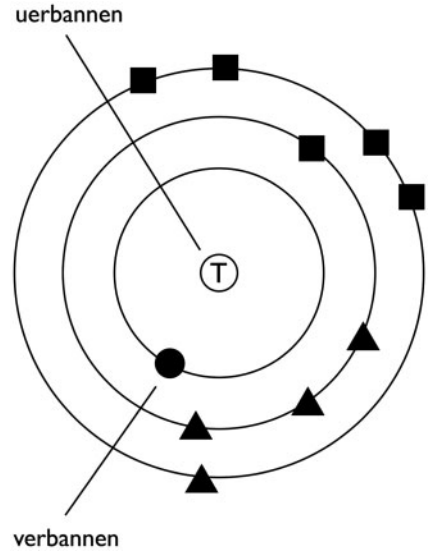

Fig. 11 Visualization of the best-case-scenario with the Levenshtein metric: the test instance (*uerbannen*) only mildly differs from a small set of items (ideally just one nearest neighbour, e.g. *verbannen*). The test token's immediate neighbourhood contains no 'false friends'

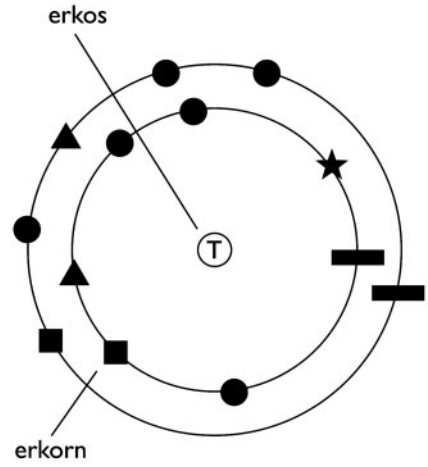

Fig. 12 Visualization of the worst-case-scenario with the Levenshtein-metric: the test instance (*erkos*) has no immediate neighbours in its surroundings. It only has a neighbouring hypersphere at an edit distance of 2: the token (*erkorn*) we would like to extrapolate the lemma (ERKIEZEN) from, is part of the closest hypersphere but this sphere contains a lot of 'false friends'. Consequently, a blind majority vote based on frequency does not lead to a correct classification

Downloaded from http://llc.oxfordjournals.org/ at UB Leipzig on January 17, 2012

erkos[ERKIEZEN]$_{neighbours}$ @k=1(edit distance of 2)
=
{2*werks[WERKEN=2], 1*eres[EREN=1], 1*argos[ARGOS=1], 2*verkoes[VERKIEZEN=2],
1***erkorn[ERKIEZEN=1]**, 4*erts[HERT=3, AARDS=1], 1*erlost[ERLOSSEN=1],
1*tros=[TROS=1], 10*elkes[ELK=10], 1*ers[HEER=1]}

Fig. 13 A detailed inspection of the nearest neighbours hypersphere (@k = 1) of the token *erkos*

distance of 1. Nearest neighbours only become visible at an edit distance of 2. The neighbourhood displayed in Fig. 13 is created, obviously containing a lot of false friends:

Without further interference, the voting algorithm picks the item *elkes* and extrapolates the lemma ELK, which was most frequently associated with the token. The test item we want to extrapolate the lemma from, is actually the less frequent *erkorn* with the desired lemma ERKIEZEN. Subsequently, each 'test token versus neighbour token' pair (e.g. *erkos* > *werks*, *erkos* > *eres*, *erkos* > *argos*) can be represented as a set of windowing, character-translation vectors, that represent the edit operations needed to transform the test token into the neighbour token. Importantly, the translation goes from test token to neighbour every time, so that each pair can be presented by the same number of vectors.

We should now be able to select the neighbour token that is most similar to the test token by assessing the 'likeliness' or 'soundness' of the character transformations needed to perform the translation. However, memory-based learning algorithms do not provide a straightforward way to answer the question: 'Given feature vector X, how probable is class C?' It is indeed typical of kNN that it makes use of the class distribution within a typically small set of nearest neighbours. All classes not occurring in this set therefore would have zero soundness. However, in a memory-based system, there is a possibility to better approximate the class 'soundness'. Instead of simply classifying the test instance (through inspecting the nearest neighbours), we can subsequently scan the instance space at steadily increasing values of k (e.g. 1–1000), until a training instance is encountered (@k = R) that was assigned label C. A confidence measure ('soundness') for the prediction of class C for feature Y, could then be approximated by the (dis)similarity score or distance in the space between the original instance X @k = 0 and the neighbours @k = R. The whole point of this 'instance space travelling' is not to find an absolute, trustworthy 'probability' score for a given prediction but rather to be able to compare two class labels. If a class C1 would be found at a larger distance from X than a class C2, C2 would be considered more 'sound' as a class for X than C1 (Fig. 14).[9]

This allows us to assign a soundness score to each of the translation vectors in each 'test token vs neighbour token' pair. A confidence measure for the whole set of translation vectors representing a word alignment can subsequently be approximated through the simple summation of the individual confidence measures (Fig. 15). This results in an indication of how likely it is that a particular test token is translated into a neighbouring token, based on the regularities observed through the alignment procedure in the training phase. The token of which the summed confidence values is lowest, can then serve as the new (and single) nearest neighbour of the instance item, from which it seems most sensible to perform class extrapolation. The results for applying this procedure are shown in Fig. 16. With this second procedure, the results are much more encouraging than with the first one. If we take the Levenshtein procedure with the majority to be our baseline (34.65%), we see that our new voting metric gains over 10% on this baseline, resulting in some 45.52% overall accuracy after cross validation.

9 Conclusions and Future Research

However 'solved' the problem might seem for modern, resource-rich Indo-European languages,

Downloaded from http://llc.oxfordjournals.org/ at UB Leipzig on January 17, 2012

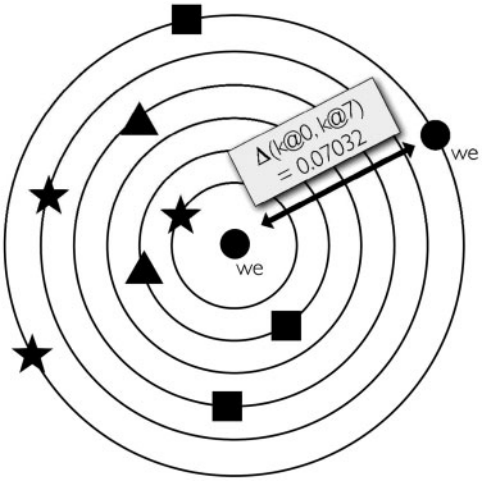

Fig. 14 Representation of our approximation of class soundness in a *kNN* model. If an instance is encountered with the class 'we', we search the instance space at steadily increasing values of *k*, until an instance was encountered with the class label 'we'. The soundness of the class label is approximated by taking the dissimilarity score (in this example 0.07032) between the test instance (@k = 0) and the first training instance encountered with the same class as the test instance (in this example @k = 7)

erkos > *werks*:

1. = = = = = = e r k o s = = we	>	0.0703200055737
2. = = = = = e r k o s = = = r	>	0.0203058792382
3. = = = = e r k o s = = = = k	>	0.0491802235479
4. = = = e r k o s = = = = = -	>	0.128389885419
5. = = e r k o s = = = = = = s	>	0.0264659421807
	+	
		0.29466193596

erkos > *erkorn*

1. = = = = = = e r k o s = = e	>	0.0156921031382
2. = = = = = e r k o s = = = r	>	0.0203058792382
3. = = = = e r k o s = = = = k	>	0.0491802235479
4. = = = e r k o s = = = = = o	>	0.0518756110036
5. = = e r k o s = = = = = = rn	>	0.129232980606
	+	
		0.266286797534

erkos > *erts*

1. = = = = = = e r k o s = = e	>	0.0156921031382
2. = = = = = e r k o s = = = r	>	0.0203058792382
3. = = = = e r k o s = = = = -	>	0.147345574934
4. = = = e r k o s = = = = = t	>	0.175827985279
5. = = e r k o s = = = = = = s	>	0.0264659421807
	+	
		0.38563748477

Fig. 15 Some classification results for the case of *erkos*, representing the soundness of the character translation. Three examples are given, namely for the pairs *erkos* > *werks*; *erkos* > *erkorn*; *erkos* > *erts*. The summed soundness of each of these pairs is taken to be the sum of the individual translation. The most likely token pair is considered to be the pair with the lowest sum (in this case *erkorn*)

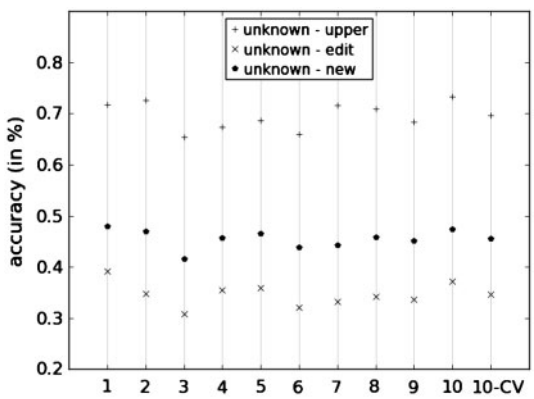

Fig. 16 Results for the second lemmatization procedure, replacing a majority vote on the candidates suggested by the Levenshtein metric by a novel procedure. This procedure yields an average result of 45.52% after cross-validation, which is a significant increase (>10%) on our baseline of 34.65%

lemmatization remains a difficult issue for resource-scarce languages with a lot of spelling variation. The situation for historical languages such as Middle Dutch is undoubtedly exemplary for other, less-studied modern languages that lack standard spelling (e.g. many vernacular African languages). This contribution might also prove relevant to other corpus linguists dealing with diachronic language resources.

In this article we have dealt with a number of issues in the lemmatization of Middle Dutch literary texts. First, we described a language-independent system to automatically acquire knowledge about (intra-lemma) spelling variation on the basis of an annotated corpus. Contrary to many earlier approaches to HSVS, this system does not rely on any prior handcrafted rules. Secondly, we explored two (memory-based) procedures to apply this system to lemmatization. The first system automatically expanded the available training data by generating new spelling variants. From a theoretical

Downloaded from http://llc.oxfordjournals.org/ at UB Leipzig on January 17, 2012

perspective, this approach yielded interesting results, since this procedure actually succeeded in generating new plausible variants, resulting in an increased overlap between training and test data. From a practical point of view, the results were less encouraging, since the procedure was not able to improve on a classic Levenshtein-based procedure for the actual lemmatization task. The second procedure implemented a new voting procedure for the original Levenshtein-suggested candidates. We described a system that is able to judge on the 'soundness' of token pairs and rerank candidates on the basis of this 'soundness score'. In our experiments, the latter system largely outperformed the former one but this does not necessarily imply that the expansion approach should be completely discarded, as there are still plenty of options for further tuning, possibly with other learners. In our view, the second procedure, however, has the benefit of simplicity. The first procedure observes the regularities in the training, subsequently turns to the intermediate step of expanding the available material and only then can it turn to the actual classification. The second procedure skips the expansion phase and simply uses the observed regularities in classification—something the medieval logician Occam would have approved of.

Acknowledgements

Mike Kestemont and Guy De Pauw are researchers with the Research Foundation—Flanders (FWO) and gratefully acknowledge the Foundation's support. All authors would like to thank Frank Willaert for his helpful comments on previous versions of this contribution and Bram Tack for his technical assistance.

References

Aha, D.W., Kibler, D., and Albert, M. (1991). Instance-based learning algorithms. *Machine Learning*, **6**: 37–66.

Chrupała, G. (2006). Simple data-driven context-sensitive lemmatization. *Procesamiento del Lenguaje Natural*, **37**: 121–7.

Crystal, D. (1997). *A Dictionary of Linguistics and Phonetics*. Oxford: Oxford University Press.

Daelemans, W. and van den Bosch, A. (2005). *Memory-Based Language Processing*. Oxford: Oxford University Press.

Daelemans, W., Groenewald, H.J., and Van Huyssteen, G.B. (2009). Prototype-based active learning for lemmatization. In Angelova, G., Bontcheva, K., Mitkov, R., and Nicolov, N. (eds), *Proceedings of the International Conference Recent Advances in Natural Language Processing (RANLP 2009)*. Borovets: Association for Computational Linguistics, pp. 65–70.

Daelemans, W., Zavrel, J., Berck, P., and Gillis, S. (1996). MBT: a memory-based part of speech tagger generator. In Ejerhed, E. and Dagan, I. (eds), *Proceedings of the Fourth Workshop on Very Large Corpora*. Copenhagen: Association for Computational Linguistics, pp. 14–27.

Daelemans, W., Zavrel, J., van der Sloot, K., and van den Bosch, A. (2007). *Timbl: Tilburg Memory Based Learner, Version 6.1, Reference Guide.* Technical Report ILK Research Group Technical Report Series No. 07-07, ILK Research Group, University of Tilburg.

De Pauw, G. and de Schryver, G.-M. (2008). Improving the computational morphological analysis of a Swahili corpus for lexicographic purposes. *Lexikos*, **18**: 303–18.

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

Erjavec, T. and Džeroski, S. (2004). Machine learning of morphosyntactic structure: Lemmatising unknown Slovene words. *Applied Artificial Intelligence*, **18**: 17–40.

Ernst-Gerlach, A. and Fuhr, N. (2006). Generating search term variants for text collections with historic spellings. In Lalmas, M. *et al.* (eds), *Proceedings of 28th European Conference on Information Retrieval Research (ECIR 2006)* Lecture Notes in Computer Sciences, **Vol. 3936**, London: Springer, pp. 49–60.

Francis, N. and Kucera, H. (1982). *Frequency Analysis of English Usage: Lexicon and Grammar*. Boston, MA: Houghton Mifflin.

Giusti, R., Candido, A. Jr, Muniz, M., Cucatto, L., and Aluísio, S. (2007). Automatic detection of spelling variation in historical Corpus: an application to build a Brazilian Portuguese spelling variants dictionary. In Davies, M., Rayson, P., Hunston, S., and Danielsson, P. (eds), *Proceedings of the Corpus Linguistics Conference*

Downloaded from http://llc.oxfordjournals.org/ at UB Leipzig on January 17, 2012

(CL 2007). Birmingham: Centre for Corpus Research (University of Birmingham).

Groenewald, H.J. (2009). Using technology transfer to advance automatic lemmatization for Setswana. In De Pauw, G., Groenewald, H., and De Schryver, G.-M. (eds), *Proceedings of the EACL Workshop on Language Technologies for African Languages (AfLaT 2009)*. Athens: European Language Resources Association (ELRA), pp. 32–37.

Halácsy, P. (2006). Benefits of deep NLP-based lemmatization for information retrieval. In Peters, C. *et al.* (eds), *Working Notes for the CLEF 2006 Workshop* Lecture Notes in Computer Science, **Vol. 4730**, Alicante: Springer.

Kempken, S., Luther, W., and Piltz, T. (2006). Comparison of distance measures for historical spelling variants. In Bramer, M. (ed.), *Artificial Intelligence in Theory and Practice*. Boston, MA: International Federation for Information Processing, pp. 295–304.

Kestemont, M. and Van Dalen-Oskam, K. (2009). Predicting the past: memory-based copyist and author discrimination in medieval epics. In Calders, T., Tuyls, K., and Pechenizkiy, M. (eds), *Proceedings of the Twenty-first Benelux Conference on Artificial Intelligence (BNAIC 2009)*. Eindhoven: Benelux Association for Artificial Intelligence, pp. 121–8.

Knowles, G. and Mohd Don, Z. (2004). The notion of a ''lemma''. Headwords, roots and lexical sets. *International Journal of Corpus Linguistics*, **9**: 69–81.

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

Lyras, D.P., Sgarbas, K.N., and Fakotakis, N.D. (2008). Applying similarity measures for automatic lemmatization: a case-study for modern Greek and English. *International Journal on Artificial Intelligence Tools*, **17**: 1043–64.

Mitton, R. (2009). Ordering the suggestions of a spell-checker without using context. *Natural Language Engineering*, **15**: 173–92.

Needleman, S.B. and Wunsch, C.D. (1970). A general method applicable to the search for similarities in the amino acid sequence of two proteins. *Journal of Molecular Biology*, **48**: 443–53.

Pilz, T., Ernst-Gerlach, A., and Kempken, S. (2008). The identification of spelling variants in English and German historical texts: manual or automatic? *Literary and Linguistic Computing*, **23**: 65–72.

Porter, M.F. (1980). An algorithm for suffix stripping. *Program*, **14**: 130–7.

Reynaert, M.W.C. (2005). *Text-Induced Spelling Correction*. Ph.D. thesis, University of Tilburg.

Souvay, G. and Pierrel, J.-M. (2009). LGeRM. Lemmatisation des Mots en Moyen Français. *Traitement Automatique des Langues*, **50**: 149–72.

Toutanova, K. and Cherry, C. (2009). A global model for joint lemmatization and part-of-speech tagging. In Association for Computational Linguistics (eds), *Proceedings of the 47th Annual Meeting of the Association for Computational Linguistics*. Singapore: Association for Computational Linguistics, pp. 486–94.

Van Dalen-Oskam, K. and van Zundert, J. (2007). Delta for Middle Dutch – author and copyist distinction in *Walewein*. *Literary and Linguistic Computing*, **22**: 345–62.

van der Voort van der Kleij, J. (2005). Reverse lemmatization of the dictionary of middle Dutch (1885-1929) using pattern matching. In Kiefer, F., Kiss, G., and Pajzs, J. (eds), *Papers in Computational Lexicography*. Budapest: Hungarian Academy of Sciences, pp. 203–10.

van Halteren, H. and Rem, M. (2009). A tagger-lemmatiser for 14th century Dutch charters. *Unpublished Paper Presented at Computational Linguistics in the Netherlands* (CLIN 2009, Groningen, 22/01/2009).

Notes

1 A similar system for Middle Dutch has been presented by Van Halteren and Rem (2009). Their approach was not to generate variants for a given input term, but rather to expand the whole available corpus with variants.

2 The corpus is not publicly available yet—due to intellectual property issues—but this might change in the very near future. Note that this corpus contains a non-literary part (with charters) as well. We have not studied this part of the corpus, since our lemmatizer will function in the context of research project into literary texts.

3 Another problem is that the corpus contains a considerable deal of clitic tokens that need a *combination* of lemmata as label (e.g. *tpeert* that needs a combination of the lemma tags DAT and PAARD). The problem with these combinatory tags is that the corpus displays enormous annotation ambiguities for this category: the annotation of the clitic tokens does not reveal the order of the lemmas in the original token (*tpeert* = DAT + PAARD = PAARD + DAT).

Downloaded from http://llc.oxfordjournals.org/ at UB Leipzig on January 17, 2012

4 Software and documentation are available from http://ilk.uvt.nl/.
5 Note that this ceiling is an artefact of our so-called lemma-as-label approach: if we treat class labels as being formally unrelated to their tokens, we cannot deal with unknown class labels. A classifier cannot predict a lemma in the test set, if it has not seen it during training.
6 Note that we also aligned tokens 'with each other' (i.e. the edit distance ≤ 1), in order not to throw out any information in our training data. A hapax, for instance, would not lead to any pairs in the case and would be thrown out if we would only accept an edit distance between token pairs equal to 1.
7 It should be noted that Timbl does not implement *kNN* in the traditional way: the value of k does not denote how many actual nearest neighbours can participate in the class voting. Rather the value indicates to what distance from the test instance neighbours are allowed to take part in the voting. As such, the algorithm actually implements a 'k nearest distances' algorithm.
8 It is important to point out that the expansion procedure could arguably be improved upon by applying a classic Viterbi algorithm to the 'path combinations' represented in Fig. 8. As such, we could assess the probability of a given combination of options by assigning probability scores to each option (or rather 'arch') in the skeleton. As such, we should be able to perform an exemplar weighting that might have a positive influence on the classification results. However, in our case it would not make much sense to implement this: probability or not, even the 'overlap—among' results stay well under the edit distance baseline. Again, from a theoretical point of view it might be interesting to see whether such a Viterbi voting would indeed improve classification. From a practical point of view however, it is impossible that this procedure would improve much over the baseline.
9 One problem is of course smoothing, in the case of a class label that has not been observed during training. In our case, we would scan until k@1000. If the desired label is not observed at that distance, we take the distance @k = 1000.

Downloaded from http://llc.oxfordjournals.org/ at UB Leipzig on January 17, 2012