Learning Compound Boundaries for Afrikaans Spelling Checking

Gerhard B. van Huyssteen* & Menno M. van Zaanen[#]
North-West University (South Africa) & University of Tilburg (The Netherlands)
* Centre for Text Technology, North-West University, Potchefstroom, 2531, South Africa
Tel: +27 18 299 1488; Fax: +27 18 299 1562
E-mail: afngbvh@puk.ac.za
[#] ILK, Tilburg University, Tilburg, 5000LE, The Netherlands;
Tel: +31 13 466 8260; Fax +31 13 466 3110
E-mail: mvzaanen@uvt.nl

Abstract
Current spelling checkers for Afrikaans still do not provide full access to desired linguistic performance, especially with respect to high lexical recall and error precision. One of the main problems is that Afrikaans is an agglutinative language with a high lexical generative power using concatenative compound formation. This means that the lexicon in an Afrikaans spelling checker can never account for all possible compounds, and other means should therefore be sought to recognise valid compounds. In this article, we investigate two approaches to finding compound boundaries. First, we describe a longest string-matching algorithm, which searches for known words at the beginning and end of the compound. Next, a machine-learning approach using decision trees is implemented. Results of both approaches are presented, indicating that the longest string-matching algorithm outperforms the machine-learning approach. However, the machine-learning approach has many advantages over the longest string-matching algorithm. The article concludes with a discussion of the advantages and disadvantages of the systems, remaining problems and possible solutions.

1. Introduction

Existing commercially available spelling checkers for Afrikaans (i.e. PUK Afrikaanse Speltoetser; Pharos Speller (and hyphenator); Ispell vir Afrikaans) still do not comply to all the desiderata for spelling checkers, specifically with regard to linguistic performance (i.e. high lexical recall, high error precision, and adequate suggestions [15]). The project under discussion is aimed at the development of an enhanced spelling checker for Afrikaans, to improve in functionality and performance over existing spelling checkers for Afrikaans. The project commenced in February 2002, and the new spelling checker was released in February 2004 [18].

The theme of this paper is the problem of handling compounds in this new spelling checker. Afrikaans, as an agglutinative language, allows for productive concatenative compound formation. However, existing spelling checkers for Afrikaans are unable to cope effectively with the extreme flexibility resulting from compound formation. Since the architectures of all existing spelling checkers are based on a basic lexicon look-up method, only compounds that are included in the lexicon will be recognised by these spelling checkers (e.g. frequently used compounds like *hondehok* 'dog house' or *strandhanddoek* 'beach towel'). Therefore, novel or less often used compounds (e.g. like *hondehokdakverf* 'dog house roof paint' or *strandhanddoekstoffabriek* 'beach towel cloth factory') cannot be recognised by these spelling checkers, which results in a lower overall lexical recall score.

To improve on the lexical recall of existing spelling checkers, we have decided (in addition to other improvements) to introduce compound analysis as part of the new spelling checker [17]. The idea is that if we can detect word boundaries within a compound, we will be able to resolve the analysis of that compound largely (i.e. find the components in the compound, and if they match with strings in the lexicon, the compound is considered valid). The challenge is therefore to find word boundaries automatically, as well as valence morphemes (such as *-s-* in *fakulteit**s**raad* 'faculty council', *-e-* in *hoed**e**doos* 'hatbox', or *-ens-* in *behoud**ens**gesind* 'conservative'), in order to determine whether a given (compound) word is a valid word.

For spelling checkers, finding compound and valence-morpheme boundaries in strings is only part of the actual problem to handle words that are not in the spelling checker's lexicon, but which are not necessarily wrong. For instance, when searching for compound boundaries it should be taken into account that unknown words are not necessarily compounds (e.g. the inflected word *brander+tjie* 'wave+DIM' should not be analysed as *brand+ertjie* 'burn+pea'). As an additional problem, not only valid words, but also invalid ones are analysed by the compound analyser; the compound analyser should therefore be able to handle invalid compounds as well as valid compounds, asking for a degree of robustness. Of course, finding word boundaries in invalid words can be extremely difficult, especially when an error occurs near a word boundary. In this article, we will only treat the already difficult problem of finding word boundaries in valid compounds and, for now, assume that the same approach can be used to handle invalid compounds. Evaluating this is considered future work.

In this paper, we will commence by giving a concise overview of compound formation in Afrikaans, explaining why compound analysis is essential for Afrikaans proofing tools. We will then discuss a longest string-matching algorithm in detail, illustrating how and why it fails to yield desirable analyses. The machine-learning technique we currently employ will be discussed next, and we will present results of both systems. We then discuss the advantages and disadvantages of both approaches. To conclude, we will discuss some remaining problems, possible solutions, and future work.

2. Compounding in Afrikaans

Compounding is a very productive word-formation process in Afrikaans. Like in other Germanic languages (e.g. German and Dutch), it usually involves the concatenation of two or more components (prototypically independent word forms like *kat* 'cat' and *kos* 'food'), to form a single orthographic word (like *katkos* 'cat food'). For purposes of this paper (and hence for the computational analysis/decomposition of compounds in Afrikaans), we need to give a *structural* description of the types of compounds, the components of compounds, and the way these components combine in Afrikaans. Inasmuch as we are interested in the structural (i.e. formal) attributes of compounds, no attention will be given to their semantic characterisation.

In terms of morphological structure and compositional features, we distinguish mainly four kinds of compounds in Afrikaans, viz.

- ? primary compounds (and hence inflected and derived compounds);
- ? neo-classical compounds;
- ? synthetic compounds; and
- ? "compounding compounds".

Primary compounds (also called "root compounds" [13]) are compounds formed prototypically from concatenated words, as in the above-mentioned *katkos* ('cat food'). However, the components of primary compounds need not only be words (i.e. independent morphemes or stems): dependent morphemes (i.e. roots) can also occur in compounds as components, although not as frequently as independent morphemes (compare for instance *leg+kaart* 'lay+card=puzzle', where *leg-* is a dependent morpheme). Likewise, in the case of neo-classical compounds, two dependent morphemes (i.e. neo-classical roots) combine with each other to form an independent orthographic word: *bio+logie* 'bio+logy=biology'.

Synthetic compounds are formed by means of affixation based on word groups or syntactic constructions, and is in Afrikaans not necessarily verbally based [3]. Compare for instance *vyfweekliks* 'five weekly', where *vyf* 'five' is a quantifier, *week* 'week' a noun, and *–liks* an adjectivaliser. Related to synthetic compounds are so-called "compounding compounds", which are compounds consisting of at least three words (mostly adjective + noun + noun), and where the first two words form a word group. For example, *mediesefondsbydrae* 'medical aid contribution' is analysed as *(mediese fonds)+bydrae* '(medical aid)+contribution'.

With regard to the number of stems/roots that can combine in compounds, it is theoretically possible that an infinite number can combine in a single orthographic word. The longest word that we have found so far in corpus data is a 53-character word, consisting of nine dependent and independent components, viz. *radio-telefoonnoodfrekwensieluisterdiensontvangtoestel* ('radio telephone emergency frequency listening service reception device'). However, from our corpus of 40,051 Afrikaans compounds (which was also used as part of our training data for the machine-learning algorithm – see 4.1 below), it is clear that compounds with only two stems/roots occur much more frequently than compounds with more than two stems/roots (see Table 1). In [10], comparable distribution patterns for German and Dutch are presented based on much smaller data sets.

Nr of Components	Distribution
2	31358 (78.30%)
3	7993 (19.96%)
4	663 (1.66%)
5	35 (0.09%)
6	2 (0.005%)

Table 1: Distribution of number of stems/roots occurring in Afrikaans compounds (n=40,051)

Like other words in Afrikaans, compounds can also undergo processes of inflection and derivation, for instance *kisfabriek**e*** 'coffin factories', where the plural *–e* is added to the compound, or *katkos**agtig*** 'like cat food', where *–agtig* is a derivational adjectivaliser. Although some authors claim that it is not only the right-hand component (the head of the compound) that shows morphology, but also the other components (e.g. *kat**jie**kos* 'kitten food') [6], we are of opinion that such an analysis is incorrect. We should rather analyse an instance like *katjiekos* as *kat+jie* 'cat+DIM=kitten', which then subsequently combines with *kos* 'food'. The morphological process is therefore not compounding internal. A notable exception is low-frequency left-headed compounds, like *prokureur-generaal* 'attorney-general', where the plural is indicated on the left-hand component: *prokureur**s**-generaal* 'attorneys-general'.

Components in compounds can be from all Part-of-Speech categories in Afrikaans, but notably from nouns, verbs, adjectives, and adverbials [4]. Although components from other Part-of-Speech categories do occur in compounds, they are either highly entrenched (e.g. *hierdie* 'this' (demonstrative pronoun), which can be analysed as *hier* 'here' (adverb) + *die* 'the' (article)), or are limited in their productivity (e.g. the pronoun *hulle* 'they' can combine with almost any person name to indicate collectiveness, as in *Pa-hulle* 'Father-they', *Peter-hulle* 'Peter-they', etc.). Also, some words never occur in compounds (e.g. *baie* 'many', *hy* 'he', or *is* 'is'); although this is in principle a finite list of words, it is very difficult to determine this list precisely (e.g. adverbs should not be included in this list, while some adverbs, like *baie* 'many', should be included).

Except for stems, roots, and inflected or derived words that can be components in Afrikaans compounds, another component deserves our attention, viz. the valence morpheme (also called, *inter alia*, interfix [1], or link phoneme [2]). Although much research has been done on the valence morpheme in the Germanic languages [8, 9, 12, 16], including in Afrikaans [4, 6], the systematics of this morpheme is still eluding. For instance, [8] illustrates that only 32% of the linking elements in the compounds found in the Dutch CELEX database can be systematically explained by a set of tendencies; it is expected that the same results could be found in Afrikaans.
In our corpus of 40,051 compounds, only 7,270 compounds contain one of the valence elements found in Afrikaans compounds (including hyphens – see below). Compare the results in Table 2.

Valence morpheme	Distribution
-s- (*verbinding**s**klank*)	5861 (80.62%)
-e- (*hond**e**hok*)	508 (6.99%)
-ns- (*lewe**ns**drang*)	101 (1.39%)
-er- (*kind**er**skoen*)	90 (1.24%)
-ens- (*nooi**ens**van*)	52 (0.72%)
-n- (*buite**n**gewone*)	2 (0.03%)
-der- (*meer**der**man*)	0 (0.00%)
Hyphen (*sterre-energie*)	656 (9.02%)

Table 2: Distribution of valence elements in Afrikaans compounds (n=7,270)

It should also be noted that hyphens could also occur as parts/components of compounds. In the case of some copulative compounds (e.g. *skilder-skrywer* 'painter-writer'), reduplications (e.g. *speel-speel* 'play-play'), and some left-headed compounds (e.g. *prokureur-generaal* 'attorney-general'), the hyphen is compulsory. For the sake of legibility, hyphens could also be used, notably in the cases of vowel accumulation (e.g. *koei-oë* 'cow's eyes'), and very long words (e.g. *dieselenjin-wipbakvragmotor* 'diesel engine tipper lorry' [14]. For purposes of morphological analysis (specifically in the context of spelling checking), this feature of Afrikaans compounds should also be taken into account.

3. Computational Approaches

In order to find a way to analyse compounds in Afrikaans effectively, we have to take the above background on compounding and compound structures in Afrikaans into account when designing an algorithm. We will first discuss a longest string-matching algorithm, indicating its strong and weak points. We will then explain a second approach, using machine learning, as an alternative to the longest string-matching algorithm.

3.1. Longest String-Matching Algorithm

We started our quest for an effective way to identify word boundaries in compounds by implementing a module that uses a longest string-matching (LSM) algorithm (for a comparable approach, see [10]). The basic idea is to search for the longest parts of the word (from the left and the right) that is an element of the spelling checker lexicon. On a given string, a longest string-matching is first performed from the right hand side of the word (i.e. looking for a matching suffix), followed by a longest string matching from the left of the remaining part of the word (i.e. the prefix part). In a compound like *rekenaartoerusting* 'computer equipment', the string *toerusting* 'equipment' is found firstly, followed by the string *rekenaar* 'computer'. Note that simple, non-compound, words can also be handled using this approach, since the suffix search already finds the entire word in the spelling checker lexicon.
In order to handle the valence elements (Table 2 above), as well as words consisting of more than two components (Table 1 above), we specify that, after the suffix and prefix parts have been removed from the original string, the remains could be either:

- ? an element from the list of valence elements; or
- ? an element in the spelling checker lexicon.

In a string like *besigheidsman* 'business man', the strings *besigheid* and *man* is found, leaving the valence *-s-* as only element. Since the *-s-* is in the list of valid valence elements, the string *besigheidsman* is correctly analysed as a compound. Likewise, the compound *katkosbak* 'cat food bowl' is decomposed into the parts *kat+kos+bak*, where *kos* is left as residual, and subsequently found in the spelling checker lexicon.
Lastly, in addition to the two-way search and the lexicon look-up of the residual, it is also specified that certain components (like articles, certain pronouns, etc.) are not allowed to be a part of the compound. These components (circa 300 words) are contained in a separate list. This effectively marks typos such as **dieman* 'theman' or **baielekker* 'verynice' as invalid.
Testing the implementation of the longest string-matching algorithm in the context of the new spelling checker proves that it often does not yield acceptable analyses. Firstly, various invalid words are analysed as valid; compare for instance the erroneous analysis of the misspelled word **weerspieel* (instead of *weerspieël* 'reflect') as *weer+spie+el* 'weather+cotter+ell', or **Peterhulle* (instead of *Peter-hulle* 'Peter-they') as *Peter+hulle.*

Secondly, the algorithm is unable to handle compounds with more than three components effectively: a valid compound like *langstepassingstringsoektog* ('longest match string search'), will be marked as invalid, because the residual *-passingstring-* 'match string' will not be found in the lexicon.
Moreover, the biggest problem of the LSM algorithm is that it depends, **for its analysis**, heavily on the quality and completeness of the lexicon (i.e. not only to validate a string, but even in the analysis phase). When parts of the compound cannot be found in the lexicon, a compound cannot be analysed into different components. Since it is possible to create compounds containing, for example, proper names (e.g. *Mandelarylaan* 'Mandela Avenue'), even a very comprehensive and large lexicon will not solve the problem, because it is almost impossible to include all possible proper names in a lexicon. The process of finding correct compound boundaries using LSM becomes even harder with compounds that are more complex.

3.2. Machine Learning

To resolve the problems of the LSM method, we investigate a machine-learning approach to the problem of finding compound boundaries. The advantages are that the system does not depend on the lexicon, no expensive searching and aligning is needed, valence morphemes are handled in a straightforward manner, and compounds that are more complex are treated similarly to simple compounds.
Machine-learning techniques generally build classifiers that are trained using annotated data. Several decisions need to be made: a specific machine-learning technique needs to be selected, and an encoding of the compound boundary information should be created.
In this article, we investigate how well decision trees [11] are able to find word and valence-morpheme boundaries within words. Of course, other machine-learning techniques can be used, but we merely would like to test the effectiveness and problems of machine learning in this context. Specific advantages and disadvantages of this approach will be discussed in Section 5.
A decision tree is a tree structure where each node represents a feature and the edges coming out of the node are linked with possible values of the feature. Leaf nodes have class labels attached to them. In each node, some additional information is stored, such as a confidence value.
Machine learning generally consists of two distinct phases, training and classification. During the training phase, the classifier, in our case a decision tree, is built using annotated data. During the classification phase, the classifier is used to annotate new, unseen data.
In the training phase, a decision tree is created from the root down. For each feature, the distribution of the training data is analysed and the average entropy of each feature is computed. The one that has the lowest degree of entropy is selected and used as the current node in the tree. This process is repeated for each of the parts in the partition of the training data based on the selected feature. Once all training data belonging to a certain node have the same class, that node becomes a leaf node with that specific class attached to it. For more information on the decision tree generation procedure, see [11].
Classification of a feature vector is done by traversing the tree from the root to a leaf node. Since each node is attributed with a feature, the value of that feature (in the input vector) is used to decide which way to descent into the tree. When a leaf node is reached, the result is known.
In the problem discussed in this article, the annotated data consists of compounds with marked word and valence-morpheme boundaries. This information is not directly usable; machine-learning algorithms generally require fixed length feature vectors as input, so a specific mapping has to be designed.
In this research, the aim is to build a classifier that for each position in a word, decides whether a word boundary or valence-morpheme boundary is present. The classifier only uses a limited context around the position to make the decision, i.e. only a certain number of characters surrounding the particular position in the word. Effectively, this means that a sliding window of a predefined size is used for each position in the word.

4. Results

To evaluate the LSM and machine-learning methods, we have applied these systems to a list of compounds. The compounds are annotated with compound boundaries and valence-morpheme boundaries, for example, using + and _ characters respectively. Examples of annotated words are then: *redding _ s + vlot* 'rescue raft', *registreer + apparaat* 'registration device' and *e- + pos + wag + woord* 'email password'. Note the use of the hyphen in *e-pos* 'email'. The hyphen is considered a regular character, so the only possible answers the classifier will provide are the plus (for compound boundaries) and the underscore (for valence-morphemes boundaries).
The LSM method is evaluated by removing all compound boundaries and valence-morpheme information (i.e. taking the plain compounds from the structured words). These words are input to the LSM module, which again inserts compound and valence-morpheme information. The inserted boundaries are compared against the original boundaries.
To train and test the machine-learning method, all compounds are converted into feature vectors. For example, the word: *eksamen + lokaal* 'exam room' is converted into the feature vectors depicted in Table 3 (using context three to the left and the right of the position).

Note that the equals sign denotes a missing value. The first six values denote the context and the final value indicates the class. Only between the *-men-* and *-lok-* there should be a +; nothing should be inserted between the rest of the characters in the word.

Left context			Right Context			Class
=	=	=	e	k	s	=
=	=	e	k	s	a	=
=	e	k	s	a	m	=
e	k	s	a	m	e	=
k	s	a	m	e	n	=
s	a	m	e	n	l	=
a	m	e	n	l	o	=
m	e	n	l	o	k	+
e	n	l	o	k	a	=
n	l	o	k	a	a	=
l	o	k	a	a	l	=
o	k	a	a	l	=	=
k	a	a	l	=	=	=
a	a	l	=	=	=	=

Table 3: Feature vectors for *eksamenlokaal* 'exam room'

Once trained, the classifier takes feature vectors without the class information as input and returns a plus, underscore, or equals character denoting the corresponding class indicating boundary information. Using this information, the feature vectors are recombined into the compound (with boundary information) and this compound is compared against the original compound. Note that the percentage correctly annotated compounds is given below, not the percentage of correctly classified feature vectors.

4.1. System Parameters

The systems have been evaluated on a semi-automatically annotated (hand corrected) list of words. There are 40,250 unique words (compounds and non-compounds) in the list, with mean of approximately 16 characters per word (639,679 characters in total). The list of words is taken from the lexicon of the *PUK Afrikaanse Speltoetser*.

As we have indicated above, the LSM method depends heavily on a lexicon. For this purpose, we have used a sub-lexicon of the *PUK Afrikaanse Speltoetser* (i.e. the original lexicon without the compounds to be analysed). The decision tree classifiers were built using the same lexicon, with 90-10 division of train and test data. All experiments are done using ten-fold cross-validation. Results given below are the mean values of the ten runs.

Decision trees are build on feature vectors extracted from the list of compounds where the context size varies from three characters to the left and three to the right, to a context size of nine characters on each side. The context sizes to the left and the right are always taken the same, but preliminary experiments showed that having unequal context sizes to the left and the right does not give significantly different results.

The algorithm for creating the decision tree has another parameter, pruning. After creating the decision tree (as described above), the tree is generalised, reducing over-fitting of the training data. It results in smaller trees and often increases accuracy on unseen data.

4.2. Numerical Results

The results of applying the machine-learning systems to the data can be found in Figure 1. The LSM system works best, with an error rate of only 5.29% (not shown in the graph), which is much better than the results of the decision trees. The best decision tree algorithm is the one with a context of six characters to the left and to the right and pruning of 95%. This system has an error rate of 18.08%.

Increasing the pruning parameter (resulting in more pruning and thus smaller tree structures) results in significantly better results ($p<2.2E-16$). Comparing the results of the systems with respect to context gives less significant results. The results of context sizes five, six, and seven are not significantly different, but these contexts are significantly better than the other context sizes.

4.3 Properties

It is not entirely clear why the LSM method works much better than the machine-learning systems, but we suspect that this has to do with the fact that the compounds in the test set are taken from the lexicon used by the LSM method, where simplex words (i.e. non-compounds) related to the compounds are found abundantly. However, because no other compound list currently exists for Afrikaans, we were left with no choice to use other compound data.

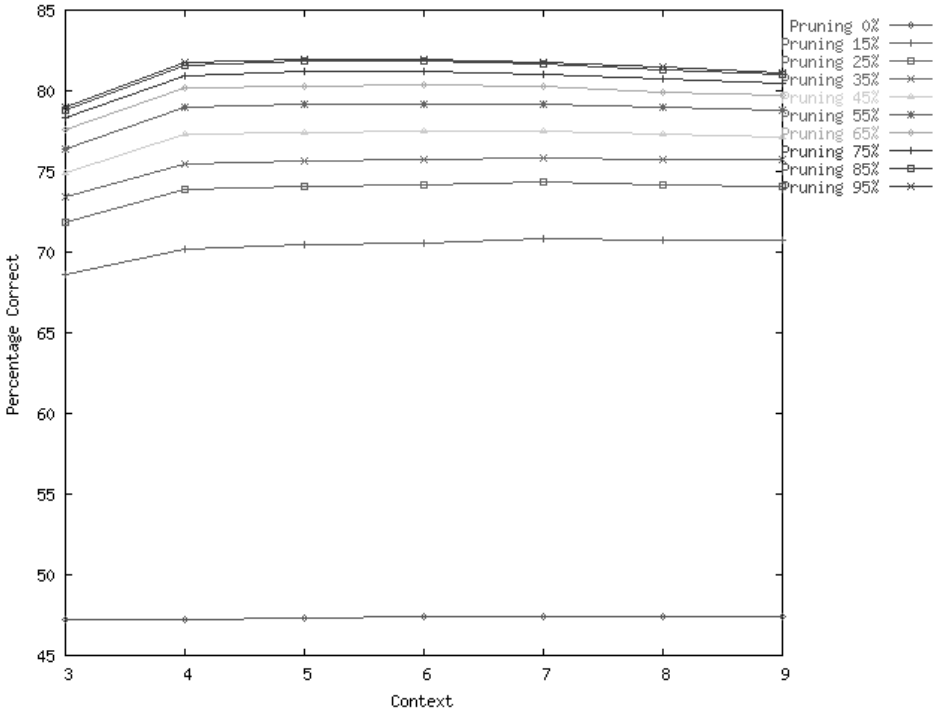

Figure 1: Results of machine-learning system

With respect to speed, the decision tree classifiers are much faster than the LSM method, because it needs fewer comparisons. The LSM method needs to find substrings of the compound in the lexicon, which is computationally intensive, whereas the classifiers only need to do some simple character comparisons per feature vector.

Another difference between the LSM and machine-learning methods is that the latter are much more robust with respect to unseen data. In contrast to the LSM method, a classifier does not see any difference between a compound with known components or with unknown components. All decisions are made locally where LSM needs global compound information.

5. Problems and Future Work

A number of problems with the current approach need to be addressed. Firstly, the results of the machine-learning approach are quite discouraging. It is unclear why this is the case, but we think there might be several reasons for this. There are not many training examples with respect to the size of the feature vector space. Perhaps increasing the size of the training data will improve results.

It may also be the case that decision trees are not the best machine-learning technique for the problem at hand. Since features are analysed one by one, a single error in the tree may lead to an incorrect result. Other machine-learning techniques, for example, based on maximum entropy or a memory-based approach are probably better suited because multiple features are used at the same time. In future research, other machine-learning techniques than decision trees will therefore be tested. In addition, more extensive parameter tuning will be performed. We hope that this will improve the results of machine-learning techniques.

A related issue is that in roughly 90% of the feature vectors1 no compound boundary or valence-morpheme boundary is present. Valence-morpheme boundaries occur in only 1% of the feature vectors. There is obviously a huge preference to return the “no boundary”-value. This can also be seen in the confusion matrix. Table 4 shows the confusion matrix with mean values of the experiments with six characters in the left and six in the right context and 95% pruning. Out of 59,862.6 feature vectors, most feature vectors are correctly classified as having no boundary. Almost all valence morphemes are correctly classified, while most errors are made where compound boundaries are not found (440.6 cases), and where incorrect compound boundaries are inserted (287.4 cases). Note that, with respect to the feature vectors, the machine learning system classifies 98.5% correctly.

Secondly, the LSM method performs quite well. The reason for this may be that the list of compounds was originally extracted from the lexicon that is eventually implemented in the LSM method.[2] The lexicon, part of the *PUK Afrikaanse Speltoetser*, was created from different sources, such as corpora and existing wordlists. The corpora contain texts on certain topics, so it may well be that compounds and non-compounds on these topics

[1] Remember that an feature vector is one position where a boundary can occur, i.e. between every character in a word.

[2] Compounds were extracted from the *PUK Afrikaanse Speltoetser* lexicon by taking words larger than a certain length. These words were hand-checked and annotated, and the remaining LSM lexicon was also hand-checked.

ended up in the lexicon. This means that the components of the compounds that are in the test data are specifically present in the lexicon used by the LSM approach. To get a better impression of the effectiveness of the LSM approach, a new test set needs to be developed, preferably with randomly found compounds. The main problem with this is that not many corpora exist for Afrikaans, and even when corpora are available, compounds need to be found by hand or semi-automatically.

class \ classified as	no boundary	valence morpheme	compound boundary
no boundary	53930.4	29.5	287.4
valence morpheme	44.2	582.5	40.6
compound boundary	440.6	45.4	4462.0

Table 4: Confusion matrix – results on context 6/6, 95% pruning

Finally, the original goal of finding compound boundaries is not only to find boundaries in correct words, but also in words that contain spelling errors. The fact that words containing errors cannot be found in the lexicon, and thus reducing the effectiveness of the LSM method, is precisely why a machine-learning approach has been chosen. We expect that such a system is much more robust with respect to spelling errors. Looking at the decision tree system, we see that decisions about word boundaries are taken based on only some characters in the context of the position at hand. Global comparison is not needed. We expect that machine-learning approaches to compound boundary recognition will perform better on compounds containing spelling errors.

Spelling errors are reflected in feature vectors in several ways. Substitutions (where characters are changed) will only have impact on that specific feature. Insertions and deletions will change larger parts of the feature vectors. This might have a bigger impact on the final results of the machine learning technique. However, decisions are made locally, which reduces the influence of spelling errors. Further research will show actual behaviour of machine learning approaches to compound boundary recognition in compounds containing spelling errors.

To evaluate the real effectiveness of the approaches on compounds containing spelling errors is not as easy as it might seem. Of course, a list of compounds containing spelling errors is needed. There are several means of obtaining such a list. Apart from gathering such a list manually, it could perhaps be generated automatically. Starting out from a list of compounds, spelling errors can be introduced as noise. The noise should model the human error process as closely as possible. For this, probabilities of specific types of errors need to be computed. These types of probabilities can be found in [7]. However, since these probabilities are for English, similar probabilities need to be computed for Afrikaans. The probabilities can then be used to introduce errors in the list of compounds. Additionally, care needs to be taken on the position of the error in the compound (i.e. whether it is near a compound boundary, or further away from it).

6. Conclusions

In this article, we discussed finding compound and valence-morpheme boundaries in Afrikaans compounds automatically. After discussing how compound formation works in Afrikaans, we have shown two methods of finding compound and valence-morpheme boundaries. The first method, LSM, depends on a lexicon of components that can be used to compose compounds, whereas the other method depends on a machine-learning technique. Here, we have taken a decision tree classifier and used a sliding window to encode the boundary information.

The results of the LSM method were significantly better than those of the machine-learning approach, but this may have several reasons: the training set of the machine-learning technique is perhaps too small, and the test set may be too much related to the lexicon used by the LSM implementation.

There are several advantages of a machine-learning approach. Compared to LSM, a machine-learning approach is much faster in finding compound boundaries, does not depend on a lexicon, and, finally, is much more robust than the LSM method. Since the ultimate goal is to find compound and valence-morpheme boundaries in words containing spelling errors, the robustness of machine-learning techniques is the most important reason why we have chosen this approach over a LSM approach for inclusion in the new Afrikaans spelling checker.

Acknowledgements

This research is partially funded by the National Research Foundation (GUN: 2053435), and the Research Focus Area: “Languages and Literature in the South African Context” at the North-West University, South Africa. Gerhard van Huyssteen is further supported by the Elisabeth Eybers bursary, awarded by the Albert Wessels Trust. No opinions in this article can be ascribed to any of these institutions.

We would like to acknowledge the help, ideas and insights of the following people: Roald Eiselen, Petri Jooste, Christo Muller, Suléne Pilon, Martin Puttkammer, Sansi Senekal, and Werner Ravyse. A special word of thanks to Martin Puttkammer for his assistance in applying and evaluating the machine-learning approach.

References

[1] Bauer, L. 1988. *Introducing Linguistic Morphology*. Edinburgh: Edinburgh University Press.

[2] Botha, R.P. 1968. *The Function of the Lexicon in Transformational Generative Grammar*. The Hague: Mouton.

[3] Botha, R.P. 1981. A Base Rule Theory of Afrikaans Synthetic Compounding. In: Moortgat, M., Van der Hulst, H. & Hoekstra, T. (eds.). *The Scope of Lexical Rules.* Dordrecht: Foris. pp. 1-77.

[4] Combrink, J.G.H. 1990. *Afrikaanse Morfologie*. [Afrikaans Morphology]. Pretoria: Academica.

[5] Daelemans, W. & Hoste, V. 2002. Evaluation of Machine Learning Methods for Natural Processing Tasks. In: *Proceedings of the Third International Conference on Language Resources and Evaluation (LREC 2002).* Las Palmas, Gran Canaria. pp. 755-760.

[6] Esterhuizen, H.L. 1986. *Semantiese verhoudings tussen die naamwoordkorrelate van Afrikaanse komposita*. [Semantic relations between noun correlates of Afrikaans compounds]. Unpublished D.Litt. thesis. Bloemfontein: University of the Orange Free State.

[7] Kernighan, M.D., Church, K.W. & Gale, W.A. 1990. A Spelling Correction Program Based on a Noisy Channel Model. *Proceedings of the 13th International Conference on Computational Linguistics (COLING).* Helsinki, Finland. pp. 205-210.

[8] Krott, A. 2001. *Analogy in morphology. The selection of linking elements in Dutch compounds*. PhD dissertation. Nijmegen: University of Nijmegen.

[9] Mattens, W.H.M. 1987. Tussenklanken in substantivische en adjectivische samenstellingen. [Binding sounds in nominal and adjectival compounds]. *Forum der Letteren*. 28: 108-114.

[10] Monz, C. & De Rijke, M. 2001. Shallow Morphological Analysis in Monolingual Information Retrieval for Dutch, German and Italian. In: Peters, C., Braschler, M., Gonzalo J. & Kluck, M. (eds.). *Evaluation of Cross-Language Information Retrieval Systems. Proceedings of the Second Workshop of the Cross-Language Evaluation Forum CLEF 2001).* Darmstadt, Germany, September 3-4, 2001. Berlin: Springer Verlag.

[11] Quinlan, J.R. 1993. *C4.5: Programs for Machine Learning.* San Francisco: Morgan Kaufmann.

[12] Schreuder, R., Neijt, A., Van der Weide, F. & Baayens, R.H. 1998. Regular plurals in Dutch compounds: linking graphemes or morphemes?. *Language and Cognitive Processes.* 13: 551-573.

[13] Spencer, A. 1991. *Morphological Theory*. Oxford: Blackwell.

[14] Suid-Afrikaanse Akademie vir Wetenskap en Kuns. (eds.). 2002. *Afrikaanse woordelys en spelreëls.* [Afrikaans Wordlist and Spelling Rules]. Ninth edition. Cape Town: Pharos Dictionaries.

[15] Van Huyssteen, G.B. & Van Zaanen, M.M. 2003. A Spellchecker for Afrikaans, Based on Morphological Analysis. In: De Schryver, G. (ed.). *6th International Terminology in Advanced Management Applications Conference: Conference Proceedings.* Pretora: $(SF)^2$ Press. pp. 189-194.

[16] Van Santen, A. 1985. *De Morfologie van het Nederlands*. [The morphology of Dutch]. 2nd edition. Dordrecht: Foris.

[17] Van Zaanen, M.M. & Van Huyssteen, G.B. 2003a. Improving a Spelling Checker for Afrikaans. In: Gaustad, T. (ed.). *Computational Linguistics in the Netherlands 2002: Selected Papers from the Thirteenth CLIN Meeting.* Amsterdam: Rodopi. pp. 143-156.

[18] Van Zaanen, MM & Van Huyssteen, GB. 2003b. Various Uses of Spelling Checkers: Learning, Teaching, and Practical Experiences. *Southern African Linguistics and Applied Language Studies.* 21(3): 327-340.