Automatic Lemmatisation for Afrikaans

A dissertation presented to

The School of Electrical, Electronic and Computer Engineering

North-West University

In fulfilment of the requirements for the degree

Magister Ingeneriae

in Electronic and Computer Engineering

by

Hendrik J. Groenewald

Supervisor: Prof. Albertus S.J. Helberg
Co-supervisor: Prof. Gerhard B. van Huyssteen
Assistant Supervisor: Prof. A. van den Bosch

November 2006

Potchefstroom Campus

DECLARATION

I hereby declare that all the material incorporated in this thesis is my own original work except where specific reference is made by name or in the form of a numbered reference. The work herein has not been submitted for a degree at another university.

Signed: 

Hendrik J. Groenewald

ACKNOWLEDGEMENTS

I wish to thank the following people and institutions:

- National Research Foundation, School for Electrical, Electronic and Computer Engineering and the Research Unit: Languages and Literature in the South African Context for funding.
- My supervisor, Prof. Albert Helberg for your professional guidance and advice.
- My co-supervisor, Prof. Gerhard van Huyssteen for always believing in me and inspiring (and pressuring!) me to work harder and do better.
- My assistant supervisor, Prof. Antal van den Bosch for your expert advice.
- The Director of the Research Unit: Languages and Literature in the South African Context, Prof. Attie de Lange for your interest in my studies and the informative post-graduate workshops that you organised.
- Sorita for all your love, prayers, words of encouragement and patience when I spend more time in *Lia*'s company than in yours.
- My parents, Frans and Elsabé for your unconditional love and support in whatever I do.
- My sister Ernalize for your help in the annotation of the training data and support throughout this study.
- Martin Puttkammer for advice on technical issues and the fact that you are always willing to help.
- Ulrike Janke and the rest of the team at CTexT for your encouragement and taking over my workload while I was studying.
- The Van Straten family for your support and interest in my work.
- Prof. Bertus van Rooy for your help with statistics.
- Elma de Kock, Annick Griebenouw and Jacinda Fourie for your help with the annotation of the training data.

ABSTRACT

AUTOMATIC LEMMATISATION FOR AFRIKAANS

By Hendrik J. Groenewald

A lemmatiser is an important component of various human language technology applications for any language. At present, a rule-based lemmatiser for Afrikaans already exists, but this lemmatiser produces disappointingly low accuracy figures. The performance of the current lemmatiser serves as motivation for developing another lemmatiser based on an alternative approach than language-specific rules. The alternative method of lemmatiser construction investigated in this study is memory-based learning.

Thus, in this research project we develop an automatic lemmatiser for Afrikaans called *Lia* "*Lemma-identifiseerder vir Afrikaans*" 'Lemmatiser for Afrikaans'. In order to construct *Lia*, the following research objectives are set: i) to define the classes for Afrikaans lemmatisation, ii) to determine the influence of data size and various feature options on the performance of *Lia*, iii) to automatically determine the algorithm and parameters settings that deliver the best performance in terms of linguistic accuracy, execution time and memory usage.

In order to achieve the first objective, we investigate the processes of inflection and derivation in Afrikaans, since automatic lemmatisation requires a clear distinction between inflection and derivation. We proceed to define the inflectional categories for Afrikaans, which represent a number of affixes that should be removed from word-forms during lemmatisation. The classes for automatic lemmatisation in Afrikaans are derived from these affixes. It is subsequently shown that accuracy as well as memory usage and execution time increase as the amount of training data is increased and that the various feature options have a significant effect on the performance of *Lia*. The algorithmic parameters and data representation that deliver the best results are determined by the use of *PSearch*, a programme that implements Wrapped Progressive Sampling in order determine a set of possibly optimal algorithmic parameters for each of the TiMBL classification algorithms.

Evaluation indicates that an accuracy figure of 92,8% is obtained when training *Lia* with the best performing parameters for the IB1 algorithm on feature-aligned data with 20 features. This result indicates that memory-based learning is indeed more suitable than rule-based methods for Afrikaans lemmatiser construction.

KEY TERMS

LEMMATISATION, MACHINE LEARNING, MEMORY-BASED LEARNING, HUMAN LANGUAGE TECHNOLOGY, NATURAL LANGUAGE PROCESSING, COMPUTER ENGINEERING, TIMBL, AFRIKAANS, MORPHOLOGY

OPSOMMING

OUTOMATIESE LEMMA-IDENTIFISERING VIR AFRIKAANS

Deur Hendrik J. Groenewald

'n Lemma-identifiseerder is 'n baie belangrike komponent in verskeie mensetaaltegnologiese toepassings vir enige taal. Tans bestaan daar 'n reëlgebaseerde lemma-identifiseerder vir Afrikaans, maar hierdie lemma-identifiseerder behaal ongelukkig baie lae akkuraatheidsyfers. Die teleurstellende prestasie van die reëlgebaseerde lemma-identifiseerder dien as motivering om 'n alternatiewe benadering vir die konstruksie van 'n lemma-identifiseerder vir Afrikaans te ondersoek. Die alternatiewe benadering vir die konstruksie van 'n lemma-identifiseerder vir Afrikaans wat in hierdie studie ondersoek word is geheuegebaseerde leer, 'n onderafdeling van masjienleer en kunsmatige intelligensie.

Ons ontwikkel dus in hierdie studie 'n outomatiese lemma-identifiseerder vir Afrikaans genaamd *Lia* (Lemma-identifiseerder vir Afrikaans). Ten einde die ontwikkeling van *Lia* moontlik te maak, identifiseer ons die volgende drie navorsingsdoelwitte vir hierdie projek: i) om die klasse vir Afrikaanse lemma-identifisering te definieer, ii) om die invloed van datastelgrootte en verskeie eienskapkeuses op die prestasie van *Lia* te bepaal, iii) om outomaties die algoritme- en parameterinstellings te bepaal wat die beste prestasie lewer in terme van akkuraatheid, uitvoersnelheid en geheuegebruik.

Ten einde die eerste doelwit te bereik, beskou ons die prosesse van fleksie en afleiding in Afrikaans, omdat outomatiese lemma-identifisering 'n duidelike onderskeid tussen fleksie en afleiding vereis. Ons definieer dan die fleksiekategorieë in Afrikaans, wat 'n aantal affikse verteenwoordig wat gedurende die proses van lemma-identifisering van woordvorme verwyder behoort te word. Die klasse vir outomatiese lemma-identifisering word op grond van hierdie affikse bepaal. Vervolgens word daar aangedui dat akkuraatheid, sowel as geheuegebruik en uitvoersnelheid verhoog met die gebruik van groter datastelle vir afrigting. Daar word ook aangetoon dat die insluiting van verskeie eienskapkeuses in die afrigtingsdata 'n statisties beduidende effek op die prestasie van *Lia* het. Die algoritme- en parameterinstellings wat die beste resultate lewer, word bepaal deur middel van *Psearch*, 'n

rekenaarprogram wat Begrensde Progressiewe Steekproefneming gebruik om 'n stel parameters vir elk van die klassifikasiealgoritmes te bepaal wat 'n goeie prestasie sal lewer.

Die evaluasieproses dui aan dat 'n akkuraatheidsyfer van 92,8% verkry word wanneer *Lia* afgerig word met die parameters en datastel wat die beste prestasie lewer vir die IB1 algoritme. Hierdie resultaat dui aan dat die geheuegebaseerde leerbenadering meer geskik is as reëlgebaseerde metodes vir die konstruksie van 'n lemma-identifiseerder vir Afrikaans.

SLEUTELTERME

LEMMA-IDENTIFISERING, MASJIENLEER, GEHEUEGEBASEERDE LEER, MENSETAALTEGNOLOGIE, NATUURLIKETAALPROSESSERING, REKENAARINGENIEURSWESE, TIMBL, AFRIKAANS, MORFOLOGIE

CONTENTS

LIST OF FIGURES

LIST OF TABLES

LIST OF ABBREVIATIONS AND ACRONYMS

ANOVA - Analysis of Variance

AUC - Area Under Curve

CALL - Computer Assisted Language Learning

CST - "*Center for Sprokteknologi*" (Centre for Speech Technology)

CTexT - Centre for Text Technology

df - degrees of freedom

DPS - Dutch Porter Stemmer

ED - Exponential Decay

FN - False Negatives

FP - False Positives

FPR - False Positive Rate

GUI - Graphical User Interface

HAT - "*Handwoordeboek van die Afrikaanse Taal*" (Desk Dictionary of Afrikaans)

HLT - Human Language Technology

ID - Inverse Distance

IG - Information Gain

IL - Inverse Linear

ILK - Induction of Linguistic Knowledge

k-NN - *k*-Nearest Neighbour

Lia - "*Lemma-identifiseerder vir Afrikaans*" (Lemmatiser for Afrikaans)

MBL - Memory-based Learning

MBLEM - Memory-based Lemmatiser

MBMA- Memory-based Morphological Analyser

Mdn - Median

ML - Machine Learning

MVDM - Modified Value Difference Metric

NLP - Natural Language Processing

NRF - National Research Foundation

Ragel - "*Reëlgebaseerde Afrikaanse Grondwoord- en Lemma-Identifiseerder*" (Rule-based Afrikaans Stemmer and Lemmatiser)

ROC - Receiver Operator Characteristic

SteDL - Stemmer with Dictionary Lookup

TiMBL - Tilburg Memory-Based Learner

TN - True Negatives

TP - True Positives

VAW - "Verklarende Afrikaanse Woordeboek" (Explanatory Afrikaans Dictionary)

WPS - Wrapped Progressive Sampling

Chapter 1: INTRODUCTION

1.1 CONTEXTUALISATION

The Centre for Text Technology (CTexT), a subdivision of the Research Unit: Languages and Literature in the South African Context at the North-West University, is one of the leaders in the field of natural language engineering in South Africa. The activities of CTexT revolve mainly around research and development of technologies and products that improve the technological status of the languages of South Africa. The product range of CTexT currently includes computer-assisted language learning (CALL) software, as well as spelling checkers for the indigenous languages isiZulu, isiXhosa, Afrikaans, Setswana and Sesotho sa Leboa.

Human Language Technology (HLT) applications for any language rely on various core technologies. One such a core technology is a morphological analyser, which is deemed one of the most important core technologies for HLT applications [1,2]. The process of morphological analysis entails the segmentation of a word-form into morphemes, combined with an analysis of the interaction of the morphemes that determine the syntactic class of the word [3].

Morphological analysers are not only used in various text-based systems (like grammar- and spelling checkers, information extraction systems and search engines), but also in speech-based applications such as speech recognisers [4]. It is therefore of utmost importance that effective morphological analysers should be developed for all South African languages in order to technologically enable these languages.

CTexT is currently in the process of developing various modules for text processing in Afrikaans as part of the NRF supported project entitled "Afrikaans Text Technology Modules" (FA2004042900059). Such modules include, among others, a lexical database, a shallow syntactic parser, as well as an automatic lemmatiser, which is the central theme of this thesis.

Lemmatisation is an important process for many applications of text mining and natural language processing (NLP) [5], and is defined as "a normalisation step on textual data, where all inflected forms of a lexical word are reduced to its common headword-form, i.e. lemma" [6]. For example the grouping of the inflected forms 'swim', 'swimming' and 'swam' under the base-form 'swim' is seen as an instance of lemmatisation. The last part of this definition applies to this project, as the emphasis is on recovering the base-form from the inflected form of the word. The base-form or lemma is the simplest form of a word as it would appear as headword in a dictionary [6].

Lemmatisation should however not be confused with stemming. Stemming is the process whereby a word is reduced to its stem by the removal of both inflectional and derivational morphemes [5]. Stemming can thus be viewed as a "greedier" process than lemmatisation, because a larger number of morphemes is removed by stemming than lemmatisation. Stemmers are usually employed in information retrieval applications (such as search engines) to reduce as many related words and word-forms as possible to a common form, which need not necessarily be the linguistically correct lemma. The emphasis in stemmers is not necessarily on linguistic correctness, but rather on robustness, as opposed to lemmatisers that produce linguistic accurate lemmas. A lemmatiser can thus be regarded as the linguistic variant of the stemmer.

There are essentially two approaches that can be followed in the development of lemmatisers, namely a rule-based approach [7] or a statistically/data-driven approach [5,8]. The rule-based approach is a traditional method for stemming/lemmatisation (i.e. affix stripping [7,9]) and entails the use of language-specific rules to identify the base-forms (i.e. lemmas) of word-forms. An example of such a language-specific rule for lemmatisation is for example to remove the string *tjie* when it occurs at the end of a word. The rule works fine in the case of the word "*seuntjie*" 'little boy', because the removal of the string *tjie* indeed produces the correct lemma "*seun*" 'boy'. The problem with the rule-based approach is that there are always exceptions to the rule. For example, the words "*leeumannetjie*" 'male lion' and "*sewejaartjie*" 'a flower species' both end with the string *tjie*, but in these two cases the *tjie* forms part of the lemmas of the words and should therefore not be removed.

Numerous efforts have been made to create rule-based stemmers and lemmatisers for various languages, with reasonably good results. Gaustad and Bouma [9] claim an accuracy of 79,23% for the Dutch Porter Stemmer (DPS) and 96,27% accuracy for the same stemmer when incorporating dictionary lookup (SteDL) using the complete CELEX database (evaluated on a set of 45 000 words). 97,02% of the words that had to be stemmed was already included in CELEX, implying that the remaining 2,98% of words was then stemmed with DPS, resulting in the high accuracy obtained. It is therefore clear that the accuracy of the SteDL approach depends on the coverage (i.e. the percentage of words in the language that is included in the lexicon) of the annotated (stemmed) lexicon. In the case of lemmatisation, "annotated" implies a lexicon of lemmatised words. Such a lexicon with high coverage is very important for improving the performance of a lemmatiser that incorporates lexicon lookup. A lexicon with high coverage will ensure that use of the rule-based part of the lemmatiser is kept to a minimum, thereby minimising the number of errors that may arise.

Jongejan and Haltrup [10] describe the development of the CST (Center for Sprokteknologi) lemmatiser, which is based on rules derived from data containing inflected word-forms, their lemmas and part of speech tags (where possible). The CST lemmatiser is language independent since it can be trained for different languages; the only requirement is that the language must only have inflectional suffixes and no inflectional prefixes (like German and Afrikaans). An accuracy figure of 94,5% (without part-of-speech tagging or dictionary lookup) is achieved by the CST lemmatiser for Danish when trained on an annotated lexicon of 450 000 word-forms.

It seems unlikely that a rule-based lemmatiser for Afrikaans will currently be able to achieve such a high success rate, due to the fact that there is no annotated lexicon for Afrikaans available yet. However, it has previously been attempted to use the rule-based approach for the construction of a rule-based stemmer/lemmatiser for Afrikaans (called *Ragel-* "*Reëlgebaseerde Afrikaanse Grondwoord- en Lemma-identifiseerder*" 'Rule-based Afrikaans Stemmer and Lemmatiser'). Although no formal evaluation of *Ragel* was done, it obtained a disappointing linguistic accuracy figure of only 67% in an evaluation on a random 1 000 word data set of complex words [11].

The alternative statistical/data-driven approach generally entails some form of statistical similarity function through which a word is lemmatised according to the example set by a similar word in a database of lemmatised words. This approach was used in the development of MBLEM (Memory-based Lemmatiser) [12], which is a lemmatiser for English, German, and Dutch. Its engine is a Tilburg Memory-Based Learner (TiMBL) server utilising data sets of English, German, and Dutch word-form lemma information. MBLEM performs a one-step mapping of instances to complex classes that contain the information needed to go from inflected form to the lemma. There is unfortunately no literature available regarding the development and performance of MBLEM.

An automatic lemmatiser for the Slovene language was created by Erjavec and Džeroski [6] who applied the same principles used in the construction of MBLEM. They split the problem of lemmatisation into two sub-problems: the first is to perform morphosyntactic tagging, while the second is to learn to perform morphological analysis which produces the lemma. A statistically-based tri-gram tagger was used to address the first problem and a first-order decision list learning system for the second. The tagger was trained on a manually annotated corpus of 100 000 words, while the morphologic analyser was trained on a morphological lexicon containing 15 000 lemmas. Erjavec and Džeroski [6] report an accuracy figure of 92% on the lemmatisation task.

Chrupala [13] also did some work on automatic lemmatisation, by constructing a lemmatiser based on Support Vector Machines, a machine learning algorithm. Chrupala was able to implement and evaluate his lemmatisation method for eight languages (Spanish, Catalan, Portuguese, French, Polish, Dutch, German and Japanese), since he had annotated data in eight languages available to serve as training data. The last 12 characters of every word, together with word context, were used as the features. For every language, a data set of 70 000 instances was used for training, and a data set of 10 000 instances was used for evaluation. The results for the eight languages are presented in Table 1 [13].

	Accuracy
Polish	80,29%
Spanish	86,35%
Portuguese	85,17%
Catalan	82,99%
German	78,88%
Japanese	89,54%
French	76,83%
Dutch	72,40%

Table 1: Lemmatisation evaluation for eight languages based on Support Vector Machines

Seeing that a rule-based stemmer/lemmatiser for Afrikaans already exists and that this stemmer/lemmatiser does not deliver satisfactory results, this study aims to develop a more effective lemmatiser based on statistical (specifically machine learning-based) methods. This lemmatiser will be called *Lia ("Lemma-identifiseerder vir Afrikaans"* 'Lemmatiser for Afrikaans').

Machine learning systems such as SVM-light [17] and SNoW (Sparse Network of Winnows) [18] could also have been used in this study to create a lemmatiser for Afrikaans. SVM-light is an implementation of Support Vector Machines in C, while SNoW is a multi-class classifier that is specifically tailored for large scale learning tasks. The learning architecture of SNoW consists of a sparse network of sparse linear functions. SNoW has been used successfully on a variety of large scale learning tasks. Considering the results of Chrupala [13], we also probably might have obtained good results (accuracy above 80%) if we chose to employ Support Vector Machines as learning algorithm. The disadvantage of the *k*-NN algorithm employed in TiMBL can be considered to be relatively slow in comparsion to other systems such as SVM-light and SNoW. The advantage of using TiMBL is that it supports both discrete and numeric features, unlike SVM-light and SNoW that only support numeric features. TiMBL therefore provides more room for experimenting with different feature options and even allows us to combine discrete and numeric features in the same experiment.

Given the scope of this research and the success of memory-based learning (MBL) in natural language applications [14,15], and specifically on the task of lemmatisation [6,8], we base this study on the assumption that MBL provides a feasible solution to the problem of lemmatisation. We will base *Lia* on the Tilburg Memory-Based Learner (TiMBL) [16], a program that implements several memory-based learning techniques.

1.2 Problem Statement

Memory-based algorithms will be used in this study to construct a classifier that can predict the lemmas of word-forms. The classifier is constructed by taking input data in the form of fixed-length patterns of feature-values and their associated class as input [16]. As simple as this might seem, some potential problems can be identified in this regard.

The first problem relates to the classes that the classifier must predict. The classes must consist of information that can be utilised to generate the correct lemma of the word-forms that were classified as inflectional word-forms. The logical way to go about the problem is to use grammatically motivated classes. For example, the class of the word "*hondjie*" 'puppy' should then be *-jie*, implying that the suffix *-jie* should be removed from the word to lemmatise it. This approach turns out to be problematic in some cases, such as "*beeldskone*" 'beautiful' where the correct lemma is "*beeldskoon*". The linguistically correct class of "*beeldskone*" is *-e* (attributive), but simply removing an *-e* on the right-hand side of "*beeldskone*" will leave us with "**beeldskon*" which is not a valid lemma. The use of grammatically motivated classes therefore seems to be problematic in such cases as "*beeldskone*", because they provide insufficient information for obtaining the lemma from the word-form. This provides motivation for finding alternative classes that will provide sufficient information for generating lemmas.

Another problem relates to the form of the training data, or more precisely, the various feature options available for the construction of the training data. Currently, the obvious way to go about this problem is to use letter sequences representing the spelling of the word-forms to be lemmatised. However, we believe that letter sequences are not the only feature-values that can be used for lemmatisation. For example, Mladenić [8] used word context as

supplementary feature-values, while Erjavec and Džeroski [6] additionally used part-of-speech tags as morphosyntactic descriptions. Supplementary features that are believed to provide information that may aid the system during the classification process should therefore be investigated in this study, including the number of features required for effective lemmatisation. If letter sequences are indeed used as feature-values, it makes sense to consider the length of the longest word in CTexT's Afrikaans lexicon [19] in order to enable every word in the lexicon to be fully represented. The longest word in CTexT's Afrikaans lexicon (i.e. "*radiotelefoonnoodfrekwensie-luisterdiensontvangstoestel*" 'radio telephone emergency frequency listening service reception device') consists of 56 characters, so 56 feature-values seem to be a good choice. The trouble with this is that the computational load on the system increases as the number of features is enlarged (i.e. the *curse of dimensionality*, see Chapter 2). It therefore seems to be better to use smaller numbers of features, but this on the other hand, entails that words consisting of more characters than the number of features used must be clipped to the desired size. Clipping of words, however, is not always desirable, because valuable information that aids the system during the classification process may possibly be discarded. Therefore, the representation of training data poses some challenges that should be thoroughly investigated.

Furthermore, according to Mitchell's definition of machine learning [20], machine learning systems improve as the amount of training data is increased. We can therefore assume that a large amount of training data is required to construct an accurate lemmatiser for Afrikaans. However, no training data in the desired format is currently available, which means that training data will first have to be annotated. To add to the problem, we do not know the exact amount of training data that is required for effective lemmatisation. We do however know that the annotation of training data is a time-consuming, labour-intensive process; so we must ensure that no more training data than necessary is annotated. The annotation process should therefore end when no significant further increase in accuracy can be achieved through the use of more manually annotated training data. We can thus conclude that determining the amount of training data required for effective lemmatisation is a vital step in the construction of *Lia*.

The amount of training data used is not the only factor that influences the accuracy of a machine learning system; it is a well known fact that the performance of machine learning systems varies as different combinations of classification algorithms and parameters are used. The problem is that we do not know the best algorithm and parameter combinations for effective lemmatisation in Afrikaans in terms of linguistic performance, execution time and memory usage. It is therefore clear that the best combinations for algorithm and parameter settings must be determined in some way.

One way of finding the best algorithm and parameter combination is to systematically do an exhaustive search, developing numerous classifiers with all possible permutations. This approach is however not desirable as it is computationally very expensive and time-consuming. However, a software package, entitled *Paramsearch 1.0 Beta* [21], is available that automatically determines combinations of algorithms and parameters that are expected to perform well on the task at hand. *Paramsearch* is also much faster than an exhaustive search. The problem with using *Paramsearch* is that it is currently only available for two of the five classification algorithms in TiMBL. We should therefore either use another software package that performs a similar task to *Paramsearch*, or we should extend the functionality of *Paramsearch* to enable it to be used with more than only two of the TiMBL algorithms.

1.3 Research Questions

The following research questions that arise from the problem statement will be addressed in this study:

1) What are the classes for Afrikaans lemmatisation?
2) What is the influence of the data size and various feature options on the performance of the system?
3) Which of the following algorithm and parameter settings deliver the best performance in terms of linguistic accuracy, execution time and memory usage?

    - Classification algorithm;
    - Feature weighting;

- Distance metrics;
- Class Voting;
- Number of nearest neighbours; and
- Frequency Threshold.

1.4 OBJECTIVES

In order to answer the above-mentioned research questions, this research has the following objectives:

1) To define the classes for Afrikaans lemmatisation;
2) To determine the influence of the data size and various feature options on the performance of the system; and
3) To automatically determine the algorithm and parameter settings that deliver the best performance in terms of linguistic accuracy, execution time and memory usage.

1.5 METHODOLOGY

In order to achieve the above-mentioned goals the following methods will be used in this research project:

1.5.1 LITERATURE REVIEW

During the first part of this project, a thorough, structured literature survey will be done on the following topics:

1) Memory-based learning; and
2) Lemmatisation.

The purpose of the literature study will be to gain knowledge about the latest advances in the field of MBL and automatic lemmatisation, which will be presented in Chapters 2 and 3

respectively. The knowledge gained through the literature study will aid the development of the lemmatiser for Afrikaans.

1.5.2 Data Annotation

As was pointed out in Section 1.2 above, the accuracy of MBL systems increases as the amount of training data is increased. The performance of MBL systems is also dependent on the form of the training data (the values of the features in the training data) and the number of features used.

The employed memory-based learning system requires the training data to be in a specific format. At the start of this project there is no data available in the specified format, therefore training data must be created. The basis of the training data will be words extracted from CTexT's Afrikaans lexicon that consists of approximately 350 000 words. The words that correspond in form to the inflectional categories defined in Section 3.4 will be extracted from the lexicon to serve as the basis for the annotated data (e.g. all words beginning with the string *ge*). Training instances that do not correspond in form to the inflectional categories will also be extracted to serve as negative training data. This will prevent the lemmatiser from being "eager" (i.e. a lemmatiser that lemmatises words that should not be lemmatised). The extraction will be done by means of a Perl script, using string matching to extract words containing strings corresponding to the inflectional categories.

Research assistants and undergraduate Afrikaans students will subsequently be used to help with the annotation of the data. The extracted data will be provided to the assistants in spreadsheets of 1 000 words each, where the extracted words will be in the first column of the spreadsheet. The annotation will be done by providing the linguistic correct lemma of every word in the second column of the spreadsheet. A manual for the annotation of the training data will also be compiled to assist the annotators in their task (see Addendum A). A recursive approach (i.e. bootstrapping) will be followed through which *Lia* will be used to generate her own training data after the first 20 000 words have been annotated. The recursive approach entails classifying more data in batches of 2 000 words each. Every new

batch will first be checked for errors by the assistants before it is added to the existing training data to serve as training data for the next batch.

1.5.3 DEVELOPMENT OF THE LEMMATISER

The design of the lemmatiser will be based on the information obtained through the literature study. This phase of the project will mainly consist of constructing the various components of the lemmatiser (described in Section 2.2). *Lia* will be based on the Tilburg Memory-Based Learner (TiMBL) [16].

Once the lemmatiser is operational, the focus of the project will shift to obtaining the best classification algorithm and parameter settings for the task automatically by using *Paramsearch* [21]. As explained earlier, *Paramsearch* is a programme developed to obtain a set of algorithm and parameter combinations that is expected to perform well on the task at hand. The various parameter options available in TiMBL are feature-weighting possibilities, distance metrics, class voting weights, number of nearest neighbours and frequency threshold. *Paramsearch* is currently not available for all the classification algorithms in TiMBL; therefore we will develop our own implementation of *Paramsearch* that will be able to operate on all the TiMBL algorithms. This adapted *Paramsearch* will also be used to determine the best data representation.

1.5.4 EVALUATION

The last phase of the project entails a thorough evaluation of the lemmatiser by utilising the algorithm and parameter settings that will prove to be the most effective for the task. The effectiveness is measured in terms of linguistic accuracy, memory usage and execution time. We view the performance metrics in the following order in terms of importance:

1) Accuracy
2) Execution time
3) Memory usage

Accuracy is viewed as the most important performance metric, since we aim to construct a lemmatiser for Afrikaans that achieves the highest possible accuracy. Execution time is considered to be the second most important metric, while memory usage is considered to be the least important metric since memory constraints are not expected to be a problem when you consider the relatively small data set that has to be stored in memory when constructing the classifier employed by *Lia*. Experiments with much larger training sets than those employed by *Lia* have been carried out without any problems in terms of memory-usage [16].

There are a number of standard metrics (such as accuracy and recall) that can be used to measure the linguistic accuracy of a lemmatiser; these performance metrics are introduced in Section 1.5.4.

Memory usage is defined as the number of megabytes in memory occupied by the training data. The different classification algorithms store the training data in different ways in memory, therefore it can be expected that differences in memory usage will be observed for different algorithms on the same set of training data. The execution time is measured in seconds as the time elapsed from the start of the process where the training is read into memory, until the classification of the last instance in the evaluation data set. All of the evaluation experiments were carried out on Pentium IV 3.0 GHz computers with 1GB RAM and Fedora Core 6 as operating system. The execution time is determined by means of a Perl script that utilises the Time::HiRes module, which is a Perl module for determining high resolution execution time. High resolution means that the execution time is reported to 6 decimal places.

The current baseline score for Afrikaans lemmatisation is the 67% accuracy figure obtained by *Ragel*. This study not only aims to improve on the accuracy score obtained by *Ragel*, but also to develop a lemmatiser with a high linguistic accuracy figure. We therefore need to set an accuracy figure that can be viewed as the standard for a successful automatic lemmatiser for Afrikaans. We define this standard by considering the results obtained in similar studies for other languages, as introduced in Section 1.1. We first consider the rule-based lemmatisers and then the statistical lemmatisers for other Germanic languages than Afrikaans.

It makes sense to compare *Lia* to lemmatisers for Dutch, since Afrikaans and Dutch are closely related. As was indicated in Section 1.1, Gaustad and Bouma [9] claim an accuracy figure of 79,23% for the Dutch Porter Stemmer without dictionary lookup. As mentioned previously, stemming is a more complicated process than lemmatisation, as it also involves the removal of derivational affixes from word-forms; we can therefore expect to achieve better results than those achieved by the Dutch Porter Stemmer. The lemmatisers constructed by Chrupala [13] are similar to *Lia* in the way that they were trained with relatively small amounts of training data. Chrupala claims accuracy figures of 72,40% for Dutch and 78,88% for German (see Table 1).

Jongejan and Haltrup [10] claim an accuracy figure of 94,5% for the CST lemmatiser for Danish (also a Germanic language like Afrikaans, although morphologically more complex than Afrikaans), based on a lexicon containing 450 000 words. Unfortunately, we will not have such a large lexicon available for the development of *Lia*, and we can therefore not expect of *Lia* to achieve such a high accuracy figure. We can expect that having access to a large lexicon should improve accuracy figures, this is however not always true because the amount of training data available is not the only factor that influences accuracy figures.

Since lemmatisation can also be viewed as a simplified process of morphological analysis [15], it seems sensible to consider the accuracy figures obtained in memory-based morphological analysis. Daelemans and Van den Bosch [3] have developed a memory-based morphological analyser (MBMA) for Dutch, where the morphological analysis task (i.e. the segmentation of word-forms into morphemes combined with an analysis of the interaction between the morphemes) can be viewed as considerably more complex than lemmatisation. Although morphological analysis is more complex than lemmatisation, we consider MBMA as an indication of a possible baseline score for *Lia*, because of the similarities between Dutch and Afrikaans. MBMA was trained with a lexical database consisting of 247 415 Dutch word-forms, and achieved an accuracy figure of about 90% on unseen words (correctly segmented and coarsely-labeled) [3]. The fact that MBMA is trained with more data could compensate for morphological analysis being a more complicated task than lemmatisation.

Based on the results obtained in comparable studies mentioned above, we therefore propose an accuracy figure of 90% as the standard for successful automatic Afrikaans lemmatisation. The performance of *Lia* will be compared to this standard in Chapter 5 to judge the suitability of memory-based learning techniques for constructing an automatic lemmatiser for Afrikaans.

For purposes of this study we will use 10-fold cross-validation as the default evaluation method. The process of 10-fold cross-validation consists of dividing the data into 10 equally sized sets or folds. The system is then evaluated on one of the folds, while trained with the remaining nine folds. This process is then repeated 10 times, using a different testing set and training set each time. The average mean accuracy is then calculated from the accuracy scores of the 10 different folds. The advantage of the 10-fold cross-validation process is that the entire data set (meaning every single instance) is used for training as well as for evaluation purposes.

10-fold cross-validation may however not be used as a method for determining an optimal combination of parameter settings, but should rather only be used for evaluating the performance of the classifier for a certain algorithmic parameter setting. The same data should never be used for parameter selection and error rate assessment, since this may produce overly optimistic accuracy figures. This will be the case if 10-fold cross-validation is used for parameter selection purposes. Therefore we once again emphasize that 10-fold cross-validation will be used in this study for evaluation purposes only, and not for determining optimal combinations of parameter settings.

1.5.4.1 Statistical Tests

The evaluation will also consist of Wilcoxon Signed-rank tests to determine if any statistical significant differences exist in the case where two classifiers are compared. Statistical significance implies that there is less than 5% chance of the observed effect occurring by chance [22]. Another statistical test that will be used is Friedman's analysis of variance (ANOVA) test, which will be used to determine if statistically significant differences exist between the means if more than two classifiers are compared. These two tests will be used

since they are non-parametric tests that do not make any parametric assumptions about the data.

i. Wilcoxon Signed-Rank Test

The Wilcoxon Signed-rank test [22] can be viewed as the non-parametric equivalent of the dependent *t*-test. The Wilcoxon Signed-rank test is used when a researcher wants to determine whether a statistically significant difference exists between the means of two groups. Non-parametric tests work on the principle of ranking the differences between measurements, where the number of positive ranks (T_+) and negative ranks (T_-) between measurements are computed. The test statistic (T) is the smaller value of T_+ and T_-. The Wilcoxon Signed-rank test requires that T, the significance level and the effect size be reported. The effect size (r) is an objective and standardised measure of the magnitude of the observed effect. The effect size can also be a negative value, which is useful when a relationship between two variables is determined, since the sign of r indicates the direction of relationship between the two variables [22]. Cohen [23] has made some suggestions about the magnitude of the observed effect:

- $r = 0{,}1$ (small effect)
- $r = 0{,}3$ (medium effect)
- $r = 0{,}5$ (large effect)

We will also report the median (*Mdn*) for each of the different groups that are compared with the Wilcoxon Signed-Rank test.

ii. Friedman's ANOVA

A Wilcoxon Signed-rank test is however not preferred when more than two groups are concerned. Instead, an ANOVA is performed when we want to compare more than two groups on the basis of a dependent variable [24]. The null hypothesis of ANOVA states that the means of the dependent variable scores (μ_k) for each level of the independent variable will not be significantly different.

$$\mu_1 = \mu_2 = \mu_k \quad \textbf{(1.1)}$$

Friedman's ANOVA is the non-parametric equivalent of the ANOVA used for normal distributed data. Friedman's ANOVA also operates on the principle of ranked data like the Wilcoxon Signed-rank test, and requires the reporting of the test statistic F, the degrees of freedom (*df*) and the significance level. The degrees of freedom are one less than the amount of groups that are being compared. For example if we are comparing the averages obtained by 3 different algorithms, the *df* will be 2. Friedman's ANOVA only determines if significant differences exist between the means of more than two groups. We use the Wilcoxon Signed-rank test as a post-hoc test to exactly determine where the significant differences lie. It is, however, required to correct for the number of tests that are performed when the Wilcoxon Signed-rank is used as a post-hoc test for Friedman's ANOVA. We correct for the number of tests that are performed by only accepting a significant result if its significance level is less than α (0,5)/number of comparisons. This type of correction is called a Bonferroni correction. The purpose of ANOVA is to provide reasons to reject or not to reject the null hypothesis. The null hypothesis is rejected when we are sure that the results obtained are not due to chance. The generally accepted significance level of $\alpha = 0{,}05$ is used throughout this study to accept or reject the null hypothesis. An alpha of 0,05 signifies a 5% chance that the observed effect is due to chance.

The Wilcoxon Signed-rank test and Friedman's ANOVA are carried out in this study by means of *SPSS 14 for Windows* [25], a statistical and analytical software package.

1.5.4.2 LINGUISTIC PERFORMANCE METRICS

A matrix like the one shown in Figure 1 can be constructed for each of the classes in the data [16]. Each classified instance can then be categorised in one of the four categories of the matrix.

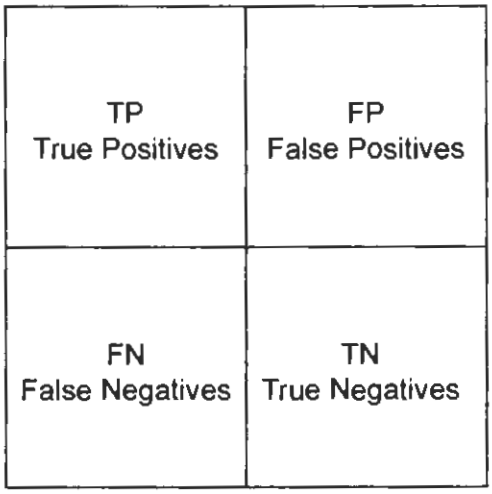

Figure 1: Matrix containing the categories for performance metrics

Suppose the class for which we are constructing the matrix is class *X*. The true positives (TP) cell will then contain the count of the instances of class *X* that were correctly classified as class *X*. The false positives (FP) are the number of instances of classes other than class *X* that were incorrectly classified as class *X*. The false negatives (FN) is the number of instances that do belong to class *X* but which were incorrectly classified, while the true negatives (TN) is the number of instances belonging to other classes which were not classified as having class *X*.

Using the categories of the matrix together with the total number of positive examples (P=TP+FN) and negative examples (N=FP+TN) we are able to calculate the following evaluation statistics:

i. Accuracy

Accuracy is a measure of a classifier's ability to predict the correct class. It is determined by dividing the number of instances correctly classified by the number of instances that was classified [16].

$$Accuracy = \frac{TP}{TP + FP} = \frac{Number\ Correctly\ Classified}{Number\ Classified} \quad \textbf{(1.2)}$$

ii. Recall

Recall measures the number of instances of a class that is recognised by the classifier. Recall is calculated by dividing the number of correctly classified instances by the number of instances that the classifier was supposed to classify [26].

$$R = \frac{TP}{P} = \frac{Number\ Correctly\ Classified}{Number\ Supposed\ To\ Be\ Classified} \tag{1.3}$$

iii. False Positive Rate (FPR)

FPR is a measure of the proportion of negative instances that were erroneously classified as being positive [16].

$$FPR = \frac{FP}{N} \tag{1.4}$$

iv. F-score

F-score is the harmonic mean of recall and accuracy and is therefore a good measure of the overall performance of the system [16,27].

$$F = \frac{2 \times precision \times recall}{precision + recall} \tag{1.5}$$

v. AUC in ROC space

The area-under-curve (AUC) is the surface of the grey area under the curve in the *receiver operator characteristics* (ROC) space in Figure 2 [16]. The ROC space is a graph of FPR on the x-axis and recall on the y-axis, while the ROC curve is a line that connects the origin of the graph and the upper right corner of the graph with a point in the ROC space [28]. For example, if *Lia* was able to correctly classify all of the instances with class X, then we would have a recall of 1 and a FPR of 0, representing a point in the upper left corner of the graph. This will result in an AUC of 1. The whole area in the ROC space would then have been covered in grey, indicating that the classifier can correctly predict the class of every instance in the evaluation data. A random classifier (0,5 recall and 0,5 FPR) will result in a straight

ROC curve from the origin to the upper right corner of the graph (the dotted line in Figure 2). Any results below the dotted line indicate very bad results, because it indicates that the classifier makes more erroneous classifications than correct ones. The AUC thus serves as an indication of percentage of instances correctly classified.

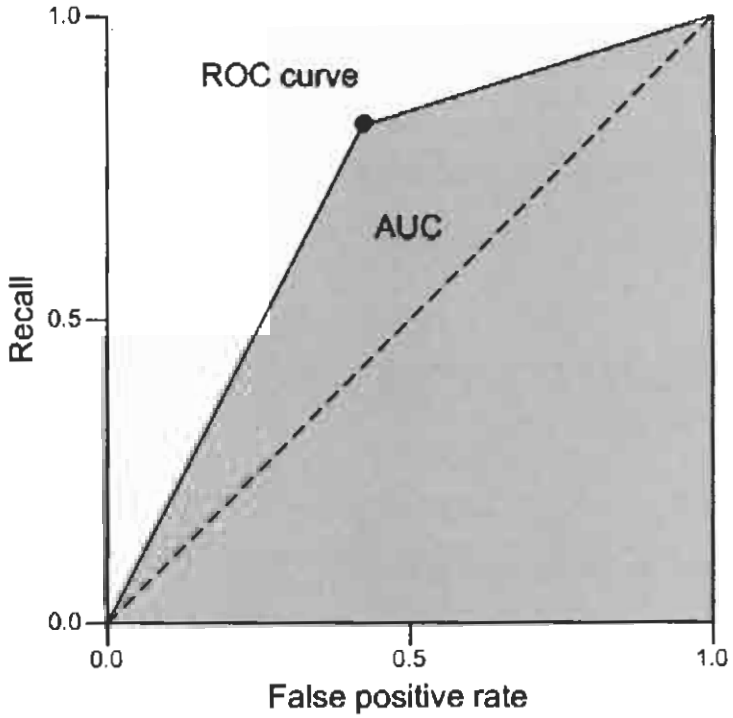

Figure 2: AUC in ROC space [16]

1.6 DEPLOYMENT

The goal of Chapter 2 will firstly be to present the architecture of *Lia*. This will be done with the aid of a graphic representation of *Lia's* architecture, where every step in the architecture will be explained and illustrated with suitable examples. This will be followed by the presentation of some background information about MBL with regard to lemmatisation. We will proceed to extend Mitchell's definition [20] of machine learning to memory-based learning of Afrikaans lemmatisation. Memory-based learning will then be introduced by providing details about the operation of the *k*-Nearest Neighbour (*k*-NN) classification algorithm. The concept description of MBL algorithms will be provided, along with the strengths and weaknesses of MBL. Chapter 2 will end with information on the working of

TiMBL, after which a short overview of the available algorithm and parameters will be presented.

Chapter 3 will focus on the classes for Afrikaans lemmatisation. The aim of Chapter 3 is to address **Research Question 1 (What are the classes for Afrikaans lemmatisation?)** by explicitly defining the classes for Afrikaans lemmatisation. The chapter will appropriately commence with background information deemed necessary to make an informed decision on the classes for Afrikaans lemmatisation. This background information includes the clarification of the distinction between inflection and derivation. Various viewpoints regarding inflection in Afrikaans will then be considered with the aim of identifying the inflectional affixes of Afrikaans. The chapter ends with a description of the classes for automatic lemmatisation.

The purpose of Chapter 4 is to describe various aspects of *Lia's* training data, more precisely the generation and presentation thereof. The focus of Chapter 4 is therefore on **Research Question 2 (What is the influence of the data size and the various feature options on the performance of the system?).** The chapter begins with an overview of the data generation process. Information about the processes of data extraction, data annotation and quality control is provided, followed by a section that illustrates the effect of training with increasing amounts of data on the accuracy of the system. Various feature options that may aid the system during the classification phase are also introduced. Chapter 4 concludes with an attempt to graphically represent the nearest neighbour relations in a small section of the training data.

The aim of Chapter 5 is to address **Research Question 3 (Which of the algorithm and parameter settings deliver the best performance in terms of linguistic accuracy, execution time and memory usage?)**. The chapter commences with an introduction to the operation of the classification algorithms and parameter options that comprise the performance component of *Lia*. The second part of Chapter 5 introduces the process of Wrapped Progressive Sampling, which forms the basis for the operation of *Paramsearch*. *Paramsearch* will be evaluated by comparing it to an exhaustive search for determining the best parameter settings. The purpose of this comparison is to determine if *Paramsearch*

represents an effective alternative to an exhaustive search. Chapter 5 concludes with a section on the algorithmic parameter options that deliver the best performance on the lemmatisation task, which is determined with the aid of our own implementation of *Paramsearch*.

A summary of the project is presented in Chapter 6. This is followed with some general conclusions on automatic lemmatisation for Afrikaans, based on the results obtained by *Lia*. The chapter concludes with directions and recommendations for future work regarding automatic lemmatisation for Afrikaans.

Chapter 2: CONSTRUCTING LIA: MEMORY-BASED LEARNING FOR LEMMATISATION

2.1 INTRODUCTION

In Chapter 1, we motivated our assumption to adopt a statistical approach to lemmatisation, specifically a memory-based learning (MBL) approach. MBL was specifically chosen due to the successes achieved by the use thereof in the past with regard to NLP tasks similar to lemmatisation [3,29]. This chapter is therefore not linked to a specific problem statement; instead the aim of this chapter is rather to present general background information about MBL.

We commence the chapter by introducing *Lia's* architecture. We provide a graphic representation of the architecture and explain the purpose of every phase in the architecture. This is followed by a section where the lemmatisation task is related to memory-based learning by extending Mitchell's [20] definition of machine learning to lemmatisation learning. It is then indicated that the lemmatisation learning task of this study is seen as an example of supervised learning or learning from examples (the training data is the set of examples used for learning). MBL in general is then introduced by focusing on the *k*-Nearest Neighbour (*k*-NN) algorithm, concept description, strengths and weaknesses of MBL, and the data requirements of MBL algorithms. This chapter concludes with information regarding the Tilburg Memory-Based Learner (TiMBL). The operation of TiMBL is described, together with a brief overview of the algorithm and parameter options available in TiMBL.

2.2 ARCHITECTURE OF LIA

The architecture of *Lia* is depicted in Figure 3. The architecture indicates that *Lia* consists of various consecutive, yet dependent processes; therefore we can refer to *Lia* as a system. The process starts with the user presenting a suitable Afrikaans word (or word list) to be lemmatised to the system. The words of the wordlist need to be in the form of a plain text

file, with one word per line. There are no restrictions or naming conventions on the filename of the text file; the name can be defined by the user.

This word (or wordlist; hereafter referred to as the evaluation instance(s)), is then formatted according to the format of the data (see Section 4.4) on which the classifier was trained. Say for instance that the word to be lemmatised is "*geslaap*" 'slept' and that the format of the involved training data is right-aligned data with 20 features. The word will then be formatted as "_,g,e,s,l,a,a,p". The purpose of the underscores appended to the left side of the word is to increase the number of features to 20.

The formatted word is now presented to the classifier, where the objective is to produce a class for the evaluation instance. The classification is performed on the basis of the algorithmic parameters and training data used in the construction of the classifier. This produced class contains the information deemed necessary for generating the lemma of the evaluation instance. If a correct classification is made in the case of "*geslaap*", the allocated class will be *Lge>* (see Section 3.5).

The next step in the process entails the generation of the lemma. This step first entails the removal of the underscores and commas appended to evaluation instance(s) during the formatting phase. The predicted lemma is then generated by considering the awarded class. In the case of "*geslaap*", the awarded class states that the string *ge* should be removed at the left-hand side of the word. This produces the word "*slaap*" 'sleep', which is indeed the correct lemma of "*geslaap*".

The last step in the process entails the generation of the system output. The output is the original evaluation instance(s) together with its awarded class(es) and predicted lemma(s). In the case of "*geslaap*", the output will be "geslaap - Lge> - slaap".

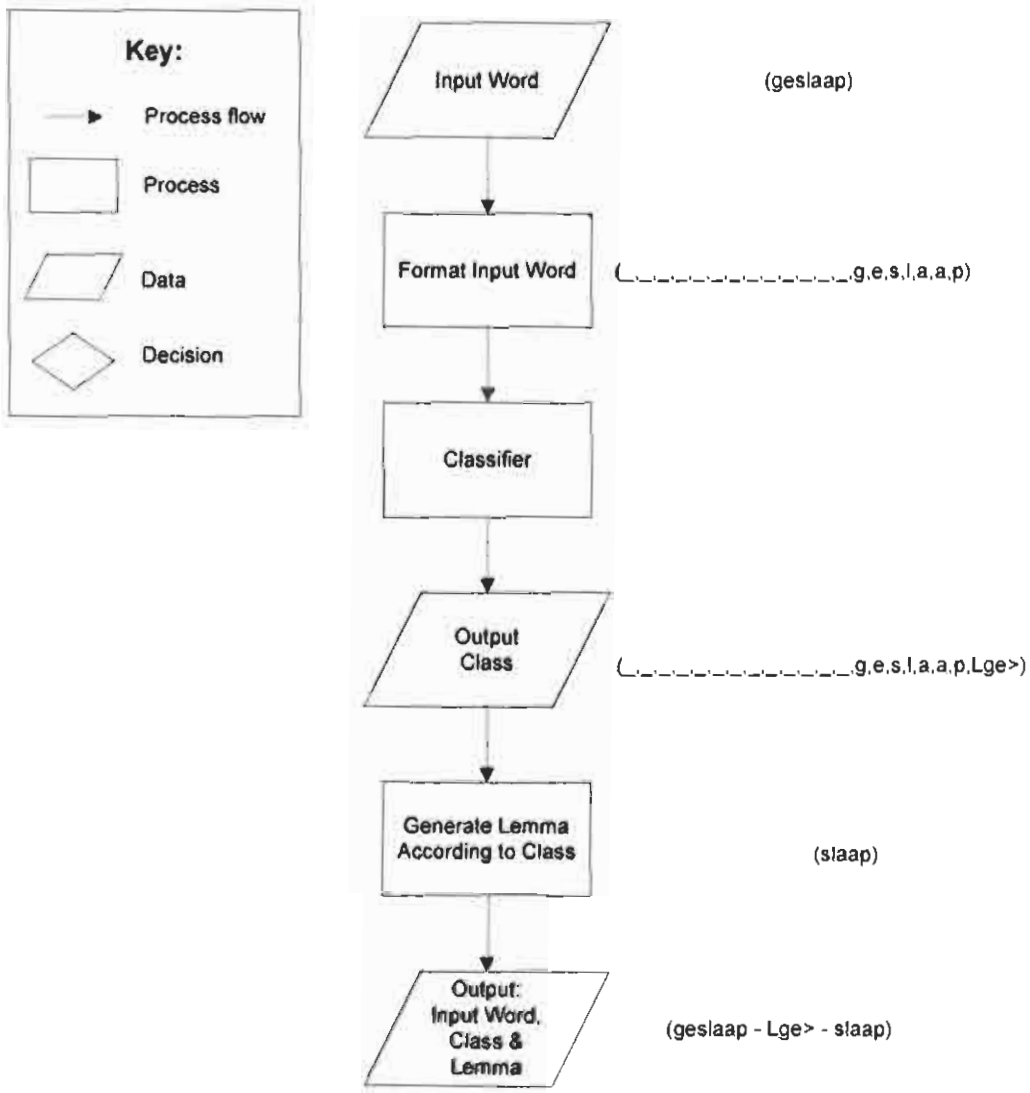

Figure 3: Architecture of *Lia*

The TiMBL-based classifier (see Section 2.5) can be viewed as the heart of *Lia*, since lemmatisation is performed according to the class(es) assigned to the evaluation instance(s). It is very important to construct the classifier for optimum performance, since the performance of *Lia* is directly dependent on the performance of the classifier.

2.3 The Lemmatisation Learning Task

The lemmatisation problem can be described as finding a mapping from a pattern of symbolic features (letters) to a symbolic class [30], which carries information about the transformation

from the given word to its lemma. The class of an instance can be predicted by observing the predictor feature attributes: the further away a feature (representing context) is from the target, the less relevant it is. The instance space is reasonably large, and there are also many training instances available. The problem is described as noisy and complex with many sub-regularities and exceptions from a linguistic point of view (see Chapter 3).

Based on this characterisation of the lemmatisation problem, it is henceforth assumed in this study that lemmatisation is an ideal task to resolve with machine learning (ML), and more specifically with memory-based learning.

The field of ML research is concerned with the construction of and ways to construct machines of programs that learn and improve with experience [20]. ML is a tool which is suitable for studying large databases, recognising patterns and making intelligent decisions based on these patterns. The advantage that ML has over conventional programming is that it is in principle much more versatile and capable of adapting to changing conditions [31]. ML is suitable for solving problems that cannot be effectively resolved by conventional programming techniques.

Mitchell [20] defines ML as follows:

> A computer program is said to **learn** from experience *E* with respect to some class of tasks *T* and performance measure *P*, if its performance at tasks in *T*, as measured by *P*, improves with experience *E*.

Mitchell's definition of machine learning can be extended to formulate our basic assumption in this study:

> *Lia* is said to learn from a database of correctly lemmatised words (i.e. Experience), with respect to lemmatisation (i.e. Task) and the percentage of correctly lemmatised words (i.e. Performance Measure), if its performance at lemmatisation (**T**), as measured by the percentage of correctly lemmatised words (**P**), improves as the size of the database of correctly lemmatised words is increased (**E**).

This implies that *Lia* will improve (learn) with more and more experience (i.e. a larger and better database of correctly lemmatised words), so that predictions about new cases can be made based on the outcomes of similar cases in the past.

Based on the above, the lemmatisation learning task of this study can best be described as supervised learning, or learning from examples [32], where *Lia's* training data is the "example" used for learning. In the case of supervised learning, the lemmatiser is provided with a set of input words (i.e. the original words to be lemmatised), as well as a set of output words (i.e. the linguistically correct lemmas). The lemmatiser therefore has to learn to produce the correct output, given a particular input from the same domain as the data that is used for training.

Alternatively, unsupervised learning entails that the system is not provided with the linguistically correct lemmas, since unsupervised learning involves how systems "can learn to represent particular input patterns in a way that reflects the statistical structure of the overall collection of input patterns" [33]. Thus, the emphasis of an "unsupervised" lemmatiser is in the first place not on lemmatisation per se, but rather on finding patterns in the data (that could be utilised for, amongst others, lemmatisation).

The main difference, therefore, between supervised and unsupervised learning is that there are no explicit target outputs associated with the input in unsupervised learning. However, since we will be using MBL to resolve the task of lemmatisation, our approach can best be described as an instance of supervised learning.

2.4 MEMORY-BASED LEARNING

The purpose of this section is firstly to introduce the most basic memory-based learning algorithm called the *k*-Nearest Neighbour (*k*-NN) algorithm. The operation of *k*-NN is described and illustrated by means of a graphical representation. We also describe the concept description of memory-based learning algorithms and its strengths and weaknesses. This section ends with the data requirements for memory-based learning.

2.4.1 THE *K*-NEAREST NEIGHBOUR ALGORITHM

Memory-based learning, also known as instance-based learning [34], is based on the classical *k*-NN classification algorithm. *k*-NN has become known as a powerful pattern classification

algorithm [16], and is considered the most basic instance-based algorithm. The assumption here is that all instances of a certain problem correspond to points in the *n*-dimensional space R^n [35]. The nearest neighbours of a certain instance are computed using some form of distance metric $\Delta(X,Y)$. This is done by assigning the most frequent category within the found set of most similar example(s) (the *k*-Nearest Neighbours) as the category of the new test example. In case of a tie amongst categories, a tie breaking resolution method is used [16].

Figure 4 illustrates the operation of the *k*-NN algorithm for the case where the instances have boolean class values. The + and - signs indicate the different classes of the instances in the feature space. Xq is the query point (the point that needs to be classified). It is clearly shown in Figure 4 that the 1-NN algorithm will classify Xq as positive, while the 5-NN algorithm will classify Xq as negative.

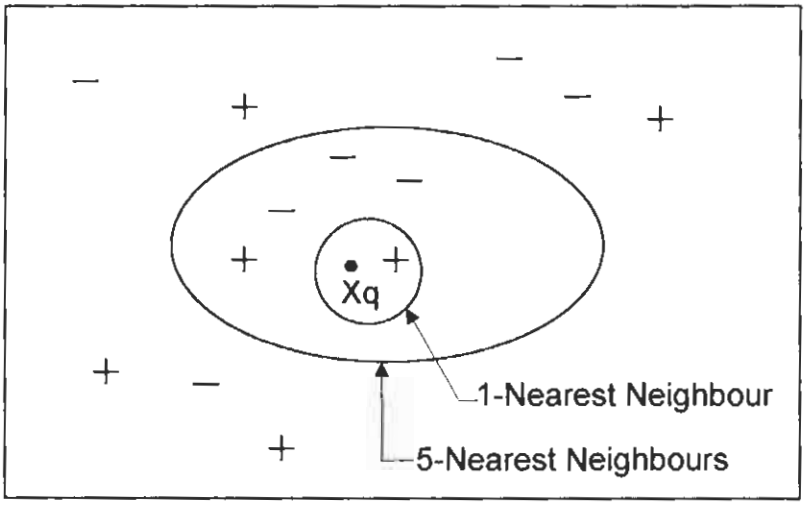

Figure 4: Graphical Representation of the *k*-Nearest Neighbour concept

2.4.2 The MBL Classification Concept

The output of the MBL algorithms used in this study is a function that maps instances to classes. The goal of this function is to provide a class for every instance.

MBL requires training data, which is a set of stored instances from which the system learns to perform classification. This set of instances may change after each training instance is

processed. MBL is based on the idea that similar instances have similar classifications. This leads to their bias for classifying instances according to their most similar neighbour's classification (the nearest neighbours).

The following functions describe all MBL algorithms [35]:

1. *Similarity Function:* This computes the *similarity* between a training instance *i* and the instances in the concept description. The similarities have numerical values.

2. *Classification Function:* The classification function receives the similarity function's results and the classification performance records of the instances in the concept description. It yields a classification for every instance in the evaluation set.

2.4.3 Strengths and weaknesses of Memory-based Learning

The strengths and weaknesses of MBL presented hereafter are taken from Wagacha [34].

Strengths:

- The *k*-NN algorithm is robust in terms of noisy training data. Provided with sufficiently large training data sets, it has been shown to be quite effective. The concept here is that by taking the weighted average of *k* neighbours nearest to the query point, it can smooth out the impact of isolated training examples.
- Information is never lost because the training instances are stored explicitly in memory. This means that all information regarding the training instances are stored and no abstractions take place.
- The *k*-NN algorithm has the ability to model very complex target functions. Lemmatisation can be viewed as a complex target function because of all the irregularities and exceptions that the task entails (see Chapter 3). Establishing rules for lemmatisation/stemming (i.e. the approach used in *Ragel*) has not been highly successful and this can be specifically attributed to these irregularities and exceptions. Due to the above mentioned strengths of MBL (robustness to noisy training data, low

level of abstraction), MBL is expected to deliver better results than the rule-based approach.

Weaknesses:

- Large volumes of annotated training data are required by MBL. Machine learning algorithms improve with experience, and in the case of MBL this experience relates to the volume of training data used. An example of a NLP task that requires a large volume of training data is the Memory-based morphological analyser (MBMA) for Dutch [3], which was trained with 247,415 fully morphological analysed words. Determining the minimum amount of training data required for obtaining a certain performance level is very important, especially in the case of *Lia* where large amounts of training data are not available. The problem regarding the size of the training data set required for reaching an accuracy level of 90% (see Section 1.5.4) will be addressed in Chapter 4.
- All of the attributes of the training instances are considered when attempting to retrieve the nearest neighbours. Irrelevant attributes (or features) may result in lower accuracy during classification. Identifying the relevant features that actually aid the system during the classification task and discarding irrelevant features are thus a very important part of the data construction process of memory-based learning systems. The problem of identifying the relevant features for *Lia* is addressed in Chapter 4.
- Instance-based methods have a high cost in terms of memory usage and computational effort in classifying instances. The reason for this is that nearly all computation takes place at classification time rather than when the training examples are first encountered. However, there are additional algorithms (such as IGTree and TRIBL) available that could be used to significantly reduce the computation required at query time. These algorithms are introduced in Chapter 5.

2.4.4 Data Requirements

MBL requires that each example or instance in the data set be represented by a set of feature-value pairs [35]. All instances must be described by the same set of attributes. The set of attributes defines an n-dimensional instance space where n is equal to the number of features

used to represent the instances. One of these attributes corresponds to the category attribute or class, while the other attributes are predictor attributes.

2.5 TiMBL

TiMBL is an MBL system that employs a number of efficient MBL algorithms with the goal of constructing classifiers for a particular task[1]. TiMBL is a project of the Induction of Linguistic Knowledge (ILK) research group at the University of Tilburg in the Netherlands, and was developed by Walter Daelemans, Jakub Zavrel, Ko van der Sloot and Antal van den Bosch [16]. Researchers at the University of Tilburg have been working on the development of MBL techniques and algorithms since the late 1980's, and TiMBL can be viewed as the product of this research. The first version of TiMBL was published in 1998, and since then it has grown into a powerful tool for NLP research. TiMBL was developed with NLP applications in mind, but it can be successfully used for any type of classification task that relies on learning from examples. The latest version of TiMBL, TiMBL 5.1, was used for purposes of this study.

TiMBL relies on two files for operation: a training file and an evaluation file. Each of these files contains a number of instances in the same format. The only difference between the files is that the training file usually contains more instances than the evaluation file. Figure 5 illustrates the basic operation of TiMBL. TiMBL ignores the category attribute (i.e. class) of the evaluation file during the evaluation phase.

The process as a whole can be divided into three phases:

- Phase 1 - Examining the training file;
- Phase 2 - Learning from the training file; and
- Phase 3 - Applying the classifier to the evaluation set.

Phases 1 and 2 can be viewed as the construction of the classifier, with Phase 3 as the evaluation phase. TiMBL is started by specifying a training and an evaluation file, together

with some algorithmic parameters (default settings[2] are used if no algorithm is specified). The first two phases of the process will proceed if a training file in the correct format is present.

Phase 1 entails the examination of the contents of the training file, followed by the computation of a number of statistics based on this examination. These statistics include the *Information Gain* and *Gain Ratio* of the attributes. Phase 2 is the learning phase and this merely consists of efficiently storing instances to memory according to the algorithm specified earlier. The system will exit at this point if an evaluation file was not specified at the start, writing the results of the first two phases to an output file. Unless otherwise stated, the name of this output file is a combination of the training file, the chosen algorithm and parameter settings.

If an evaluation file was indeed specified, the process will continue to Phase 3, which consists of applying the classifier created in the two previous phases to the evaluation data, and writing the results to the output file. The results to be reported can be customised by the user, but TiMBL will by default report the accuracy obtained (percentage of correctly classified test instances).

[1] The information presented in the following paragraphs is obtained from Daelemans *et al.* [1], unless otherwise stated.
[2] IB1 with information-theoretic feature weighting

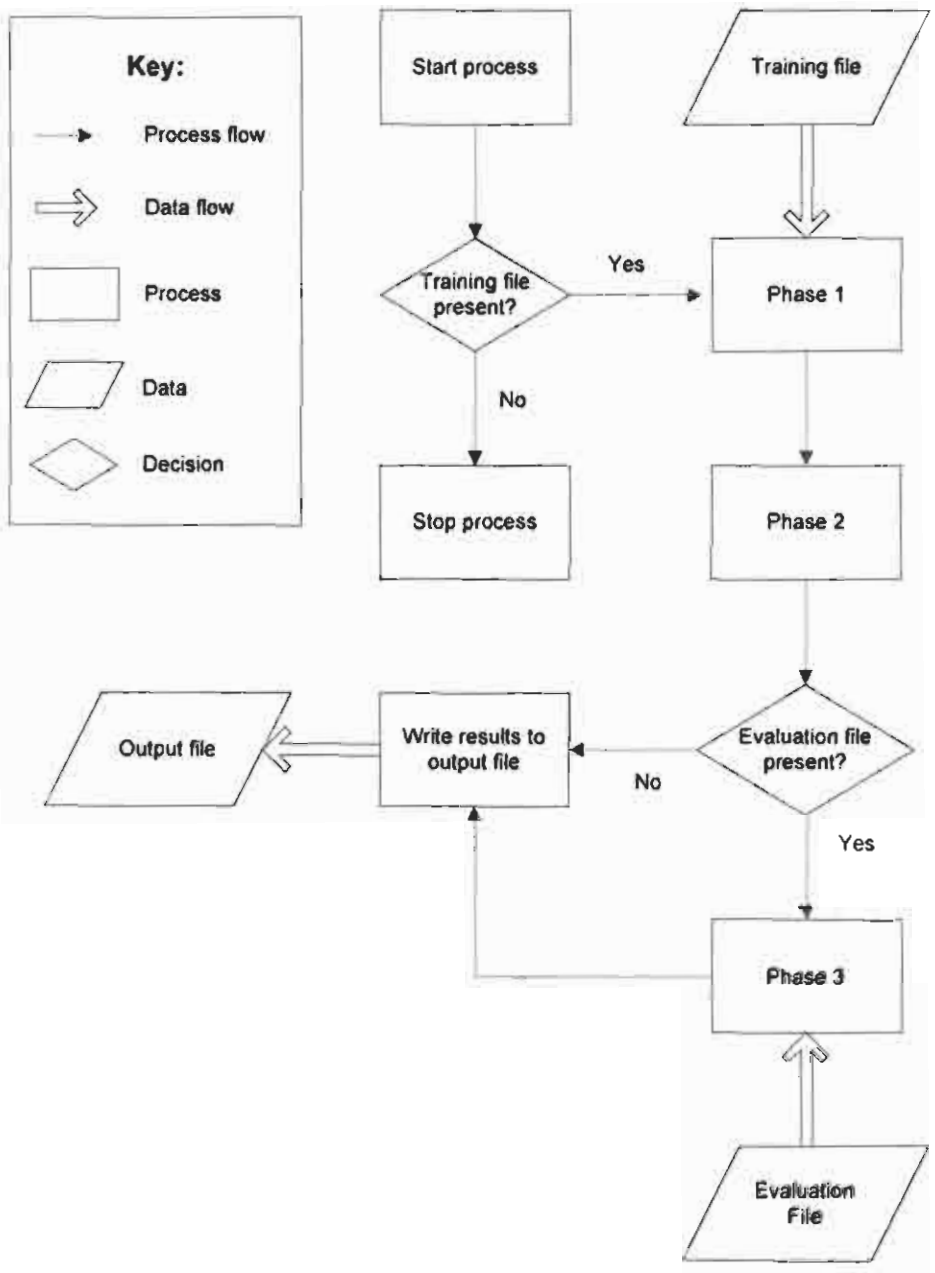

Figure 5: Flowchart of the working of TiMBL

There are a several algorithms and parameter options available in TiMBL, which will be discussed in detail in Chapter 5. These options are:

- Classification algorithm (*IB1, IGTree, TRIBL, IB2, TRIBL2*);
- Distance Metric (*Overlap, Modified Value Difference Metric, Jeffrey Divergence*);
- Feature weighting possibilities (*No Weighting, Gain Ratio, Information Gain, Chi-squared, Shared Variance*);

- Number of nearest neighbours;
- Frequency threshold for switching from *Modified Value Difference Metric* to *Overlap*;
- Class voting weights (*Normal Majority Voting; Inverse Distance Weighting, Inverse Linear Weighting and Exponential Decay Weighting*)

Different formats for the training and evaluation files are supported by TiMBL, including Compact, C4.5, ARFF, Columns, Sparse and Binary. In this research, the C4.5 format [36] is the default method by which the training data is presented to TiMBL. C4.5 format implies that attribute values are separated by commas and that the last attribute value denotes the class of the instance. C4.5 therefore requires attribute values (i.e. features) and classes. The classes and attribute values will be respectively defined in Chapters 3 and 4. Figure 6 shows training data in C4.5 format for determining whether an object can be classified as a screw, pen, key or scissors. The attribute values or features are size, shape and number of holes and are represented by the first 3 comma separated values. The last attribute (in bold) is the class, and this represents the classification, as illustrated in Figure 6.

small, compact, none, **screw**
large, long, 1, **pen**
small, other, 2, **key**
large, other, 2, **scissors**
small, compact, 1, **nut**
small, long, 1, **key**

Figure 6: Training data for in C4.5 format

2.6 CONCLUSION

The goal of this chapter was to present some background information about the system architecture of *Lia*, as well as MBL specifically relating to lemmatisation. This chapter therefore commenced with a section where the architecture of *Lia* was introduced and the various consecutive phases explained. We then focused on the lemmatisation learning task, and related it to memory-based learning by extending Mitchell's definition of machine learning [20] to the lemmatisation learning task for Afrikaans. The nature of the lemmatisation problem in Afrikaans (i.e. a noisy, complex problem with many sub-

regularities) provided further motivation for using MBL to resolve the problem of lemmatisation in Afrikaans. It was then stated that the lemmatisation learning task of this study is seen as an instance of supervised learning, or learning from examples.

Memory-based learning in general was introduced in Section 2.4 by explaining the operation of the *k*-NN algorithm, the basic memory-based algorithm. The *k*-NN algorithm is used to compute the nearest neighbours of a query instance by using some form of distance metric. The concept description of MBL algorithms was introduced next, which consists of a set of stored instances (training data), a similarity function, classification function and an output in the form of a classification. The concept description is a function that maps instances to classes. The strengths and weaknesses of MBL algorithms were also discussed, followed by the data requirements of MBL.

The chapter concluded with information regarding TiMBL. Some background information on TiMBL was first presented, where after the focus moved to a detailed description of the three phases (i.e. examining the training file, learning from the training file and applying the classifier to the evaluation set) constituting the operation of TiMBL. The first two phases deal with the construction of the classifier, with the last phase being the evaluation phase. The section on TiMBL concluded with a brief overview of the algorithm and parameter options available in TiMBL.

Chapter 3 will now focus on the linguistic aspects of lemmatisation. The various viewpoints regarding inflection in Afrikaans will be considered with the emphasis on explicitly defining the inflectional affixes required for automatic lemmatisation for Afrikaans.

Chapter 3: CLASSES FOR LEMMATISATION IN AFRIKAANS

3.1 INTRODUCTION

Where the focus of Chapter 2 was on the process of MBL for lemmatisation, the purpose of this chapter is to define the classes for Afrikaans lemmatisation. Afrikaans is a morphologically productive language, meaning that valid new words may be formed by means of word-formation processes such as compounding and derivation [37]. In addition, Afrikaans also has various processes of inflection, specifically on nouns, verbs and adjectives [38]. For purposes of this project, we are interested only in inflection and not in derivation, since *Lia* will only be trained to remove inflectional affixes from inflected words for purposes of automatic lemmatisation.

Automatic lemmatisation in Afrikaans is still an unresolved issue, as nothing has been written about what automatic lemmatisation in Afrikaans entails and how it should be done. Where a large body of literature is available on the subject for languages such as English and Dutch, this is unfortunately not the case for Afrikaans. However, within lexicography, lexicology and morphology, much has been written about what should be considered a lemma in Afrikaans [39]. Since clearly distinguishing between inflection and derivation in Afrikaans is an essential aspect of automatic lemmatisation, this chapter will give an explanation of inflectional processes in Afrikaans, in order to determine the affixes that should be removed by *Lia*.

This chapter is therefore directly linked to **Research Question 1**, since we will identify the classes for automatic Afrikaans lemmatisation. The chapter commences by explaining the distinction between inflection and derivation, after which an overview of the debate on inflection and derivation in Afrikaans is given. No in-depth investigation will be conducted, however, since this is not the main focus of this research. We will rather only provide a summary from standard works of reference. The Afrikaans inflectional categories are then defined for purposes of this study. These inflectional categories form the basis for a

discussion regarding the classes for Afrikaans lemmatisation, before these classes are defined in a final section of this chapter.

3.2 INFLECTION AND DERIVATION

Inflection and derivation are traditional notions in the domain of morphology, the sub-discipline of linguistics that studies the internal structures of complex words [38]. Both are morphological processes, but of a different nature. The process of inflection refers to the creation of different forms of the same lemma, while different lemmas are formed through the process of derivation. For instance, the Afrikaans words "*speler*" 'player', "*spelers*" 'players' and "*spelertjie*" 'little player' are considered different word-forms of the lemma "*speler*". None of these three different forms has their own lemma entries in an Afrikaans Dictionary such as the *Verklarende Handwoordeboek van die Afrikaanse Taal* (HAT); instead they are dealt with under the same entry for the lemma "*speler*". The suffixes *-s* (as in "*spelers*") and *-tjie* (as in "*spelertjie*") could therefore be seen as inflectional affixes.

The word "*speler*", on the other hand is considered to be the product of derivation, i.e. the creation of a new word (i.e. lemma) through adding a derivational affix to another word/lemma/stem. In the case of "*speler*", the suffix *-er* was attached to the lemma "*speel*" 'play'. "*Speler*" is therefore entered into the dictionary as a new lemma since it was derived from the lemma "*speel*".

One important aspect of the difference between inflection and derivation is the fact that inflectional processes never change the lexical category (i.e. part-of-speech category) of the word [40]. For instance, the addition of the suffix *-erig* to the lemma "*speel*", represents a change in word type from "*speel*" (verb) to "*spelerig*" 'playful' (adjective), which is considered a derivational process. This can be compared to the addition of the inflectional suffix *-s* (which indicates the plural form) to the lemma "*speler*". In the latter case there is clearly no change in word type from "*speler*" (noun) to "*spelers*" (noun).

From the literature, the differences between inflection and derivation can be summarised as in Table 2 [38,41]:

Inflection	Derivation
Produces word-forms of a single lemma (e.g. "*spelers*" and "*spelertjie*" are both considered to be different word-forms of the lemma "*speler*").	Produces new lemmas (e.g. the addition of the suffix -*erig* to the lemma "*speel*" forms the new lemma "*spelerig*").
Class maintaining (e.g. "*speler*", "*spelers*" and "*spelertjie*" are all nouns).	Class changing as well as class maintaining (e.g. "*speel*" is transformed from a verb to an adjective through the addition of the suffix -*erig* to form "*spelerig*").
Marks agreement (e.g. the derivational -*s* in the word "*sits*" marks agreement to the subject in the sentence: "*The man sits behind his desk.*").	Does not mark agreement (e.g. the word "*badly*" does not mark agreement in the sentence: "*The relationship ended badly.*").
Semantically regular (e.g. the addition of the inflectional suffix -*e* to the end of words like "*mense*" 'humans', "*diere*" 'animals' and "*honde*" 'dogs' always means 'more than one').	Semantically irregular (e.g. the derivational suffix -*lik* added to the end of words like "*vertroulik*" 'confidential' and "*menslik*" 'human-like' does not always have the same meaning. In the case of "*vertroulik*", the -*lik* means '**with** confidence' and in "*menslik*" it means '**like** a human').
Cannot be replaced by a simple root form in a sentence (e.g. the word "*gesit*" 'sat' cannot be replaced by the simple root form "*sit*" 'sit' in the sentence: "*Hy het op die stoel gesit.*" 'He sat on the chair.').	May be replaced by a simple root form in a sentence (e.g. The word "*aanskoulik*" 'spectacular' can be replaced by the simple root form "*mooi*" 'beautiful' in the sentence: "*Die vertoning was aanskoulik.*" 'The show was spectacular.').
Usually marked further from the root than derivation (e.g. the word "*spelertjie*" contains the inflectional suffix -*tjie*, as well as the derivational suffix -*er*. The derivational suffix is occurs first after the root "*speel*" than the inflectional suffix).	Usually marked further from the root than inflection.
Involves few variables in a closed system (e.g. there are a limited number of affixes that falls into the inflectional categories).	May involve many variables in an open system (e.g. there is a large number of affixes that can be categorised as derivational affixes).

Table 2: Differences between inflection and derivation

Tuggy [42] states that there is no absolute binary distinction between inflection and derivation as depicted in Figure 7. Instead, he claims that this distinction is better viewed as gradual or scalar among several parameters. In other words, instead of the schema presented in Figure 7, Figure 8 would be a more accurate presentation, with the possibility of an affix

falling in between the two poles of the scale. From this viewpoint, an affix cannot be classified as clearly inflectional or derivational; instead some affixes lean towards the inflectional side and others towards the derivational side of the scale.

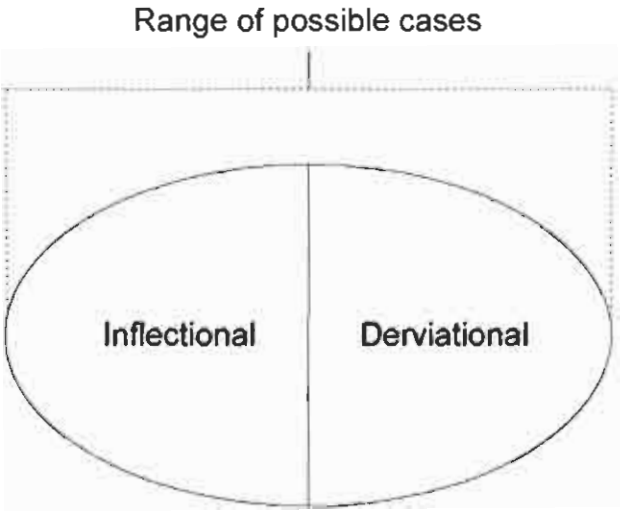

Figure 7: Model of the distinction between inflection and derivation

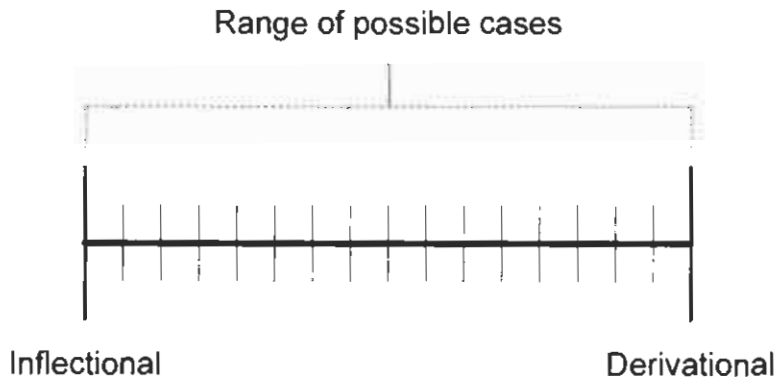

Figure 8: Alternative model of the distinction between inflection and derivation

The claim of Tuggy [42] concerning the gradual distinction between inflection and derivation might be satisfactory from a linguistic point of view. However, this claim is somewhat problematic in the case of automatic lemmatisation where a clear distinction between inflection and derivation is required. The goal of the following section subsequently is to provide this "clear" distinction between inflection and derivation for Afrikaans.

3.3 Opposing Viewpoints on Inflection in Afrikaans

The few publications available on the subject of inflection and derivation in Afrikaans do not always agree on the boundaries between these two categories. Some linguists claim that the distinction is necessary, while others reject it altogether. The root of this problem is that most authors base their criteria on the grammatical structures of highly inflected languages such as classic Greek and Latin, while Afrikaans is actually more an analytic language [43].

One of the authors who maintain the distinction is De Villiers [40], who states that inflectional affixes have the following characteristics:

i. They have clear meaning or value;
ii. They do not change the word type; and
iii. They exclude the possibility of affixture, meaning that inflection is nearer to the stem than derivation for Afrikaans (see Table 2).

According to De Villiers' criteria, the inflectional affixes indicating the plural form (e.g. the *-s* in "*spelers*" 'players') and degrees of comparison (e.g. the *-er* in "*mooier*" 'more beautiful', or the *-ste* in "*mooiste*" 'most beautiful') are the only inflectional affixes in Afrikaans.

Other Afrikaans linguists who also recognise inflectional affixes in Afrikaans are De Klerk [44], Van der Walt, Van Aardt and Eksteen [45], and Posthumus [46,47]. All these linguists, including De Villiers [40] define the following inflectional affixes for Afrikaans:

i. The plural *-e* (as in "*boeke*" 'books');
ii. The plural *-s* (as in "*heuwels*" 'hills');
iii. The comparative *-er* (as in "*gevoeliger*" 'more sensitive'); and
iv. The superlative *-ste* (as in "*hoogste*" 'highest').

All of the authors, except for De Villiers [40], also define the following prefix as an inflectional affix:

v. The past tense *ge-* (as in "*geloop*" 'walked').

In addition, De Klerk [44] and Van der Walt, Van Aardt and Eksteen [45] define another inflectional affix, namely:

vi. The attributive *-e* (as in "*pragtige*" 'beautiful').

Furthermore, the partitive genitive *-s* is also defined as an inflectional affix by Posthumus [46], Jenkinson [48] and Du Toit [49]:

vii. The partitive genitive *-s* (as in "*iets moois*" 'something beautiful').

The diminutive is also recognised as an inflectional category by De Klerk [44], Posthumus [50], Jenkinson [51], Van Heerden [52] and Van der Merwe [53]:

viii. The diminutive affixes (e.g. the *-jie* as in "*hondjie*" 'puppy', or *-pie* in "*boompie*" 'small tree').

In reaction to this, Combrink, wrote an article in 1974 [54] in which he contradicted the views of these authors altogether. He even went so far as to say:

> *The term inflection, as used until recently in the traditional and structural descriptions of Afrikaans, is nothing but a useless Latinism. The distinctions between so-called derived and inflected forms do not cover the facts of Afrikaans, and besides this it does not have any further application that is illuminating or even interesting.*" [Translated-HJG]

Combrink still maintains this view on inflection in 1990, because he describes the distinction between inflection and derivation as ungrounded [43]. In reaction to this, Jenkinson [38] however states unequivocally that Afrikaans is a language that preserves a few of the canonical inflectional categories. He says that it seems as if a clear distinction between inflection and derivation for Afrikaans is not possible. This view is similar to Tuggy's [42],

who describes the distinction as being of a gradual nature. Jenkinson [38] bases this view on the fact that language cannot be viewed as a formal logic system that always adheres to rules.

3.4 INFLECTIONAL AFFIXES FOR LEMMATISATION

Despite the radical views of Combrink, and because of the fact that there is no consensus about the inflectional affixes in Afrikaans (which is clear from the above overview), we nevertheless accept all of the inflectional categories presented by the above-mentioned linguists for purposes of automatic lemmatisation. We also include the past participle and infinitive, as they are inflectional categories for Dutch [55]. We have to define the inflectional categories in Afrikaans explicitly, as this is required for the construction of an automatic lemmatiser (the lemmatiser has to be trained to recognise specific affixes as inflectional). These inflectional categories[3] are:

i. Plural (e.g. the *-s* in "*tafels*", 'tables' and the *-e* in "*mense*", 'humans');
ii. Degrees of comparison (e.g. the *-er* and *-ste* in "*kleiner*" 'smaller' and "*kleinste*" 'smallest');
iii. Diminutive (e.g. the *-jie* in "*hondjie*" 'puppy');
iv. Past Tense (e.g. the *ge-* in "*geloop*" 'walked' and the -ge- in "*uitgeloop*" 'walked out');
v. Past Participle (e.g. the *ge- -te* in "*getrapte*" 'trampled');
vi. Infinitive (e.g. the *-e* in "*drinke*" 'drink');
vii. Attributive (e.g. the *-e* in "*pragtige*" 'exquisite'); and
viii. Partitive Genitive (e.g. the *-s* in "*pragtigs*" 'exquisite').

Any lemmatiser for Afrikaans (including *Lia*) should be able to remove all affixes in these eight inflectional categories, yielding linguistically correct lemmas. These affixes are used as the basis for defining the classes for Afrikaans lemmatisation in Section 3.5. Although it seems easy, Afrikaans lemmatisation proves to be no trivial task: it entails more than just removing the correct affix from the word to obtain the correct lemma. For instance, *Lia* has to deal with a number of complexities, such as ambiguity of affixes, ambifixes, morphonological processes and exceptions.

[3] A list of all the affixes belonging to the defined categories is provided in Addendum B

i. Ambiguity of affixes

A rule-based lemmatiser will tend to remove the suffixes *-tjie* and *-jie* erroneously in the case of words like "*jobskraaltjie*" ('a grass species') and "*suurpootjie*" ('a tortoise specie'), since these suffixes normally mark the diminutive form. However, in these words the *-tjie* and *-jie* do not indicate the diminutive form, as it forms part of the lemma of the word. This is quite a common phenomenon (i.e. the ambiguity of affixes) across all the inflectional categories identified above.

ii. Ambifixes *ge-...-t/-de*

Words that are in the past participle form (like "*ingedraaide*" 'screwed in') should be lemmatised as "*indraai*" 'screw in'. However, words that are in the past participle form that start with *onge-* are not lemmatised according to the manner that other past participles are lemmatised. Instead, only the suffixes *-de* or *-te* should be removed during lemmatisation. "*Ongenooide*" 'uninvited' must accordingly be lemmatised as "*ongenooi*", instead of the invalid lemma "**onnooi*".

iii. Morphonological processes

Due to morphonological processes, words like "*paaie*" 'roads' are not lemmatised by just removing the *-e* that indicates the plural form; a *-d* should also be appended at the end of the word during the transformation to the lemma. More examples are "*stede*" 'cities' (lemma: "*stad*" 'city') and "*oë*" 'eyes' (lemma: "oog" 'eye').

ix. Exceptions

Lia also has to deal with a number of exceptions, such as the following:

i. Words like "*domme*" 'stupid' as in the "*domme swaap*" 'stupid nitwit' must be lemmatised to "*dom*", because of the attributive *-e* at the end. "*Dommes*" 'stupid

people' on the other hand should not be lemmatised as "*dom*", but rather as "*domme*" 'stupid person'.

ii. Different variants of words must be retained. The word "*afgrawe*" 'dig down' for example is a variant of the lemma "*afgraaf*"; hence should "*afgrawe*" not be lemmatised.

iii. Loan words (from other languages) that are included in dictionaries for Afrikaans are not allowed to be removed from the training data. Examples of such words are "*chlorium*" (Latin) 'chlorium', "*ghnoe*" (Khoi) 'steenbok (an antelope species)' and "*akrostigon*" (Classic Greek) 'acrostic'. The inclusion of these words is problematic as they do not conform to the grammatical structure of Afrikaans words.

iv. Words like "*juweliersware*" 'jewellery', "*skoolure*" 'school hours' and "*handelsgoedere*" 'trade goods' do not have a singular form. The lemmas should remain "*juweliersware*", "*skoolure*" and "*handelsgoedere*".

v. The word "*bloes*" 'blouse' has a variant "*bloese*" and also a plural form "*bloese*" 'blouses'. In these cases the word must be lemmatised to the simplest form that is "*bloes*".

The above mentioned complexities and exceptions will be encountered by any lemmatiser for Afrikaans. All of the words represented by these complexities and exceptions will be extremely difficult to lemmatise with a rule-based lemmatiser, unless they are included in a separate exception list, or if the lemmatiser employs some form of dictionary lookup. It is assumed here that *Lia* will be better equipped to lemmatise these words than a rule-based lemmatiser. MBL essentially has its own builtin exception list. If many exceptions occur, MBL will be able to handle this well. This study aims to prove that the MBL algorithms that *Lia* is based on are much more suited to modelling the complexities and exceptions of lemmatisation in Afrikaans than any set of expert rules (see the strengths of MBL algorithms in Section 2.4.3).

3.5 Defining The Classes

Defining the format of the classes forms an important part of the data-construction phase, as will be presented in this section and in Chapter 4. The logical way to go about the problem is

to use grammatically motivated classes, as introduced in Section 3.4. For instance, the grammatically motivated class of the word "*maatjie*" 'little friend' (lemma: "*maat*" 'friend') is *-jie,* implying that the suffix *-jie* should be removed from the word during lemmatisation. However, lemmatisation of Afrikaans words does not always entail the removal of affixes only: in some cases it also involves replacing the removed affix with some string of characters to accommodate morphonological changes (of 3.4 iii). Consider for example the word "*slote*" 'ditches' where the linguistically correct class is the *-e* that indicates the plural form. Removing the *-e* from "*slote*" produces the word "*slot*" 'lock', which is coincidentally a valid Afrikaans word, but not the correct lemma (i.e. "*sloot*" 'ditch'). The lemmatisation of "*slote*" therefore actually involves the removal of the *-e* and the insertion of the letter *o*.

A possible solution to overcome this problem computationally (i.e. without expert knowledge), is to use Levenshtein distance to map the invalid lemma to the correct one. Levensbtein distance is the edit distance between two strings, and is determined by calculating the minimum number of operations (insertion, deletion and substitution) necessary to transform one string into another [56]. This method entails the computation of the Levenshtein distance between the invalid lemma and every word in an Afrikaans lemma list, which we suspect to be a computationally expensive process. More importantly, we also expect this approach will not at all be error free, for example consider the case of "*balletjie*" 'small ball' where the linguistically correct class will be the suffix *-tjie* that indicates the diminutive form. If we remove the suffix *-tjie* we are left with the invalid lemma "**balle*". The use of Levenshtein distance will subsequently indicate the lemma as "*ballet*" 'ballet' instead of the correct lemma "*bal*" 'ball'. The reason for this problem is that the Levenshtein distance between "*balle*" and "*ballet*" is shorter than the distance between "*balle*" and "*bal*". The use of grammatically motivated classes combined with Levenshtein distance is therefore problematic in cases such as "*balletjie*", where grammatically motivated classes do not provide adequate information for generating the lemma from the inflected form of the word.

Another possible solution, and the one we choose to implement, is to use non-grammatically motivated classes. These non-grammatically motivated classes are automatically assigned on tbe basis of a comparison between the word and its correct lemma by means of a Perl script. The classes are derived by determining the character string (and the position thereof) to be

removed and the possible replacement string during the transformation from word-form to lemma by comparing the word-form and its expected lemma. The positions of the character string to be removed are annotated as *L* (left), *R* (right) and *M* (middle). If a word-form and its lemma are identical (i.e. negative data), the class awarded will be "*0*", denoting the word should be left in the same form. This annotation scheme yields classes like in the third column of Table 3.

Extracted Word-Form	Manually Identified Lemma	Automatically Derived Class
"*Geel*" 'yellow'	"*Geel*" 'yellow'	*0*
"*Geslaap*" 'slept'	"*Slaap*" 'slept'	*Lge>*
"*Hondjie*" 'puppy'	"*Hond*" 'dog'	*Rjie>*
"*Bote*" 'ships'	"*Boot*" 'ship'	*Rte>ot*
"*Omgedraaide*" 'turned-over'	"*Omdraai*" 'turn-over'	*MgeRde>*

Table 3: Data preparation for Lia

Thus, the class of "*geslaap*" 'slept' will be *Lge>*, where the *L* implies that the inflectional prefix *ge-* should be removed on the left-hand side of the word in order to lemmatise it. Accordingly, the class of the word "*bote*" 'boats' will be *Rte>ot*, denoting the *-te* at the right-hand side of the word should be replaced by *-ot*. Words in the past participle form, for instance "*omgedraaide*" 'turned over', will receive the class *MgeRde>*, meaning that the *-ge-* and the *-de* should be removed at the middle and at the right-hand side of the word respectively.

This annotation scheme is still practical in the case when the same substring occurs more than once in a word. For example the word "*oorgegee*" 'surrendered' has class *Mge>*, which implies that the string *-ge-* should be removed from the middle of the word. The problem is that there are more than one place in the word where the string *-ge-* can be removed. This seemingly ambiguity is resolved by removing the first occurence of the string *-ge-* in the word, since the inflectional prefix that needs to be removed always physically occurs at the beginning of the word before the lemma.

This method of class assignment eliminates the generation of incorrect lemmas like "**beeldskon*" or "**ball*" and can therefore be viewed as a suitable alternative to

grammatically motivated classes. The negative aspect of using non-grammatically motivated classes is that it enlarges the number of possible classes from 74 (when using grammatically motivated classes) to 271, which complicates the lemmatisation process even further (larger numbers of classes increase the number of possible classifications for every instance in the training data, consequently increasing the probability of a wrong classification). The distribution of the most frequent classes in the training data is presented in Figure 9. A list containing the frequencies of all the classes in the training data is provided in Addendum D.

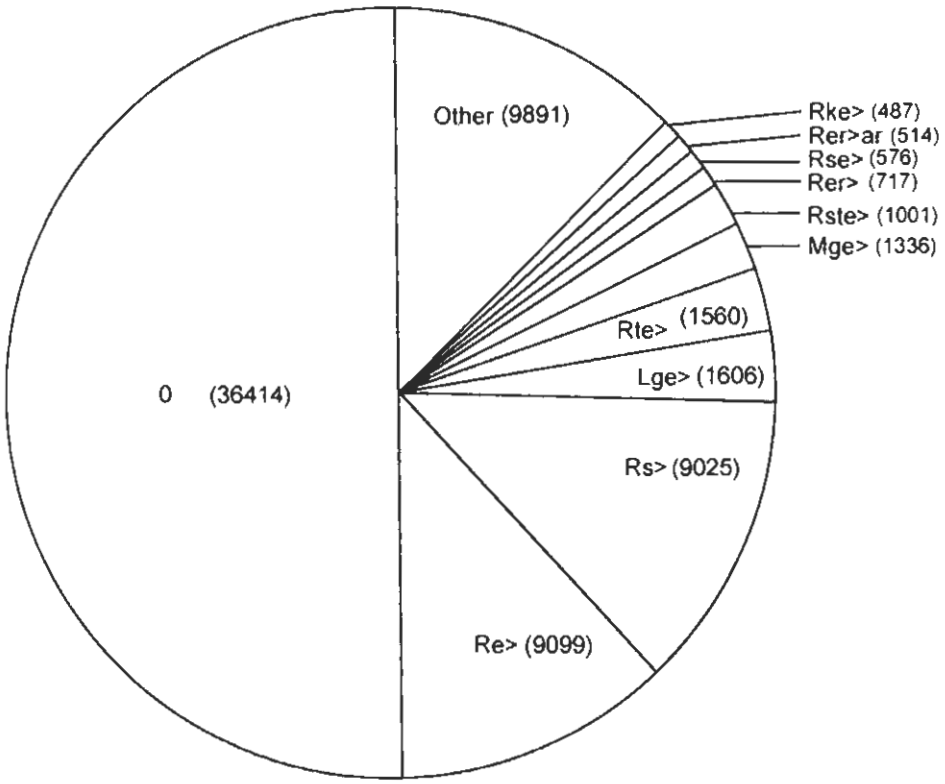

Figure 9: Frequency of the classes

3.6 CONCLUSION

The purpose of this chapter was to define the classes for automatic lemmatisation for Afrikaans. We started the chapter by providing the background information required for answering Research Question 1. We emphasised the fact that Afrikaans is a morphologically productive language, with both inflectional and derivational processes. It was further pointed out that automatic lemmatisation for any language requires a clear distinction between inflection and derivation and that such a clear distinction does unfortunately not exist for

Afrikaans. In order to make an informed distinction between inflection and derivation we presented information on the processes of inflection and derivation, with specific reference to the differences between the two processes. A short overview of the opposing viewpoints of various Afrikaans linguists was presented to aid the task of distinguishing between the categories of inflectional and derivational morphemes. As there is no consensus among linguists on the inflectional categories for Afrikaans, we proceeded to accept all the inflectional categories mentioned by Afrikaans linguists. We also pointed out the complexities and exceptions that form an inherent part of the Afrikaans lemmatisation task, and which should be handled effectively by any lemmatiser for Afrikaans. The chapter concluded with a section where we directly answered Research Question 1, by defining the classes for Afrikaans lemmatisation. We explained the process of automatic class assignment and provided a distribution chart of the classes in *Lia's* training data. A comprehensive list of all classes is provided in Addendum D.

Chapter 4: DATA FOR LEMMATISATION IN AFRIKAANS

4.1 INTRODUCTION

In Section 1.2 it has been argued that the accuracy of *Lia* is partially dependent on the format of the classes and various facets of training data. Where Chapter 3 defined the classes for Afrikaans lemmatisation, the focus of this chapter is on the various aspects of *Lia's* training data. Chapter 4 therefore directly addresses **Research Question 2** (i.e. **what is the influence of the various feature options on the performance of *Lia*?).**

This chapter commences with an explanation of the data generation process in *Lia*, where we describe the processes of data extraction and annotation. The quality control process is introduced in Section 4.2.3, where we present alternative ways to manual checking for locating errors in the data. The influence of the size of the data set on the accuracy of *Lia* is presented in Section 4.3., which is followed by an overview of the various available feature options and their influence on the performance of *Lia* when trained with the default parameter settings of TiMBL. The feature options relate to the number of features, feature alignment and that include syllables, number of letters contained in the syllables and probabilities based on the last letter and the affixes contained within the word. The chapter concludes with a graphic visualisation of the nearest neighbour relations in an extraction of 1 000 instances from the training set.

4.2 DATA GENERATION

At the start of this project, we did not have training data available in the desired format for automatic lemmatisation. This can be attributed to the fact that Afrikaans is a resource-scarce language [57], especially if it is compared to a language such as Dutch that has various resources (such as annotated and aligned corpora) available. We therefore had to generate our own training data by extracting words from existing resources, and annotating the extracted words by providing the linguistically accurate lemma for each word. The classes were then

derived from this annotated data and appended to the training data by means of a Perl script. The aim of this section is to provide an overview of the process of generating training data for *Lia*

4.2.1 EXTRACTION

The training data for this project was extracted from CTexT's Afrikaans lexicon [19], which consists of circa 350 000 words. The extraction was done by means of a Perl script that performs string matching with regular expressions to extract words that contain strings corresponding to the inflectional categories defined for purposes of this study (see Section 3.4). All the words that correspond in form to the inflectional affixes defined in Chapter 3 were extracted. For example, both the words "*geel*" 'yellow' and "*geslaap*" 'slept' were extracted during this process, because both words begin with the string *ge-*. Note that the lemma of "*geslaap*" is "*slaap*" 'sleep', but the word "*geel*" 'yellow' is already a lemma. However, it is important to also train *Lia* with lemmas such as "*geel*" 'yellow', since *Lia* should not only learn **how** to lemmatise, but also **when** to lemmatise words and when not to (see below). This extraction yielded 104 420 words, which form a subset of words from which more instances can be extracted and annotated to serve as training or evaluation data. This subset was divided into groups of 1 000 words each to facilitate the annotation process.

It would not have been sensible to annotate and train with all of the available 104 420 words in the subset, because we want to construct a lemmatiser that achieves an accuracy figure of 90% (see Section 1.5.4), while being trained with the lowest possible number of training instances. The annotation of training data, especially for a resource-scarce language such as Afrikaans, is an expensive, time consuming and labour intensive process. This provides the motivation for trying to use as small a training set as possible to achieve the desired accuracy of 90%.

After the annotation process (see Section 4.2.2), a further 13 000 words were extracted from the remaining part of the lexicon (i.e. the part of the lexicon that was left after the 104 420 words had been extracted) with the purpose of enlarging the number of negative instances (i.e. words that should not be lemmatised) in the training data. This extraction was done at the end of the data construction phase, because it was not necessary to annotate these words,

since they contain no inflectional affixes. This data was therefore not part of the data that was presented to the assistants for annotation. We decided on 13 000 words, since this increased the percentage of negative instances in the training data to 50%. The motivation behind this decision was to prevent the lemmatiser from being over-eager (i.e. removing word parts that are not inflectional by nature, or lemmatising words that should not be lemmatised at all) during the lemmatisation process. A lemmatiser trained with no negative instances can be expected to be over-eager, as it will always try to classify (or lemmatise) words that are already lemmas. We also increased the number of negative instances in the training data, since we believe that a large number of the words that *Lia* will encounter in free text will be uninflected word-forms.

4.2.2 ANNOTATION

The above-mentioned groups of 1 000 words each were provided to the annotators in the form of spreadsheets. The words were placed in the first column of the spreadsheets and the annotation of the training data entailed providing the linguistically correct lemma for every word in the second column. This seems like a relatively easy task, but as was pointed out in Chapter 3, lemmatisation for Afrikaans is quite a complex task with many exceptions. The annotation was performed by various research assistants. A technical report (see Addendum A), with guidelines for performing manual Afrikaans lemmatisation, was therefore composed to assist the research assistants in the annotation process. We did not measure inter-annotator agreement, but alternative methods of quality control was used (see the next section).

We followed a recursive (i.e. bootstrapping) process in annotating the training data. This recursive process entails using a first prototype of *Lia* to help annotate her own training data after the first 5 000 instances in the training data were annotated by hand. This prototype of *Lia* is then used to classify more training instances in batches of 2 000 words each, which then only need to be manually checked for errors. This saves time and resources since the annotators do not need to annotate the training instances, but rather only manually check each batch of training data, before it is then added to the existing training data to serve as training data for the next consecutive batch. This bootstrapping process is continued until the desired

linguistic accuracy figure of 90% is reached. We ensured that the minimum amount of training data deemed necessary for obtaining the desired linguistic accuracy was annotated. Research assistants were not required to manually provide the classes during the annotation process; the classes were assigned automatically by means of a Perl script (see Section 3.5). The 13 000 additionally extracted words were all assigned class 0, as it comprises of negative data only.

4.2.3 Quality Control

Quality control was carried out on the data before it was used for training and evaluation purposes. The quality control process consisted of every word being manually checked for lemmatisation errors. Even though the quality control was carried out meticulously, there were still errors in the data. The elimination of these errors is very important to ensure an accurate lemmatiser, because errors in the training data have a negative influence on the accuracy of the system.

There are basically two alternative ways other than manual checking to locate the errors in the data. The first is to perform a frequency count of the classes in the data to determine the unique classes, since unique classes in the data could possibly indicate spelling and/or lemmatisation errors. There are however quite a large number of valid unique classes in the data (63 unique classes from the total number of 271 classes, or 23%), for example the class *Rmmie*> that belongs to the word "*mammie*" 'mother'. The second way to locate errors is to consider the instances that were incorrectly classified by *Lia* and to determine why the incorrect classification was made. In every error case we had to determine if the incorrect classification could be ascribed to the percentage of errors that *Lia* made during classification or whether it could be ascribed to errors in the training data. In the latter case, the errors were then corrected to improve the quality of the training data.

This section described the various phases of the process of generating training data for *Lia*, namely data extraction, annotation and the quality control process. As previously mentioned, our goal is to achieve an accuracy figure of 90% by training *Lia* with the smallest possible data set, since data annotation is such a resource-expensive process. Sections 4.3 emphasises

the importance of training the system with large amounts of data (i.e. data sets larger than 10 000 instances) by illustrating the increase in accuracy that can be achieved when training *Lia* with increasing amounts of training data.

4.3 DATA SIZE

In Section 2.3. it was mentioned that machine learning systems improve with experience [20]. In the case of *Lia*, the experience is based on the amount of data used during training, where the assumption is that *Lia's* accuracy will improve as more data is used for training. The improvement of *Lia's* accuracy figure with increasing amounts of training data is presented in Figure 10. The training data for the preliminary experiments carried out in this chapter is right-aligned data (see Section 4.4.3) consisting of 72,226 instances with 38 features (see Section 4.4.2) each.

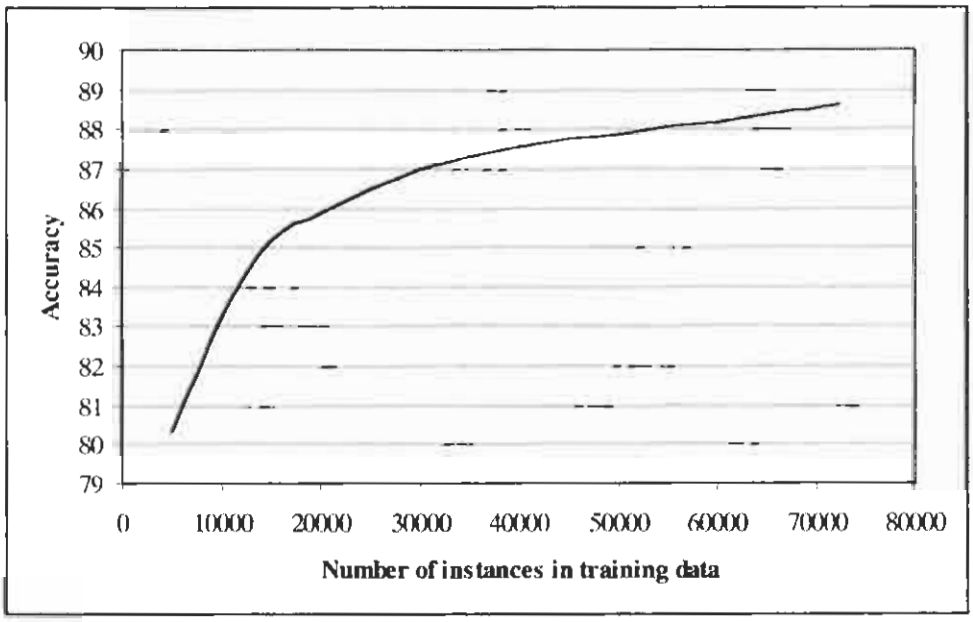

Figure 10: Improvement in accuracy with increasing amounts of training data

Figure 10 indicates that *Lia* achieves a relatively high accuracy figure when using a small training set consisting of only 5 000 words (i.e. 80,33%). The graph has a steep gradient from 5 000 instances to 20 000 instances, which indicates that a relatively large increase in accuracy is achieved by enlarging the number of training instances from 5 000 to 20 000. The gradient decreases from 20 000 instances onwards, indicating that it is increasingly difficult

to achieve further improvement in accuracy by increasing the number of training instances. The graph displayed in Figure 10 illustrates the results achieved after training *Lia* on right-aligned data with 20 features (see 4.4.2), and training the classifier with the default TiMBL settings[4]. Figure 10 shows that despite the fact that *Lia* was trained with a data set consisting of 72 226 instances, the desired accuracy figure of 90% was not achieved.

It is interesting to note the increase in execution time as the amount of training data is increased.

Figure 11 shows that a linear relationship exists between the number of instances in the training data and the execution time. This provides further motivation for trying to use as small a training set as possible, as training with more instances (i.e. larger data sets) increases the execution time.

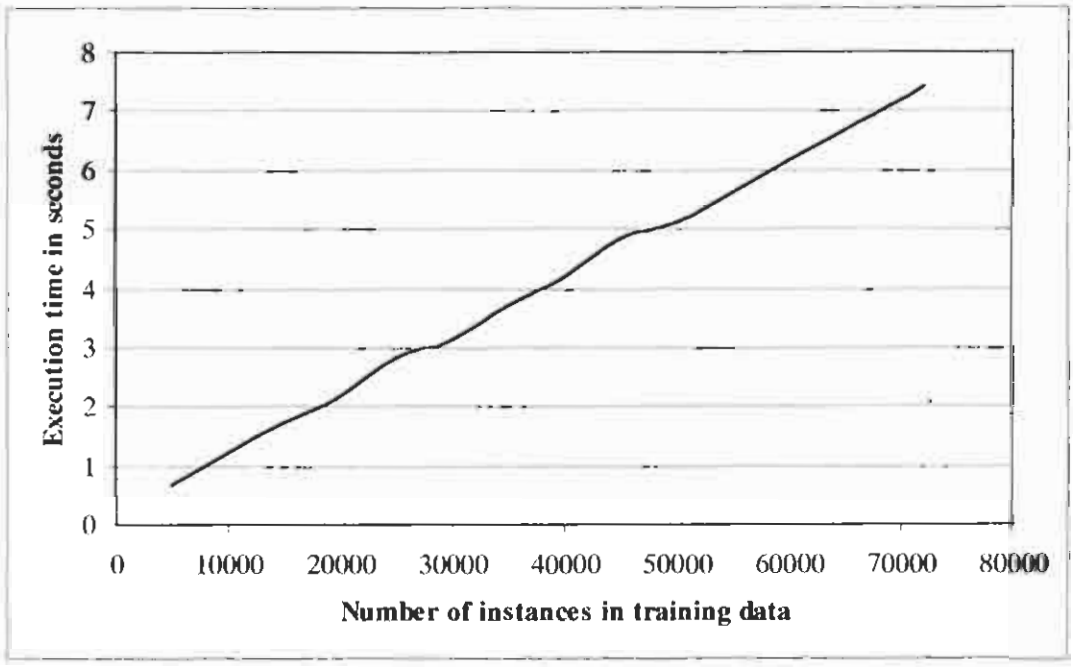

Figure 11: Increase in execution time with increased amounts of training data

Increasing the amount of training data does not only affect the accuracy and classification time of the system, but also affects the memory usage of the system. Figure 12 indicates that a linear relationship also exists between memory usage and the amount of training instances

[4] IB1 with Gain Ratio feature weighting and k=1

used for training. This linear relationship can be attributed to the fact that training *Lia* with more data will result in more data being stored in memory.

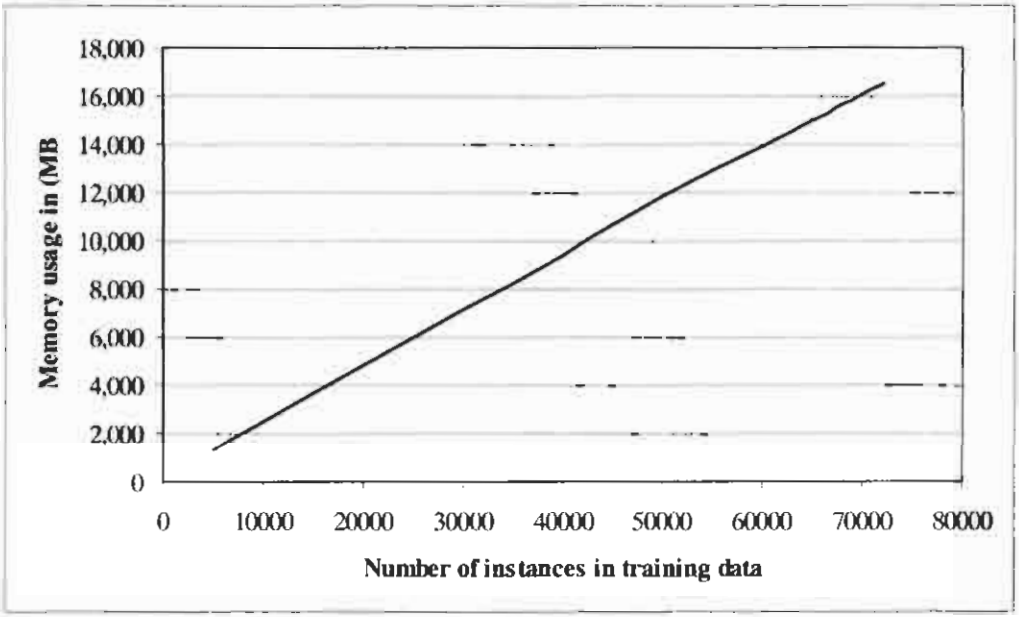

Figure 12: Increase in memory usage with increased amounts of training data

It is very interesting to make a prediction regarding the amount of training data that will be required to reach the desired accuracy figure of 90%. Figure 13 shows a logarithmic trend line that was automatically inserted to make such a prediction. The reliability of the logarithmic trend line is indicated by the R^2 value (the nearer it is to 1, the more reliable it is). The trend line in Figure 13 has a R^2 value of 0,9579, which is a relatively good fit to the line of the data. Figure 13 shows that an accuracy figure of 90% will be reached when *Lia* is trained with training data consisting of about 100 000 instances. Since we do not have access to training data consisting of 100 000 instances (or the resources to annotate such a large amount of training data), we are forced to investigate alternative methods (i.e. experimenting with different feature options or adjusting the parameter settings) in order to achieve an accuracy figure of 90%.

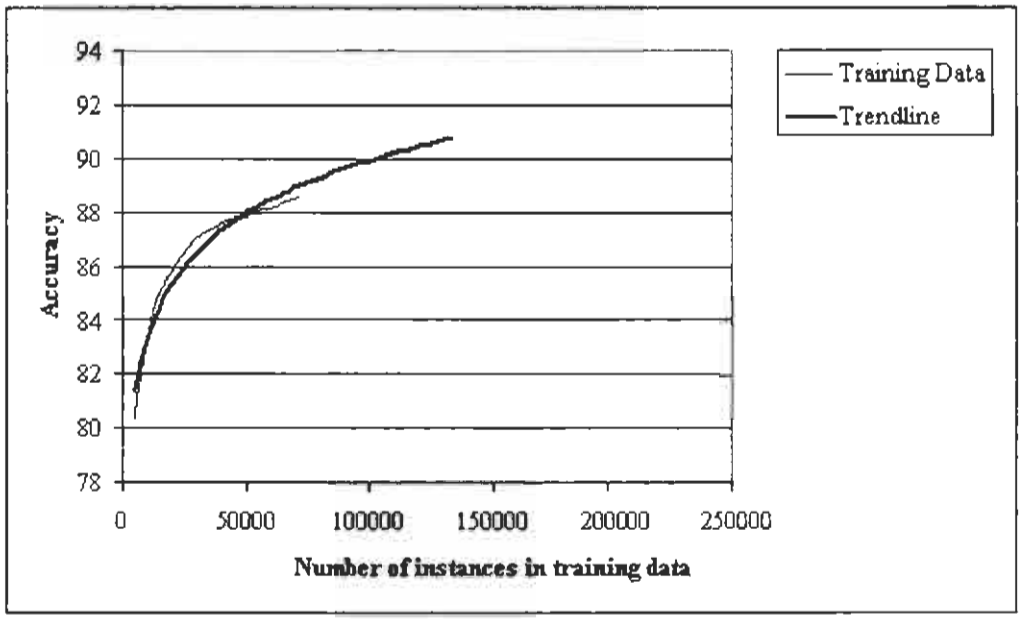

Figure 13: Forecast of the number of training instances required to achieve 90% accuracy

4.4 VARIOUS FEATURE OPTIONS

4.4.1 INTRODUCTION

The previous section indicated that it should be possible to obtain an accuracy figure of 90% when training *Lia* with a data set consisting of 100 000 instances. Since we do not have so much training data available, the focus of this section is on experimenting with various feature options that could possibly improve the accuracy figure of *Lia* without training with more data. For experimentation with training data size we have only considered using right-aligned training data with 20 letters combined with underscores (for empty characters) as features. We are however not restricted to only using 20 letters as features; we can experiment with different numbers of features, as well as with feature positioning options other than right-alignment. We can also extend the training data with a number of additional features that might have a positive effect on the lemmatisation accuracy of *Lia*. The reasoning behind experimenting with different numbers of features, feature positioning and the inclusion of additional features was that we hoped to improve the accuracy of the system by providing more information about the words that we wanted to lemmatise.

This subsection commences by considering the number of features to be included in the training data. We will then investigate the alignment and positioning of features. This is followed by experiments with a few additional features, viz.:

- Syllables;
- Number of letters contained in the syllables;
- Lemmatisation probability based on the last letter of the word; and
- Lemmatisation probability based on the morphemes contained within a word.

The aim of this subsection is not to draw any final conclusions regarding the performance of the system with various data representations, since we are experimenting here only with the default TiMBL algorithm and parameter settings. There are many more algorithmic parameter combinations available in TiMBL that may yield different results than the default algorithmic settings, which will be further investigated in Chapter 5. Instead, the purpose of this subsection is to introduce the different data representations and to get a preliminary idea of their influence on the performance of the system; more conclusive results will be presented in Chapter 5.

4.4.2 Number of Features

The first data set generated for the development of *Lia* consisted of 5 000 training instances, where every instance consisted of 38 features. Since less than 0,02% of the words in CTexT's Afrikaans lexicon [19] contains more than 38 letters, we decided to specifically use 38 features to ensure that no information is lost during the generation of the training data. The problem with using 38 features is that 54,3% of the words in the lexicon consist of 10 or less letters. This leads to data that contains more underscores (i.e. empty feature-positions) than letters. The danger of having more underscores than letters in the data is that the system then classifies evaluation instances on the basis of underscores instead of letters. Execution time and memory usage are also expected to be directly related to the number of features used; it will therefore always be better to use as few features as possible.

The problems experienced with 38 features prompted us to do further experimentation with data sets having fewer than 38 features with the aim to improve the overall performance of the system in terms of linguistic accuracy, execution time and memory usage. We therefore included additional data sets for evaluation, consisting of 13, 20 and 30 features. We opted for a minimum of 13 features since this is somewhat higher than the average word length of 11,53 characters in the Afrikaans lexicon. We arbitrarily included features sizes of 20 and 30, because 20 and 30 are round numbers between 13 and 38.

The additional data sets containing 13, 20 and 30 features were not generated by simply removing the number of redundant features at the beginning of the word. Instead, we considered the class of the involved word to determine where to remove the redundant features. Words starting with the string *ge-* were for example shortened by removing features at the end of the word, while the features of words ending in a possible inflectional suffix were reduced by removing letters at the beginning of the word. The features of words like "*omgedraaide*" 'turned-over' where *-ge-* is inserted in a separable verb were reduced by removing the letters in the middle of the word between *omge-* and *-de*.

The features were reduced in this manner to preserve the inherent grammatical structure of the data, which is required for effective lemmatisation. We did not experiment with data having fewer than 13 features, because it was assumed that too much important information about the grammatical structure of the data would have been lost through this. Figure 14 illustrates the different representations of the data.

Table 4 shows a comparison based on average accuracy, average execution time and average memory usage for different numbers of features. We could also have experimented with other feature sizes larger than 38, but this is considered to be unnecessary since Table 4 indicates that no significant increase in accuracy is achieved by using larger numbers of features.

13 Features

v,e,r,n,i,e,t,i,g,b,a,r,e,Rre>ar
g,e,_,l,d,p,a,r,i,t,e,i,t,0
o,e,r,s,v,e,r,s,k,i,l,l,e,Rle>
a,a,n,_,g,e,_,m,a,a,k,t,e,Mge>Rte>
_,_,_,_,_,_,_,o,p,s,l,a,e,0
r,s,i,e,n,i,n,g,s,r,a,d,e,Rde>ad

20 Features

_,_,_,_,_,_,_,v,e,r,n,i,e,t,i,g,b,a,r,e,Rre>ar
g,e,_,_,_,_,_,_,_,_,l,d,p,a,r,i,t,e,i,t,0
_,_,_,_,_,_,k,o,e,r,s,v,e,r,s,k,i,l,l,e,Rle>
a,a,n,_,_,_,_,_,g,e,_,_,_,_,m,a,a,k,t,e,Mge>Rte>
_,_,_,_,_,_,_,_,_,_,_,_,_,_,o,p,s,l,a,e,0
h,e,i,d,s,h,e,r,s,i,e,n,i,n,g,s,r,a,d,e,Rde>ad

30 Features

,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,v,e,r,n,i,e,t,i,g,b,a,r,e,Rre>ar
g,e,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,l,d,p,a,r,i,t,e,i,t,0
_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,k,o,e,r,s,v,e,r,s,k,i,l,l,e,Rle>
a,a,n,_,_,_,_,_,_,_,_,g,e,_,_,_,_,_,_,_,_,m,a,a,k,t,e,Mge>Rte>
_,o,p,s,l,a,e,0
s,t,e,s,g,e,s,o,n,d,h,e,i,d,s,h,e,r,s,i,e,n,i,n,g,s,r,a,d,e,Rde>ad

38 Features

_,v,e,r,n,i,e,t,i,g,b,a,r,e,Rre>ar
g,e,_,l,d,p,a,r,i,t,e,i,t,0
_,k,o,e,r,s,v,e,r,s,k,i,l,l,e,Rle>
a,a,n,_,_,_,_,_,g,e,_,m,a,a,k,t,e,Mge>Rte>
_,o,p,s,l,a,e,0
g,e,_,_,_,_,_,e,s,t,e,s,g,e,s,o,n,d,h,e,i,d,s,h,e,r,s,i,e,n,i,n,g,s,r,a,d,e,Rde>ad

Figure 14: Comparison of training data having different numbers of features

The results of Table 4 were obtained by training *Lia* with the default TiMBL algorithm settings on right-aligned training data. We see from Table 4 that the execution time and memory usage increase as the number of features is enlarged. The maximum average accuracy is obtained when using 38 features, indicating that enlarging the number of features does improve the linguistic accuracy in this case. Linguistic accuracy is the first consideration when we want to decide on a data configuration or algorithmic parameter that delivers the best performance. Execution time and memory usage are considered as secondary to linguistic accuracy, and is only considered when a tie (based on linguistic accuracy) exists between two or more data configurations or algorithmic parameters. Further experiments

where a large number of algorithmic parameters will be considered to determine the number of features that delivers the best performance in terms of the trade-off between accuracy and execution time will be discussed in Chapter 5. Friedman's ANOVA was used here to determine if significant differences exist between the average accuracies reported in Table 4.

Number of Features	Average Accuracy	Average Execution time (in seconds)	Average Memory Usage (in MB)
13	88,574	4,778	9,773
20	88,607	7,391	13,177
30	88,554	9,969	14,395
38	88,622	12,224	14,687

Table 4: Comparison of performance for different numbers of features

The ranks produced by Friedman's ANOVA are displayed in Table 5. The significance as indicated by the ANOVA is 0,003 which is lower than the α-value of 0,05; therefore we can deduce that different numbers of features have a statistically significant effect on the accuracy when training *Lia* with the default TiMBL algorithmic parameters and right-aligned training data (F = 14,051; p<0,05; df = 3). The Wilcoxon Signed-rank test was used to follow up the main analysis and results indicate that significant differences exist between 20 and 30 features (T = 0; r = -0,84; p<0,083); and also between 30 and 38 features (T = 0; r = -0,88; p<0,083). A Bonferroni correction was applied and therefore all the effects are reported at a significance level of 0,083.

Number of features	Mean Rank
13	2,65
20	2,65
30	1,30
38	3,40

Table 5: Ranks for different numbers of features

4.4.3 Feature Positioning

4.4.3.1 Left Alignment

The training data was presented to TiMBL in C4.5 format (see Section 2.5) [36], where every feature of each instance is separated by a comma. For instance, every space between the

commas in Figure 15 represents a feature value (Section 4.4.2 explains why so many features are used). The training data was at first left-aligned as in Figure 15, but this resulted in a very low accuracy figure of only 60,348% (see Table 6).

```
g,e,e,l,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,0
g,e,s,l,a,a,p,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,Lge>
h,o,n,d,j,i,e,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,Rjie>
b,o,t,e,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,Rte>ot
o,m,g,e,d,r,a,a,i,d,e,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,Mge>Rde>
```

Figure 15: Left-aligned training data

4.4.3.2 RIGHT ALIGNMENT

We realised that since the majority of inflectional affixes are suffixes (only one inflectional prefix *ge-* occurs in Afrikaans, which can also be inserted between the preposition and stem in so-called particle verbs), the training data could possibly rather be right-aligned as in Figure 16.

```
_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,g,e,e,l,0
_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,g,e,s,l,a,a,p,Lge>
_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_, h,o,n,d,j,i,e,Rjie>
_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,b,o,t,e,Rte>ot
_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,o,m,g,e,d,r,a,a,i,d,e,Mge>Rde>
```

Figure 16: Right-aligned training data

A remarkable increase from an accuracy figure of 60,348% to 88,607% was achieved by the use of right-aligned data as seen in Table 6. This increase in accuracy can be attributed to the fact that right-alignment ensures that the suffix part of every word is always in the same feature position, which is not the case if the data is left-aligned. Right-alignment also caused a significant increase on the execution time and average memory usage, but these increases are viewed as unimportant when the significant increase in accuracy is considered. The results in Table 6 were once again obtained by training *Lia* on data containing 20 features with the default TiMBL settings.

Alignment	Average Accuracy	Average Execution time (in seconds)	Average Memory Usage (in MB)
Left-alignment	60,348	59,236	9,567
Right-alignment	88,607	7,501	13,211

Table 6: Performance comparison between left-aligned and right-aligned data

The results of a Wilcoxon Signed-rank test indicate that alignment has a significant effect on accuracy. The accuracy obtained with right-aligned data (*Mdn* = 88,585) is significantly higher than that obtained with left-aligned data (*Mdn* = 60,405) ($T = 0$; $p<0,05$; $r = -0,886$).

4.4.3.3 FEATURE ALIGNMENT

The success achieved by right-aligning the training data as seen in the previous section inspired us to conduct further experiments with feature positioning. A common mistake that *Lia* initially made was that the classes of ambifixes (in words like "*geabsorbeerde*" 'absorbed' (class: *Lge>Rde*)) was confused with the classes of prefixes or suffixes (in words like "*gedrink*" 'drank' (class: *Lge>*) or "*verdofde*" 'dimmed' (class: *Rde>*)). The reason for this was that the letters of the inflectional prefix *ge-* were at different feature positions for different instances when the data was right-aligned. The same confusion was experienced with words that were in the past participle form. This problem of inflectional prefixes that are not aligned lead us to define the concept of "feature alignment", in order to reduce the amount of confusion experienced.

Feature alignment implies that all words containing the possible prefix *ge-* are treated like "*geslaap*" 'slept' in Figure 17, or alternatively like "*omgedraaide*" 'turned-over' when *-ge-* is inserted in a past participle verb. Feature alignment ensures that similar features are always aligned at the same feature positions and thereby eliminates any confusion that may arise.

```
g,e,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,e,l,0
g,e,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,s,l,a,a,p,Lge>
_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,h,o,n,d,j,i,c,Rjie>
_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,b,o,t,e,Rte>ot
g,e,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,l,d,w,o,l,f,0
o,m,_,_,_,_,_,_,_,_,_,g,e,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,d,r,a,a,i,d,e,Mge>Rde>
```

Figure 17: Right aligned, feature-aligned data with 38 features

We wanted to ensure that the largest possible part of the word before the string *ge-* is retained, because the preservation of the grammatical structure of the word is very important during the classification process. The position of the prefix *ge-* was therefore determined by the amount of features in the training data. The *ge-* prefix was inserted in feature positions 13 and 14 when feature-aligned data with 38 and 30 features were used, positions 9 and 10 for data with 20 features and positions 5 and 6 for 13 features.

The accuracy gained by the use of feature-aligned data is presented Table 7. The results in Table 7 indicate an increase in accuracy from 88,607% to 91,197% when using feature-aligned data. This increase in accuracy may seem small at first, but the significance of the increase in accuracy is only realised when the relative difficulty of improving the accuracy figure when nearing 90% is considered (see Figure 10). This result is highly significant for this study as a whole since it indicates that an accuracy figure of 90% can indeed be achieved without using a large data set consisting of 100 000 instances as predicted in Section 4.3. The use of feature-aligned data however has a negative effect on the execution time and memory usage. These small increases in execution time and memory usage are once again justified when the significant gain in accuracy when using feature-aligned data is considered.

Alignment	Average Accuracy	Average Execution time (in seconds)	Average Memory Usage (in MB)
Right-alignment	88,607	7,501	13,211
Feature-aligned	91,197	10,951	14,888

Table 7: Increase in accuracy when using feature-aligned data

Results of the Wilcoxon Signed-rank test indicate that the accuracy obtained with feature-aligned data (*Mdn* = 91,248%) is significantly higher than that obtained with right-aligned data (*Mdn* = 88,585%), ($T = 0$; $p<0,05$; $r = -0,886$). Although this section indicated that an accuracy figure of 90% could be obtained by training *Lia* with feature-aligned training data, we will experimented some more to determine if further increases in accuracy could be obtained by appending the training data with a number of additional features.

4.4.4 ADDITIONAL FEATURES

The additional features that were appended to the training data with the view of improving the linguistic accuracy figure of Lia are introduced in this section. The additional features will be discussed in subsections 4.4.4.1 to 4.4.4.3, before the results obtained through the use of the features are presented in subsection 4.4.4.4. Note that the additional features were appended to the left hand side of every instance in the training data. For example, if an additional feature is appended to data containing 20 features, the data will then consist of 21 features.

4.4.4.1 SYLLABLES

The syllabification of the words in the training data was done by means of a hyphenator for Afrikaans that was developed by CTexT, called *Calomo* ("*C5 Afrikaanse Lettergreepverdeler vir Outomatiese Morfologiese Ontleding*" 'C5 Afrikaans Hyphenator for Automatic Morphological Analysis') [58]. *Calomo* takes any Afrikaans word as input and produces an output that indicates the syllable boundaries of the involved word. For example, the word "*hondehokdak*" 'dog house roof' will be analysed as "*hon-de-hok-dak*", where the hyphens indicate syllable boundaries [58]. The syllables that *Calomo* provided were appended to the left side of every instance in the training data as seen in Figure 18, thereby increasing the number of features from 20 to 27. The number of features appended to the training data was established as 7, since only 1,71% of the words in the training data have more then 7 syllables. If the syllables to be appended are less than 7, the extra feature positions are filled by underscores. If more than 7 syllables are to be appended, the extra syllables at the left hand side of the word are discarded.

```
ma,gi,ster,stu,dies,_,_,_,_,_,_,_,m,a,g,i,s,t,e,r,s,t,u,d,i,e,s,Rs>
on,gel,dig,heid,_,_,_,o,n,_,_,_,_,_,_,_,g,e,_,_,l,d,i,g,h,e,i,d,0
tro,pie,se,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,t,r,o,p,i,e,s,e,Re>
ro,by,ne,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,r,o,b,y,n,e,Re>
half,tro,pie,se,_,_,_,_,_,_,_,_,_,_,_,_,_,h,a,l,f,t,r,o,p,i,e,s,e,Re>
```

Figure 18: Training data using syllables as features

4.4.4.2 NUMBER OF LETTERS CONTAINED IN SYLLABLES

The use of syllables as features in the training data inspired us to experiment with the number of letters contained in the syllables as features. For example the word "*robyne*" 'rubies' contains three syllables, "*ro-by-ne*". Each of the syllables contains two letters, therefore the word was annotated as 2,2,2,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,r,o,b,y,n,e,Re>. The reason for including the number of letters contained in the syllables as features is because it provides more information about the morphological structure of the word. The word "*hemele*" 'heavens' for example is annotated as 2,2,2,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,h,e,m,e,l,e,Re> and the word "*jolyte*" 'parties' as 2,2,2,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,j,o,l,y,t,e,Re>. Both these words have the same number of letters contained in their syllables and they both have the same class. The addition of the number of letters contained in the syllables reduces the distance between these two words compared to the case when only letters and underscores are used as features.

The number of letters in the syllables was appended to the left-hand side of the instances in the training data, as shown in Figure 19, to yield training data similar to the data displayed in Figure 18.

2,2,4,3,4,_,_,_,_,_,_,_,m,a,g,i,s,t,e,r,s,t,u,d,i,e,s,Rs>
2,3,3,4,_,_,_,o,n,_,_,_,_,_,_,g,e,_,_,l,d,i,g,h,e,i,d,0
3,3,2,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,t,r,o,p,i,e,s,e,Re>
2,2,2,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,r,o,b,y,n,e,Re>
4,3,3,2,_,_,_,_,_,_,_,_,_,_,h,a,l,f,t,r,o,p,i,e,s,e,Re>

Figure 19: Training data using the number of letters in the syllables as features

4.4.4.3 PROBABILITY

Another additional feature that was appended to the training data is the probability that the word should be lemmatised, hereafter referred to as "lemmatisation probability". The lemmatisation probability can be based on a variety of the features of the word.

The plural form in Afrikaans is usually (but not always) formed when the letters *e* or *s* are appended to nouns. The letters *e* and *s* at the end of words can therefore serve as an indication that the word should probably be lemmatised. This resulted in the addition of an extra feature to the training data, based on the lemmatisation probability of words ending on certain letters. The probabilities associated with the letters (including letters containing diacritics) are displayed in Table 8.

The probabilities were calculated by determining the number of inflected and uninflected words in the training data that end on certain letters and substituting these values into equation 4.1.

$$\text{Lemmatisation Probability (letter)} = \frac{\text{number of inflected words ending on (letter)}}{\text{number of words ending on (letter)}} \qquad \textbf{(4.1)}$$

These probabilities do not present the lemmatisation probabilities of the words found in the full Afrikaans lexicon, since the probabilities were computed by only considering the words in the training data. The lemmatisation probabilities were only appended to the evaluation data and not to training data. The training data was instead appended with a feature with a value of 1 if the instance is to be lemmatised or 0 if the instance is already a lemma. This information was derived by considering the class of the word. Any other class than class 0 signifies that the value 1 must be appended, while class 0 indicates that the value 0 should be appended.

The logic behind the idea of using probabilities in the data was that the distance between a word with a high lemmatisation probability and a word that should definitely be lemmatised (lemmatisation probability of 1) will be shorter than the distance between a word with a low lemmatisation probability and a word that should be lemmatised. For example the distance between the evaluation instance 0.7527,k,o,p,l,a,m,p,e and the training instance 1,s,e,i,n,l,a,m,p,e will be shorter than the distance between 0.1733,o,l,i,e,l,a,m,p,_ and 1,s,e,i,n,l,a,m,p,e.

Word ending on letter	Probability that word should be lemmatised
a	0,1328
b	0,2000
c	0,0000
d	0,0946
e	0,7525
ê	0,4571
ë	0,7418
f	0,1594
g	0,0419
h	0,0000
i	0,2887
j	0,0000
ï	0,0000
k	0,1146
l	0,1405
m	0,0910
n	0,0809
o	0,0345
p	0,1733
q	0,0000
r	0,2386
s	0,6381
t	0,0710
u	0,2077
v	0,0000
w	0,0000
x	0,0000
y	0,0914
z	0,0000

Table 8: Lemmatisation probability of words ending on the displayed letters

Figure 20 presents evaluation data with lemmatisation probabilities (based on the last letter of the word) as an additional feature. An example of training data for the same instances as in Figure 20 is presented in Figure 21.

```
0.752503577,o,b,b,e,l,m,a,s,j,i,e,n,p,r,o,b,l,e,m,e,
0.140485313,_,_,_,_,_,_,_,_,_,k,l,i,m,t,o,e,s,t,e,l,
0.094578552,_,_,_,_,_,_,_,_,_,v,i,e,r,d,u,b,b,e,l,d,
0.238616912,g,e,_,_,_,_,_,_,_,_,_,r,e,g,u,l,e,e,r,
0.114567901,_,_,_,_,_,_,_,_,_,_,_,_,_,_,o,o,g,l,i,k,
0.752503577,s,a,a,m,_,_,_,_,g,e,_,_,_,_,_,d,r,y,w,e,
```

Figure 20: Evaluation data with the probability of the last letter as a feature

```
1,o,b,b,e,l,m,a,s,j,i,e,n,p,r,o,b,l,e,m,e,Rme>em
0,_,_,_,_,_,_,_,_,_,k,l,i,m,t,o,e,s,t,e,l,0
0,_,_,_,_,_,_,_,_,_,v,i,e,r,d,u,b,b,e,l,d,0
1,g,e,_,_,_,_,_,_,_,_,_,r,e,g,u,l,e,e,r,Lge>
0,_,_,_,_,_,_,_,_,_,_,_,_,_,_,o,o,g,l,i,k,0
1,s,a,a,m,_,_,_,_,g,e,_,_,_,_,_,d,r,y,w,e,Mge>
```

Figure 21: Training data with lemmatisation probability of the last letter as a feature

The inclusion of lemmatisation probability based on the last letter of the word as an additional feature encouraged us to also include lemmatisation probability based on the strings contained within a word that correspond to the inflectional affixes as an additional feature. The inclusion of this additional feature is justified when considering for example that words ending on the affix *-etjies* have a lemmatisation probability of 0,9677. The inflectional affixes used as the basis for computing the lemmatisation probabilities are the inflectional affixes of the eight categories of inflection for Afrikaans as defined in Section 3.4. The complete list of the affixes is included in Addendum B. The lemmatisation probabilities for the affixes were computed in the same manner as the lemmatisation probabilities of the last letters. The probabilities were once again computed on the training data and can therefore not be accepted as valid for the vocabulary of Afrikaans. The training and evaluation data has the same form as the training and evaluation data for the probabilities based on the last letter of the word (See Figure 20 and 21). The lemmatisation probabilities of all the inflectional affixes used in this study can be found in Addendum C.

We also wanted to include word length as a feature, since we reasoned that there is a greater probability that longer words should be lemmatised than shorter words. Word length proved to be a worthless feature because there is only a small difference between the average word length of the word-forms (10,79 letters) and their respective lemmas (10,15 letters) in the training data. This small difference does not justify the inclusion of an additional feature based on word length.

4.4.4.4 RESULTS

The results obtained by including the additional features presented in this section are presented and compared in

Table 9 in terms of average accuracy, execution time and memory usage. The results in

Table 9 show that feature-aligned data delivers the best results in terms of accuracy, execution time and memory usage. We once again used data with 20 features and the default TiMBL algorithm and parameter settings for constructing the classifiers.

Data Representation	Average Accuracy	Average Execution time (in seconds)	Average Memory Usage (in MB)
Feature-aligned	91,197	10,951	14,888
Syllables	90,775	61,778	23,563
Number of Syllables	90,390	24,515	19,352
Last letter probability	91,203	144,345	15,034
Affix probability	88,229	105,791	15,034

Table 9: Performance comparison for additional features

The ranks produced by Friedman's ANOVA for the different data representation options are displayed in Table 10. Results from Friedman's ANOVA indicate that the different data representations have a significant effect on the obtained accuracy (F = 38,819,051; p<0,05 ; df = 4). The results obtained through the Wilcoxon Signed-rank test indicate that no significant differences exist between the data sets for feature-aligned and last letter probability data (T = 0; r = -0,507; p<0,005). A Bonferroni correction was applied and therefore all the effects are reported at a significance level of 0,005.

Data Representation	Mean Rank
Feature-aligned	4,35
Syllables	2,10
Number of Syllables	2,90
Last letter probability	4,65
Morpheme probability	1,00

Table 10: Ranks for different data representation options

This subsection indicated that other features additional to letter sequences may aid *Lia* in the classification process. We must once again emphasize that no deductions should be made regarding the performance of these data sets for classification algorithms other than IB1. The

results presented in this subsection only aim to introduce the various data representations and to get an idea of what can be expected from the data representations in terms of performance. The interaction betweens algorithm parameters and data representation may result in data representations other than feature-aligned data delivering better results for other classification algorithms. This will be further investigated in Chapter 5.

4.5 VISUALISING THE NEAREST NEIGHBOUR SET

It is often difficult to visualise the nearest neighbour relations in an instance base. This is also true in the case of *Lia,* as each instance in *Lia's* training data has at least 13 features. This means that every instance is represented as a set of points in a 13-dimensional space. Visualising this 13-dimensional space is very hard, if not impossible.

We made an effort to represent an extraction from *Lia*'s training data in a two-dimensional space by using *Kngraph* [59]. The result of this is a graphical presentation of the nearest neighbour relations among the different instances in the training set (see Figure 22 below). Every instance is represented as a node with arcs denoting the nearest neighbour relations. The 10 most frequent classes in the training data have been coloured in. Figure 22 was constructed by considering an extraction of only 1 000 instances from the training set, resulting in the exact number of 1 000 nodes in the figure. It is possible to construct a similar figure for all of the instances in the training set, but this will result in a very crowded picture that will be meaningless if included in this document.

Different coloured nodes that are connected indicate incorrect nearest neighbour relations and therefore represent errors that will be made during the classification process. Figure 22 shows definite structure in terms of the positioning of similar instances. The purple coloured nodes (class *Mge>*) for example are only found in two different regions in Figure 22. The red nodes (class *0*) are found throughout all the regions in the training set, but this is due to the fact that these instances are so different in form, compared to the quite similar instances belonging to class *Mge>*.

Figures denoting nearest neighbour relations are also useful for locating errors in the training data, because nodes that are not connected to other nodes indicate definite errors. *Knngraph* is also very useful for predicting whether a certain problem can be successfully resolved through the use of MBL. This can be done by studying the positioning of similar coloured nodes and also the number of links between the nodes in the representation of the instance base. The less structure in terms of the positioning of similar coloured nodes and the fewer the links between similar coloured nodes, the less suited is the problem to be resolved by means of MBL.

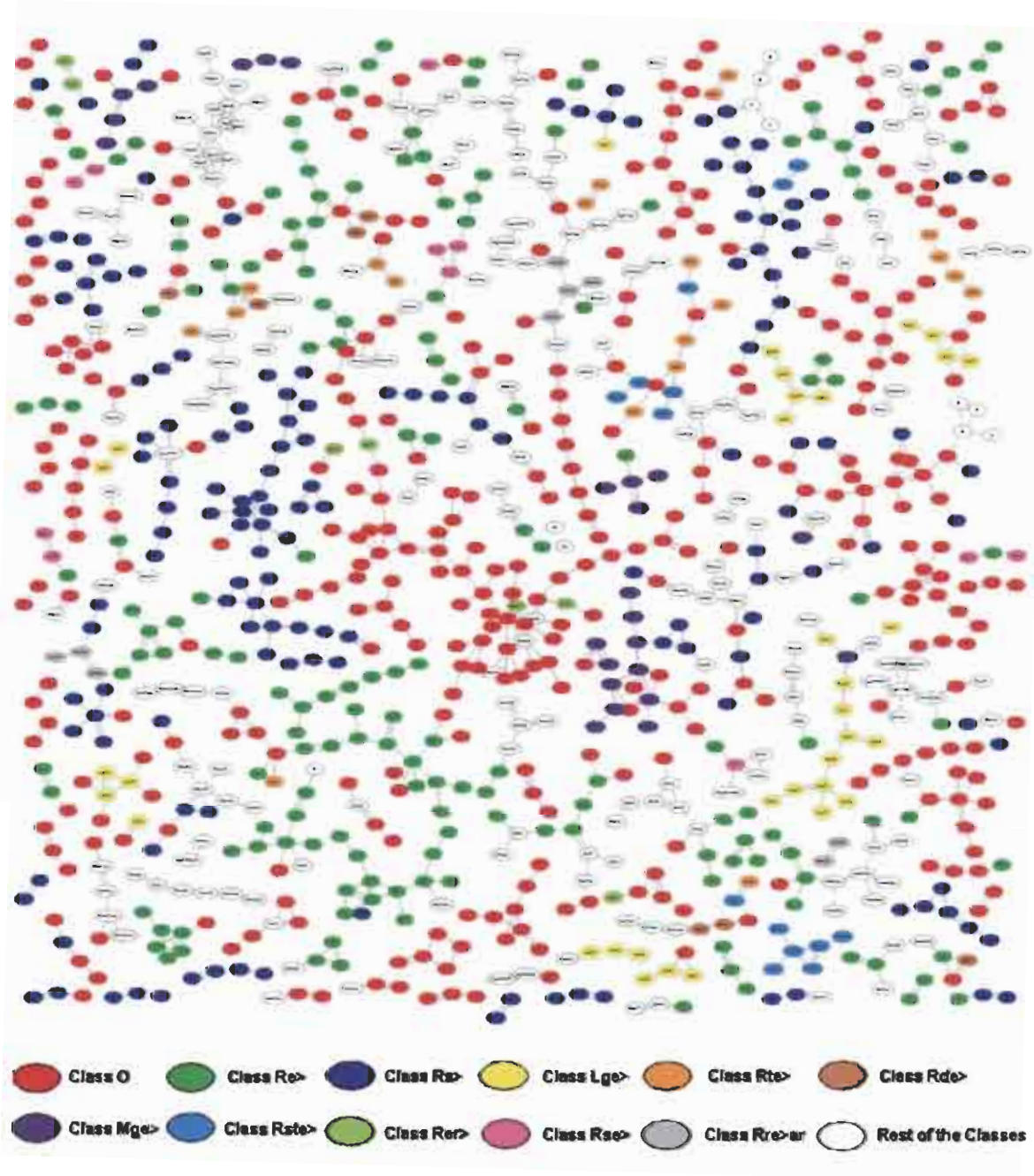

Figure 22: The Nearest neighbours in a training set of 1 000 words

4.6 CONCLUSION

The purpose of the first part of this chapter was to describe the various aspects of *Lia's* training data. The description of the training data started in Section 4.2 with an introduction to the process of data generation, where we first concentrated on the extraction of the training data. We argued that it is very important to also train *Lia* with negative instances, since *Lia* should not only learn how to lemmatise, but also when to lemmatise and when not to. We

also motivated the decision of training *Lia* with the lowest possible number of training instances. The data annotation process was described in Section 4.2.2, where a bootstrapping process was followed by using *Lia* to assist in the annotation process. We also explained how the quality of the training data was ensured.

Section 4.3 addressed the first part of **Research Question 2 (What is the influence of the data size on the performance of the system?)** by considering a graph of accuracy versus the number of training instances. This illustrated the fact that machine learning systems improve with experience, and that *Lia* is no exception. The graph also showed that it is difficult to improve the accuracy figure of *Lia* by just adding more instances to the training data. We introduced a logarithmic trend line which indicated that an accuracy figure of 90% will be achieved by training *Lia* with data containing approximately 100 000 instances. Due to the fact that we do not have an annotated training data set of 100 000 instances available, and that the generation of training data is such a resource expensive operation, we focused on reaching an accuracy figure of 90% by experimenting with a number of feature options.

Section 4.4 introduced the various feature options (i.e. number of features, alignment and positioning of features and additional features) available for extending the training data. The aim of Section 4.4 was to address the second part of **Research Question 2 (What is the influence of the various feature options on the performance of the system?).** We therefore proceeded to indicate the influence of the different feature options on *Lia's* performance for each of the feature options. We saw that the number of features in the training data, the alignment of features and the use of additional features in the training data all have a statistically significant effect on the accuracy of *Lia*. A significant result was obtained in Section 4.4 when the use of feature-aligned data resulted in an accuracy figure of 91,167%. This result is very important, since it indicates that an accuracy figure of 90% can indeed be reached without training with a large data set of 100 000 words.

The chapter concluded with a visualisation of the nearest neighbour relations in an extraction from the training data. This visualisation showed that *Lia's* training data has definite structure in terms of the positioning of similar coloured nodes, thereby providing further motivation for following a memory-based learning approach in the construction of *Lia*.

Chapter 5 will now focus on the various aspects of the performance component of *Lia*. Chapter 5 commences with information regarding the operation of the TiMBL classification algorithms and parameters, before the focus will shift to obtaining the best data representation and parameter settings for each algorithm in order to provide answers to **Research Question 3 (Which algorithm deliver the best performance in terms of linguistic accuracy, classification time and memory usage?)**.

Chapter 5: PARAMETER SETTINGS FOR LEMMATISATION IN AFRIKAANS

5.1 INTRODUCTION

Any memory-based learning system such as *Lia* consists of two components, namely a learning component and a performance component [60]. The learning component involves storing data in memory, while the performance component involves performing a classification based on the data in memory. This classification is done according to some classification algorithm and parameters. While Chapter 4 in part focussed on the learning component (i.e. the structure of the data), the focus of this chapter is be more specifically on the performance component (i.e. the classification algorithms and parameters) of *Lia*.

We start this chapter with a description of the classification algorithms available in TiMBL. We then describe the various algorithmic parameter settings available for these classification algorithms. We first describe four different feature weighting possibilities, which are followed by an introduction to the various distance metrics used for computing the distance between two instances. Then the different ways in which a class is assigned to a new query instance (i.e. class voting) are discussed. The frequency threshold for the two distance metrics, *Modified Value Difference Metric* and *Jeffrey Divergence* are introduced in Section 5.3.2. The overview of the algorithm and parameter options is concluded with an explanation of the tie breaking process in TiMBL.

Section 5.4 introduces the process of Wrapped Progressive Sampling which forms the basis of the operation of *Paramsearch*. We then motivate the construction of our own implementation of *Paramsearch* called *PSearch*. The results obtained with using *Paramsearch* and *PSearch* are then compared with the results from an exhaustive search to determine their suitability for producing algorithmic parameters that are expected to perform well at the task of Afrikaans lemmatisation.

The chapter concludes with a section that explicitly addresses **Research Objective 3 (Which algorithmic parameter settings deliver the best performance in terms of linguistic accuracy, execution time and memory usage?)**. *PSearch* is used in this section to determine the data representations and algorithmic parameters for IB1, IB2, TRIBL and TRIBL2 that deliver the best performance in terms of linguistic accuracy. *PSearch* was not used to determine the best data representation and algorithmic parameters for IGTree, because IGTree has a small number of available parameter options that makes it possible to determine the best data representation and algorithmic parameters by means of an exhaustive search.

5.2 CLASSIFICATION ALGORITHMS

5.2.1 IB1

IB1 [35] is the basic instance-based algorithm used in TiMBL, and its operation is similar to the basic *k*-NN algorithm previously described in Section 2.4.1. IB1 uses the *Overlap* similarity metric (see Section 5.3.2.1) to compute the distance between a new instance and the data that is already stored in memory.

The concept description of IB1 changes over time with the addition of more instances to the training data stored in memory (hereafter referred to as the instance space). This concept description is derived from the similarity and classification functions of IB1 on the training data, and provides an idea of the classification boundaries between the different classes in the instance space. The concept description enables us to determine the regions in the instance space that belong to certain classes and which will be used to classify new instances according to the *k*-NN algorithm employed by IB1.

For example, consider a two-dimensional instance space defined by two classes, positive '+' and negative '-' [35]. 100 randomly selected instances containing both classes are used as training data. The approximated target concept, indicated by the regions inside the dotted lines, is presented in Figure 23 in the form of a Voronoi diagram [61] at three different stages during training. The predicted target concept is indicated by the regions enclosed by the solid lines. Each solid line lies halfway between groups of positive and negative instances. The

ideal situation is that the predicted and approximated target concepts correspond. The first stage is displayed when only 5 instances are considered; we see that there is some degree of correspondence between the predicted and approximated target concepts. This degree of correspondence increases as the number of training instances is enlarged. The positive and negative signs are not displayed for the third stage, because the figure will be crowded when all of the 100 instances are displayed.

The Voronoi diagram of Figure 23 shows some similarity with Figure 22 (cf. Chapter 4), where *Kngraph* was used to construct a graphical representation of the nearest-neighbour relations in an extraction from *Lia's* training data. One difference between the two figures is that Figure 22 (cf. Chapter 4) represents an attempt to plot a 38-dimensional space in 2 dimensions, while Figure 23 is a true representation of a 2-dimensional space in 2 dimensions. The other difference is that the target concept is represented by groups of nodes having the same colour in Figure 22, compared to the regions enclosed by solid lines in Figure 23. Figure 23 indicates how IB1's approximation of the target concept improves as more data is used for training. This improvement of the concept description with the addition of more training data, provides further proof that the performance of memory-based learning algorithms relies on the amount of training data used.

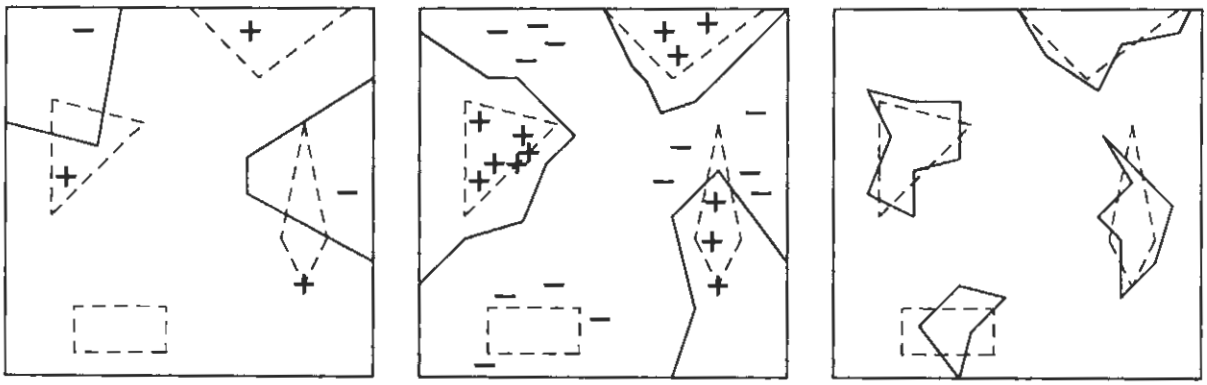

Figure 23: The improvement of *IB1*'s concept description with training [35]

5.2.2 IB2

The IB1 algorithm stores all the instances of the training data in memory. This implies that all instances, even those that fall well within the borders of the target description, are saved in memory. Instances like these have no effect during classification, as they do not affect the generalisation accuracy of the system - they only confirm the positions of the already existing target descriptions. It will therefore make sense to restrict the instances in memory to those that will have an effect (i.e. the instances that lie beyond or on the boundaries of the target description) on the classification of other instances [16]. This approach of not storing instances that fall within the boundaries of the target descriptions forms the basis of the operation of the IB2 algorithm.

The operation of IB2 is therefore identical to the operation of IB1, except that IB2 only saves misclassified instances in the training data [35]. IB2 starts with an instance base containing only a small portion of the available training instances. This initial number of training instances in memory can be set by the user. IB2 then adds instances into memory only when they are misclassified by the *k*-NN algorithm, on the basis of the instances already in memory at that point.

Additional instances are added because they do not form part of the concept description of the instances in memory at that particular time. They are included to expand the concept description because we can assume that they represent a part of the instance space where there is not enough instances contained in memory for accurate classification [16].

IB2 does not lower or improve the generalisation performance of IB1, but it lowers the computational burden of the system since there are less training instances stored in memory. The negative aspect of IB2 is that it is sensitive to noisy training data, since it has a strategy of only storing misclassified instances [35].

5.2.3 IGTree

The IGTree algorithm compresses an instance base as utilised by IB1 and IB2 into a tree structure. This tree structure is organised through the concept of *Information Gain* (IG) [62].

The advantage of IGTree is that only a small number of training instances is stored in memory (compared to the IB1 algorithm that stores all of the training instances). This reduced storage structure reduces the computational effort and the execution time.

IGTree entails two different algorithms, one for constructing the tree structure, and the other one for classification. The tree structure involves that instances are stored as arcs of connected nodes [30]. Each of the nodes contains a test and a class label. The test is based on one of the features, while the class label represents the default class at that node. The arcs denote the outcomes of the tests at the nodes.

The order in which features are used for testing in the tree is determined by the IG of the involved features. The depth of the tree is equal to the number of features [16]. The reasoning behind the IGTree structure is that certain features have more weight in determining the class value. This weight is determined by the IG of the feature. When the search through memory is conducted during the classification phase, the search can be restricted to matching test instances to those stored instances that have the same feature value at that feature. The matching can be expanded by also examining the second most important feature, followed by the third, etc.

This tree structure compresses the instance base, because similar instances share partial paths [16]. The paths are restricted to those input feature values that distinguish the classification from all the other instances in the training set. It is not necessary to fully store an instance as a path when only a few feature values of the instance make the instance classification unique. The feature-values with lower IG are discarded and thus not stored in the tree.

The terminal or leaf nodes (nodes that do not have any further arcs connected to them), represent a definite class [63]. Non-terminal nodes represent the most probable classification at that stage of the tree. Determining the correct class of an instance involves traversing the tree from top to bottom (i.e. matching from the top of the tree downwards). Traversing involves matching all feature-values of the test instance with arcs in the order of the overall feature IG and obtaining a classification when a leaf is reached or using the classification implied by a non-terminal node if a leaf node is not reached.

The tree structure of IGTree is further compressed by pruning [16]. The leaf-node daughters of a mother node that have the same class as their mother are discarded from the tree. The reason for this is that the class information does not contradict the class of the mother node. This tree compression method does not have any negative influence on the accuracy of the algorithm. Figure 24 shows the conversion of an instance base into an IG tree structure [30].

Size	Shape	Holes	Class
small	compact	1	*nut*
small	long	none	*screw*
small	long	1	*key*
small	compact	1	*nut*
large	long	1	*key*
small	compact	none	*screw*
small	compact	1	*nut*
large	long	none	*pen*
large	long	2	*scissors*
large	long	1	*pen*
large	other	2	*scissors*
small	other	2	*key*

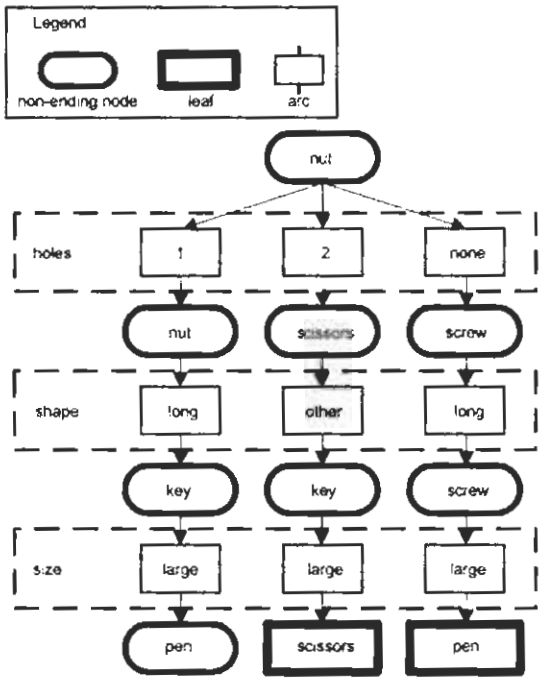

Figure 24: Conversion of an Instance base into an IGTree

The features in Figure 24 are arranged in order of decreasing IG, e.g. *holes*, *shape* and then *size*. The first node contains the class with the highest frequency; in this case it is the class "nut". An arbitrary class is selected if there is more than one class with this highest frequency. The number of *holes* is then included in the tree; this may be 1, 2 or none. The

most likely outcome for having the feature values of the first row of features is then presented at the next node level. For example, having 1 *hole* means that you are most likely working with a "nut", 2 *holes* are probably "scissors" and no *holes* implies that you may have the class "screw". *Shape* is the next feature that is included in the tree. The feature values of *shape* are determined by the feature values of the previous feature. 1 *hole* for example implies that the most likely *shape* of the object is "long", 2 *holes* denote "other", while no *holes* implies the most likely *shape* is again "long". The most likely outcome for having both the feature values of the first and second row are then included in the tree structure. For example 1 *hole* and "long" imply that the object is most likely a "key", no *holes* and shape "other" denote class "screw" and 2 *holes* and shape *other* imply that the most likely class is again "key". This way of converting to a tree structure is repeated throughout the entire depth of the tree.

5.2.4 TRIBL AND TRIBL2

IGTree performs less well (compared to IB1) on data sets with small IG differences between the features [16]. TRIBL was designed as a combination between IB1 and IGTree with the aim to exploit the trade-off between the search speed of IGTree and the generalisation accuracy of IB1 [16].

TRIBL splits the classification of new instances into a quick decision-tree (IGTree) traversal based on the first (most important and most class-disambiguating) features, followed by a slow but accurate *k*-NN (IB1) classification based on the remaining less important features [63]. For TRIBL, a parameter determines the switching point in the feature ordering from IGTree to IB1. During search, the normal IGTree search algorithm is used, until a feature having an IG value below the average IG for all the features is reached. This point represents the shift form IGTree to IB1.

TRIBL2 does not employ a fixed switching point [32]. During the classification of an instance it continues to use IGTree as long as it finds matching feature values in the weighting-governed feature ordering. It only reverts to IB1 classification when a mismatch is found. The reasoning behind this mismatch-based switching is that the switch to IB1 is only invoked when mismatching occurs, being the typical point in which IB1 can improve on decision-tree-style classification.

5.3 ALGORITHM PARAMETERS

There are large numbers of parameters available that can be used in conjunction with the five classification algorithms. These algorithms may have a significant effect on the accuracy of *Lia,* therefore it is very important to select the subset of parameters that delivers the best performance. The various parameters are now introduced.

5.3.1 FEATURE WEIGHTING

5.3.1.1 INFORMATION-GAIN FEATURE WEIGHTING AND GAIN RATIO

Overlap metric [16] (see Section 5.3.2.1) simply counts the mismatches between the corresponding feature-values. This is a good method to use in the absence of information about feature relevance. The alternative method is to compute statistics about the relevance of features by looking at the features that are good predictors of the class labels. These features are determined by computing the Information Gain (IG) [62]. IG is a reflection of how knowledge about a feature's value decreases the uncertainty about the class of the instance.

IG [62] of feature i is measured by computing the uncertainty (entropy) between the case in which the value of the feature is known and the case in which it is unknown (Equation 5.1) [16]. The IG of a feature i is computed by subtracting the summation over the probabilities of feature i's values multiplied by the entropy of the class entropy (having knowledge about the feature's values) from the class entropy (having no knowledge about the feature's values) [16]. C is the set of different classes. V_i is the set of values for feature i.

$$w_i = H(C) - \sum_{v \in V_i} P(v) \times H(C \mid v) \qquad \textbf{(5.1)}$$

Where

w_i is the IG of feature i. $H(C)$ (equation 5.2) is the entropy of the class labels. $P(C)$ is the probability of the class C.

$$H(C) = -\sum_{c \in C} P(c) \log_2 P(c) \quad \textbf{(5.2)}$$

The downside of IG is that it tends to overestimate the relevance of features having large numbers of values [16]. Consider the situation where you have a database of students (the instances). One of the features is a unique student number. The consequence of this is that the student number feature has a very high IG, but it does not give any information about new instances.

IG is normalised for features with different numbers of values through *Gain Ratio* (Equation 5.3). *Gain Ratio* is IG divided by split info (*si*(*i*)), the entropy of the feature-values (Equation 5.4) [16].

$$w_i = \frac{H(C) - \sum_{v \in V_i} P(v) \times H(C \mid v)}{si(i)} \quad \textbf{(5.3)}$$

$$si(i) = -\sum_{v \in V_i} P(v) \log_2 P(v) \quad \textbf{(5.4)}$$

The *Gain Ratio* and *IG* values can be used as weights w_i in the distance metric equation (Equation 5.9 in Section 5.3.2.1) which can then be defined as the weighted distance metric (Equation 5.5).

$$\Delta(X, Y) = \sum_{i=1}^{n} w_i \delta(x_i, y_i) \quad \textbf{(5.5)}$$

5.3.1.2 CHI-SQUARED WEIGHTING

The *Gain Ratio* statistic has an unwanted bias towards features that have more values, because it does not correct for the number of degrees of freedom of the contingency table of classes and values [16]. The contingency table is a frequency table of feature values (on the y-axis) for each of the classes (on the x-axis). The *chi-squared* statistic [16] is computed as indicated by Equation 5.6:

$$x^2 = \sum_i \sum_j \frac{(E_{ij} - O_{ij})^2}{E_{ij}} \tag{5.6}$$

O_{ij} is the observed number of cases with value v_i in class c_j (O_{ij}=n_{ij}). E_{ij} is the expected number of cases which should be in cell (v_i,c_j) of the contingency table if the null hypothesis of no predictive association between feature value and class is true. E_{ij} is computed as in Equation 5.7.

$$E_{ij} = \frac{n_{.j} n_{i.}}{n..} \tag{5.7}$$

Where

$n_{.j}$ denotes the sum of all the frequency values for class j, with $n..$ the total number of cases (the number of cells in the table) and n_i the sum of all the frequency values for value i. The value of the *Chi-square* statistic can be used as a weight in Equation 5.5.

5.3.1.3 SHARED VARIANCE WEIGHTING

The *Chi-squared* statistic can be explicitly corrected for the degrees of freedom through the use of the Shared Variance [16] measure as shown in Equation 5.8:

$$SV = \frac{x_i^2}{Nx(\min(|C|,|V_i|)-1)} \tag{5.8}$$

Where:

$|C|$ is the number of classes, $|V_i|$ is the number of values for feature I and x^2 is the *Chi-squared* statistic from Equation 5.7. N is the number of instances. The value obtained form the Shared Variance measure can also be used as a weight in equation 5.5.

5.3.2 DISTANCE METRICS

The following distance metrics [16,32] are used to determine the similarity between a query and an instance.

5.3.2.1 OVERLAP METRIC

The most basic metric that works for patterns with symbolic features like those that will be used by *Lia*, is the *Overlap* metric. The equations for the *Overlap* metric are given in equations 5.9 and 5.10.

$$\Delta(X,Y) = \sum_{i=1}^{n} \delta(x_i, y_i) \tag{5.9}$$

$$\delta(x_i, y_i) = 0 \text{ if } x_i = y_i \text{ or } \delta(x_i, y_i) = 1 \text{ if } x_i \neq y_i \tag{5.10}$$

Δ(X,Y) is the distance between instances X and Y. The instances are represented by n features, with δ the distance per feature. The distance between two instances is equal to the sum of the differences between the features.

5.3.2.2 MODIFIED VALUE DIFFERENCE METRIC

Overlap is limited to exact matches between feature values. All values of a feature are seen as equally dissimilar. Logically, we know that some feature values are more similar than others. *Modified Value Difference Metric* (*MVDM*) was defined to include this information about similar feature values.

MVDM determines the similarity of the values of a feature by looking at the co-occurrence of values with target classes. The distance between two values v_1 and v_2 of a feature is computed through the difference of the conditional distribution of the classes C_i for these values (Equation 5.11).

$$\delta(v_1, v_2) = \sum_{i=1}^{n} | P(C_i \mid v_1) - P(C_i \mid v_2) | \tag{5.11}$$

$P(C_i|v_1)$ is an example of a conditional probability which, expressed in words, means the probability of class C_i occurring, given feature value v_1. These probabilities are computed from the relative frequencies in the training set.

Feature relevance is not computed in *MVDM*, but there is an implicit feature weighting system present. The class probabilities of informative features will lean towards a particular class. This implies that on average the $\delta(v_1,v_2)$ will be large for that particular feature. Uninformative features will accordingly have a smaller $\delta(v_1,v_2)$ because the value of the conditional class probabilities will be quite similar.

The way in which the nearest neighbour sets are composed in *MVDM* differs from the way in which it is done in *Overlap*-based metrics. *Overlap* causes an abundance of ties in the nearest neighbour position; this problem is reduced or completely resolved by using *MVDM*. For example: if your test instance differs one mismatch from the nearest neighbour, *Overlap* will yield all the instances which have different feature values in the mismatch position as nearest neighbours. Contrary to this, *MVDM* will only propose the nearest neighbour with the lowest delta in the mismatch position. This means that *MVDM* yields a much smaller nearest neighbour set than the *Overlap*-based metrics.

5.3.2.3 JEFFREY DIVERGENCE METRIC

The difference between *MVDM* and *Jeffrey Divergence Metric* is that *MVDM* computes a straightforward geometrical distance between two class distributions, while *Jeffrey Divergence* introduces a logarithm term. The distance between two features is computed according to *Jeffrey Divergence* as in Equation 5.12:

$$\delta(v_1,v_2)=\sum_{i=1}^{N}\left(P(C_i\mid v_1)\log\frac{P(C_i\mid v_1)}{m}+P(C_i\mid v_2)\log\frac{P(C_i\mid v_2)}{m}\right) \quad \textbf{(5.12)}$$

$$m=\frac{P(C_i\mid v_1)+P(C_i\mid v_2)}{2} \quad \textbf{(5.13)}$$

Jeffery Divergence assigns relatively larger distances to value pairs of which the class distributions are further apart. It basically assigns more prominence to zero probabilities than *MVDM*.

5.3.3 CLASS VOTING

5.3.3.1 MAJORITY VOTING

Majority voting [32] is the basic method used to determine the class to be assigned to a query. It consists of doing a frequency count of the classes occurring in the nearest neighbour set, and then assigning the most frequently occurring class to the query. This means that all instances in the set of nearest neighbours contribute equally to the decision about which class to assign to the query, irrespective of their distances from the query point.

5.3.3.2 DISTANCE WEIGHTED CLASS VOTING

It may be argued that instances in the nearest neighbour set with a small distance to the query should have more influence on the decision about which class to assign to the query than instances that are further away from the query. This is a more logical way of determining a class than just ordinary majority voting. There are three types of distance weighted class voting weights [16] namely, *Inverse Linear Weighting*, *Inverse Distance Weighting* and *Exponential Decay Weighting*.

- **INVERSE LINEAR WEIGHTING (IL)**

Inverse Linear weighting assigns a heavier weight to a neighbour that is a smaller distance away from the query than a neighbour that is a greater distance away. The weight of an instance$_j$ (w_j) in the set of nearest neighbours is computed as in Equation 5.14:

$$w_j = \frac{d_k - d_j}{d_k - d_1} \text{ if } d_k \neq d_1 \text{ or } w_j = 1 \text{ if } d_k = d_1 \qquad \textbf{(5.14)}$$

Where:

d_j is the distance to the query of the j'th nearest neighbour

d_1 the distance of the query to the nearest neighbour

d_k the distance from the query to the furthest (k'th) neighbour.

When d_j is (nearly) as large as d_k, the nominator in Equation 5.14 will be very small, so the weight of instance$_j$ will be small. The weight will be one if instance$_j$ is the nearest ($d_j = d_1$).

- **INVERSE DISTANCE WEIGHTING (ID)**

Inverse Distance weighting [64] implies that weights, inversely proportional to the distance from the query instance, are awarded to the neighbouring instances according to equation 5.15:

$$w_i = \frac{1}{d_j} \text{ if } d_j \neq 0 \quad \textbf{(5.15)}$$

- **EXPONENTIAL DECAY WEIGHTING (ED)**

The weight of a neighbour point is solely dependant on its distance from the query instance in equations 5.14 and 5.15. This leads us to make the false assumption that the relationship between distance and relevance to the query is fixed over different data sets. Exponential Decay Weighting, based on work done by Shepard [65], is used to overcome this false assumption. Shepard proposes a universal perceptual law which states that the relevance of a previous stimulus to the birth of a new stimulus is an exponential decreasing function of its distance in psychological space. Equation 5.16 states the formula for exponential decay weighting:

$$w_j = e^{-\alpha d_j^{\beta}} \quad \textbf{(5.16)}$$

Figure 25 [16] shows the function curves of ID and ED (with varying values for α and β). ID and IL assign differing weights to close neighbours and less differing weights to more distant neighbours. ED has a shallower curve than ID for higher distances, meaning that there is less variance in the weights awarded to the more distant neighbours. ID also assigns very high weights to near-matching neighbours; ED on the other hand, assigns a maximum weight of one to these neighbours. Higher weights are awarded if the value of α is increased, while an

increase in β tends to make the ED curve bell shaped. This means that less differing weights are assigned to near-matching neighbours.

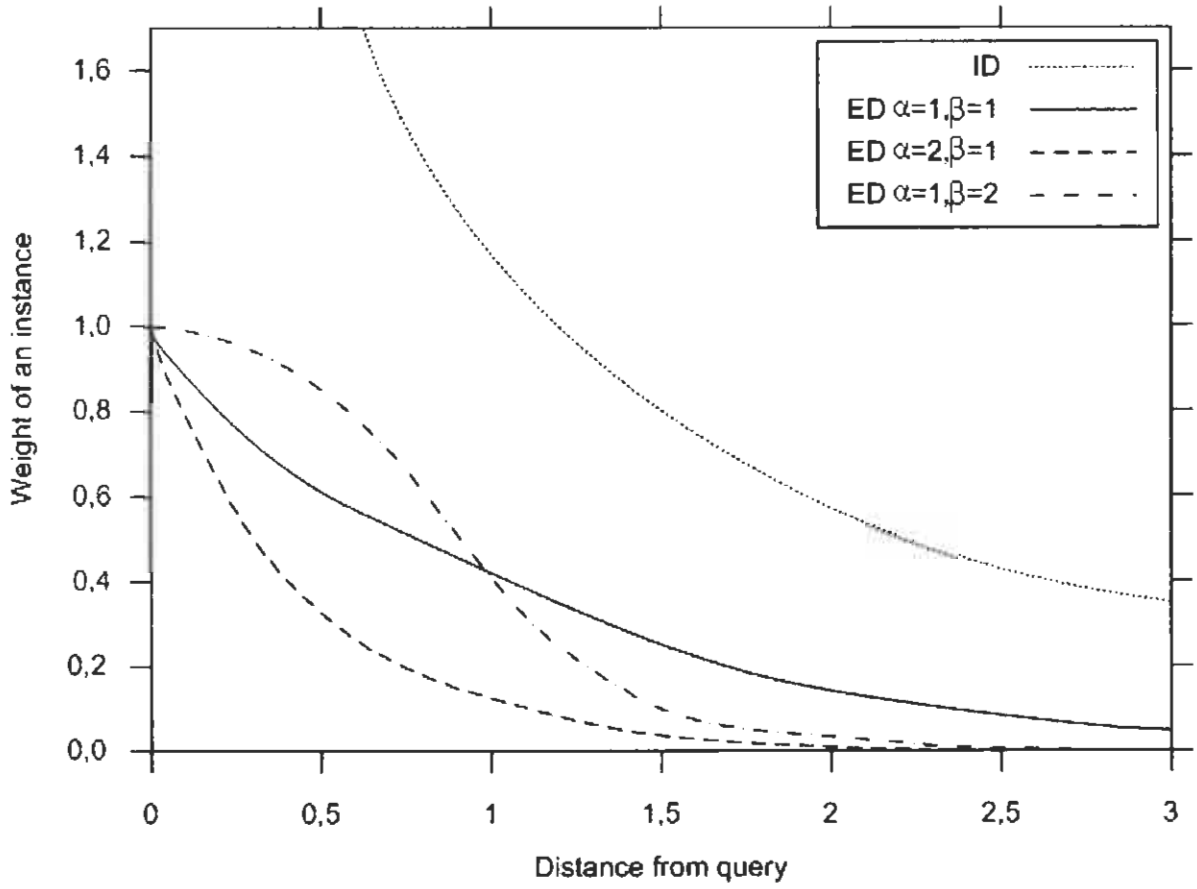

Figure 25: Inverse Distance (ID) and Exponential Decay (ED) with three different settings for α and β

5.3.4 Frequency Threshold

The frequency threshold can only be used with the *MVDM* and *Jeffrey Divergence Distance Metrics*. When one or both values in a pair of matched values occur less frequently in the data set than the value of the frequency threshold, the distance metric used is switched from either *MVDM* or *Jeffrey* to the *Overlap* distance metric [16].

5.3.5 Tie breaking

The last step of *k*-NN classification is taking the majority category among the set of nearest neighbours, where their vote is either unweighted or weighted by their distance function. Ties

are very likely to occur, especially when a weighted distance function was not used. The procedure for tie breaking [16] in TiMBL is as follows: The value of the *k* parameter is incremented by 1, and the additional nearest neighbours at this new *k*th distance are added to the current nearest neighbour set. The majority category in this new neighbour set is now taken as the category to be awarded. This will hopefully resolve the tie, otherwise TiMBL can be set to avoid ties by making a random choice of a classification from a class distribution in a nearest-neighbour set, weighted by the distribution of the classes in the set.

5.4 Automatic Parameter Selection

5.4.1 Wrapped Progressive Sampling

It is a well-known fact that large fluctuations in generalisation accuracy can be observed when changing the parameter settings of memory-based learning algorithms [60]. One of the objectives of this study is to determine the best data representation and parameter setting for the Afrikaans lemmatisation task. However, obtaining these parameter settings for optimal performance can be a difficult task. The default method used, is to train the system with all available settings for every algorithm and then to compare the obtained accuracies. However, this approach is time-consuming and computationally intensive, because the system has to be retrained for every possible parameter setting of the involved algorithm. For instance, TiMBL has five algorithms, six distance metrics, five feature weighting possibilities and three class voting weights. The number of nearest neighbours to consider can also be defined for some algorithms. If four classification algorithms are considered (IGTree is not included as it only has five parameter options) together with ten different numbers of nearest neighbours, we will, for example, have to train the system 3 600 times (4x6x5x3x10=3 600). This number of experiments is only for purposes of illustration as some of the parameter settings can only be used with certain algorithms. The problem is that this approach is computationally very expensive. An exhaustive search through all the combinations of parameters for IB1 on one representation of *Lia's* training data took 176 hours and 28 minutes to complete. Keep in mind that we have four data representations and five different additional features. This will result in 20 different experiments that could each take a minimum of 176 hours and 28 minutes. This would result in more than 147 days of continuous training and evaluation,

which is not desirable at all. An alternative to using an exhaustive search for algorithmic parameter optimisation is to use Wrapped Progressive Sampling (WPS) [66]. The rest of this section will describe the process of WPS and the results obtained through the use of it.

WPS can be viewed as a competition between different algorithm parameter settings to determine the average best-performing setting, and can be compared to a race between mountaineers to reach the top of a mountain [21]. The race begins with all of the mountaineers rushing up the mountain. After some time, the mountaineers that were least successful in terms of the height reached, are dropped from the race. The race however continues with the rest of the mountaineers while the climbing gets increasingly difficult. This process of less successful mountaineers dropping out is continued until only one mountaineer is left. If the top of the mountain is reached with more than one mountaineer still in the race, a random selection is made between the mountaineers to determine the winner. In our case the different combinations of parameter settings are the mountaineers, while the mountain is the amount of data used for training purposes.

WPS is based on progressive sampling, a process starting through selection of small samples and progressively increasing them as long as model accuracy improves. The term "wrapped" in WPS denotes that the feature selection algorithm exists as a wrapper around the induction algorithm [67]. Having access to massive amounts of data does not necessarily imply that the algorithms must use them all. Samples often provide the same accuracy with far less computational cost [60].

5.4.2 Determining the sizes of the progressive data sets[5]

WPS starts with a data set of training instances, used as basis for sampling smaller training and evaluation sets. 80% of the original data set is used for training, while the rest of the data set is used for evaluation. An exponential series of data sets is generated according to the following three step procedure:

[5] The information presented in subsections 5.4.2 and 5.4.3 is from Daelemans and Van den Bosch [60], unless stated otherwise.

Step 1: Let n be the number of instances in the data set used for training. An exponential sequence of 20 data sets is created by using a factor f as in Equation 5.17.

$$f = \sqrt[20]{n} \tag{5.17}$$

Step 2: A sequence of $i = \{1 \ldots d\}$ data sets is generated containing $size_i$ number of instances each. For i=1, $size_1$ = 1 and then for every i>1, the number of instances contained is defined as:

$$size_i = size_{i-1} * f \tag{5.18}$$

Step 3: The data sets are then limited to those containing more than 500 instances. A data set of 500 instances is used as the first set. An evaluation set, 20% the size of every generated data set, is also generated for every data set. The evaluation sets are extracted from the 20% of the original database set aside for evaluation.

5.4.3 PROCEDURE

WPS is performed over the sequence of data sets generated, and operates on a pool of algorithm settings. This pool of algorithm settings contains all the possible settings of the involved algorithm, referred to as $s_1 \ldots s_c$. C is the total number of possible parameter settings available for the algorithm. The WPS process can be split into three steps:

Step 1: WPS is used to perform experiments with all the possible settings ($s_1 \ldots s_c$) on the first data set (the data set containing 500 instances), while evaluating with the evaluation set generated for the first training set (the evaluation set containing 100 instances, 20% the size of the first data set of 500 instances). The performance of each setting is judged on the number of correctly classified evaluation instances.

Step 2: Next, badly performing settings are removed from the pool of algorithm settings (i.e. the least successful mountaineers are dropped from the race). This editing of badly-performing settings from the pool of settings should be very carefully done by the WPS algorithm. The danger here is that settings that perform badly on a small amount of training

data, but perform better on larger amounts of training data can be wrongfully removed from the pool of settings. Instead of sorting the accuracies obtained through each parameter setting and cutting away the lower half for example, WPS attempts to estimate the subset of accuracies that stands out as the best performing group. This is done through dividing all the accuracies into 10 equally sized bins, $b_1 \ldots b_{10}$. The number of unique accuracies in each bin is $|b_i|$. The bins selected for the next step is determined in the following way: The bin with the highest accuracies is taken as the first selected bin. Every preceding bin that represents an equal number of settings or more than its subsequent bin is also selected. This selections of preceding bins is halted as soon as $|b_i| < |b_{i+1}|$.

Step 3: The current training and evaluation sets are discarded next, and replaced by their bigger-sized successors (i.e. the mountain gets higher, the climbing more difficult and the mountaineers start dropping out). The remaining parameter settings are then evaluated on these bigger data sets, and step 2 is repeated. This process goes on until one of the following conditions is met:

1. Only one setting is left;
2. More than one setting is left after training and evaluating with the largest available data set. The TiMBL default algorithm setting is selected if it is among the settings that are left; otherwise a random selection is made.

Paramsearch is evaluated in the next section to determine its suitability for determining the best data representation and algorithmic parameters for *Lia*.

5.4.4 PARAMSEARCH EVALUATION

A software package employing WPS, called *Paramsearch 1.0* (Beta) [21], is currently available for the purpose of determining the best algorithmic parameters for a certain classifier. Unfortunately IB1 and IGTree are the only TiMBL algorithms supported by *Paramsearch*, which turned out to be unsatisfactory in the sense that we are also interested in the performance of other algorithms.

Experiments with *Paramsearch* on *Lia's* data proved to be very fast, but the problem is that the differences between the sizes of the progressive data sets generated by *Paramsearch* at the start of the WPS procedure are too small. These small differences between the sizes of the progressive data sets cause a large number of parameter settings to be filtered out early on in the WPS process. Table 11 serves as an illustration of the problem relating to the relation between data set size and number of parameter settings evaluated. The sizes of the progressive data sets were generated according to equations 5.17 and 5.18 in Section 5.4.2.

Data set Number	Number of Instances in the Data set	Number of Parameter Settings Evaluated
1	500	925
2	720	232
3	1 245	13
4	2 154	3
5	3 727	3
6	6 448	1

Table 11: Relation between data set size and number of parameter setting evaluated by *Paramsearch*

Table 11 indicates that 99% of the parameter settings were filtered out based on a training set containing only 2% of the available instances. The problem therefore is that too many parameter settings are filtered out at the beginning of the process when the data set sizes are still very small. Some of these parameter settings that are filtered out in the beginning could possibly include settings that could have performed very well later on in the process when more data is used for evaluation, as will be illustrated in Section 5.4.5.

To overcome the problems related to the small differences in the progressive data sets and the fact that *Paramsearch* can only be used for two of the five TiMBL algorithms, we created our own implementation of *Paramsearch*, which we call *PSearch* [68]. *PSearch* was programmed in Perl and works basically on the same principles as the original *Paramsearch*, with the major difference being that *PSearch* can utilise all of the TiMBL algorithms. Another difference is in the way in that the sizes of the training and evaluation sets are generated (i.e. the height of the checkpoints on the mountain).

The approach followed by *PSearch* is therefore to discard equations 5.17 and 5.18 for the generation of the progressive data set sizes and instead manually define the sizes of the progressive data sets based on prior knowledge of the available data set. Table 12 displays the sizes of the progressive data sets, as well the number of parameter settings evaluated by *PSearch*.

Data set Number	Number of Instances in the Data set	Number of Parameter Settings Evaluated
1	800	925
2	1 500	630
3	2 000	238
4	4 000	135
5	15 000	114
6	30 000	72
7	40 000	33
8	57 781	6

Table 12: Relation between data set size and number of parameter setting evaluated by *PSearch*

Compared to Table 11, Table 12 shows a more gradual decrease in the number of evaluated parameter settings. Table 12 therefore represents a much more desirable situation, as the chances of good performing parameter settings being eliminated at the beginning of the process are decreased.

5.4.5 COMPARING *PARAMSEARCH* AND *PSEARCH*

The purpose of this subsection is to compare the performance of *Paramsearch* and *PSearch*. Both *Paramsearch* and *PSearch* were used to generate a set of combinational settings that are expected to do well on the lemmatisation task. The comparative evaluation was done for the IB1 algorithm only, with feature-aligned data containing 20 features as data set. We also performed an exhaustive search throughout all of the 925 possible combinational parameter settings for IB1, to serve as basis for the comparison of the performance of *PSearch* and *Paramsearch*. The five best combinational settings that *Paramsearch* and *PSearch*

respectively yielded are displayed in descending order in Table 13 and 14, together with their rankings according to the exhaustive search. The exhaustive search ranking signifies a position out of 925, as determined by the rankings of the algorithmic parameters produced in an exhaustive search. A ranking of 1 signifies the best combinational setting out of 925, while a ranking of 925 signifies the combinational setting with the lowest accuracy score. For example the setting with IB1 as classification algorithm, Jeffrey divergence as distance metric, frequency threshold of 2, Gain Ratio as feature-weight, 5 nearest neighbours and Inverse Linear class voting weight was ranked by *Paramsearch* as the best performing parameter setting, while the exhaustive search in fact indicated that this setting is ranked at position 271 out of 925 possible settings.

Method	Classification Algorithm	Distance Metric	Frequency Threshold	Feature-weighting	Number of Nearest Neighbours	Class Voting Weights	Ranking (Paramsearch	Ranking (Exhaustive Search)
Paramsearch	IB1	Jeffrey	2	Gain Ratio	5	Inverse Linear	1	271
Paramsearch	IB1	Jeffrey	2	Gain Ratio	7	Inverse Linear	2	149
Paramsearch	IB1	Jeffrey	1	Gain Ratio	7	Inverse Linear	3	154
Paramsearch	IB1	Jeffrey	1	Gain Ratio	9	Inverse Linear	4	88
Paramsearch	IB1	Jeffrey	2	Gain Ratio	9	Inverse Linear	5	89

Table 13: Top 5 parameter settings produced by *Paramsearch*

Method	Classification Algorithm	Distance Metric	Frequency Threshold	Feature-weighting	Number of Nearest Neighbours	Class Voting Weights	Ranking (PSearch)	Ranking (Exhaustive Search)
PSearch	IB1	MVDM	2	Information Gain	11	Inverse Linear	1	1
PSearch	IB1	MVDM	1	Information Gain	11	Inverse Linear	2	2
PSearch	IB1	Jeffrey	2	Information Gain	11	Inverse Linear	3	7
PSearch	IB1	Jeffrey	1	Information Gain	9	Inverse Linear	4	11
PSearch	IB1	Jeffrey	2	Information Gain	9	Inverse Linear	5	12

Table 14: Top 5 parameter settings produced by *PSearch*

The results in Table 13 and 14 indicate that *PSearch* delivers combinational settings with higher rankings than *Paramsearch*. *PSearch* even delivered the combinational settings ranked 1 and 2, which shows that *PSearch* is more suitable than *Paramsearch* for producing combinational settings for the task of Afrikaans lemmatisation.

The results from the Wilcoxon Signed-rank test indicate that the accuracy figure obtained with the best setting that *PSearch* delivered (*Mdn* = 92,825) is significantly higher than the result obtained with the best setting obtained with *Paramsearch* (*Mdn* = 92,092) (T = 0; p<0,05; r = -0,886).

Speed is the one aspect where *Paramsearch* delivers better results than *PSearch*. Table 15 shows a comparison of the execution time of *Paramsearch* and *PSearch* when finding the best algorithmic parameters for IB1 on feature-aligned data with 20 features.

	Execution Time (in seconds)
Paramsearch	525,34
PSearch	1768,39

Table 15: Comparing the speed of *Paramsearch* and *PSearch*

Table 15 indicates that the execution time of PSearch is more than 3 times the execution time of Paramsearch. This relatively large difference in execution time seems to be trivial when compared to the execution time of an exhaustive search. An exhaustive search throughout all of the possible parameter combinations for IB1 (on the same data set used for the comparison of Table 15) took 176 hours and 28 minutes to complete, which is much longer than the execution time of *PSearch*. The 20 minute difference in execution time between *PSearch* and *Paramsearch* therefore seems to be acceptable considering the fact that *PSearch* is more likely to produce better performing algorithmic parameter settings than *Paramsearch*.

5.5 Finding The Best Data Representation and Algorithmic Parameters for *LIA*

The objective of this subsection is to determine the best data representation and set of algorithmic parameters for all TiMBL algorithms. The objective of this section is therefore to specifically address **Research Question 3 (Which algorithmic parameters deliver the best performance in terms of linguistic accuracy, execution time and memory usage?)**. The results of Section 5.4.5 provide motivation for using *PSearch* rather than *Paramsearch* for the experiments in this section. *PSearch* was not used to determine the best algorithmic parameters of IGTree, since IGTree only has 5 possible parameter settings. This limited number of available parameter settings makes it possible to determine the best algorithmic parameters of IGTree by means of an exhaustive search throughout all of the possible parameter options.

PSearch was therefore only used to determine the 10 best parameter combinations of IB1, IB2, TRIBL and TRIBL2 for each of the different data representations. We have five different additional features (i.e. feature-aligned, syllable count, syllables, last letter probability and morpheme probability) along with four different numbers of features (i.e. 12,20,30 and 38), resulting in 20 different experiments for each of the four algorithms. A

total of 80 experiments are thus performed. These 10 best parameter combinations that *PSearch* yielded were used to construct classifiers using all of the available training data. 10-fold Cross-validation was performed on each of the ten classifiers and the average F-Score, AUC, Accuracy and Execution time are computed. The results of these 4 experiments are separately displayed in Table 16 to 19. We do not include memory usage in Table 16 to 19, since the two most important performance metrics, accuracy and execution time represent adequate information for determining the best data representation for each of the four algorithms.

The values of the parameters used in the evaluation are displayed in the following list:

1) Distance Metrics (Overlap, MVDM, Jeffrey Divergence)
2) Feature-weights (No Weighting, Gain Ratio, Information Gain, Chi-squared, Shared Variance)
3) Number of nearest neighbours (1,3,5,7,9,11,13,15,19,25,35)
4) Frequency Threshold (1,2)
5) IB2 Initial number of instances (50% of the available training data)
6) Class Voting Weights (Normal majority voting, Inverse Distance weighting, Inverse Linear weighting, Exponential Decay weighting)
7) TRIBL offset, the index number of the feature (counting from 1) after which TRIBL should switch from IGTree to IB1 (13 features - 3,6,9; 20 features - 5,10,15; 30 features - 7,15,23; 38 features - 9,19,28)

Algorithm	Data Representation	Average F-score	Average AUC	Accuracy Accuracy	Average Execution time
IB1	Feature-aligned 13	0,810	0,851	0,853	2,008
IB1	Feature-aligned 20	0,923	0,946	0,928	17,590
IB1	Feature-aligned 30	0,922	0,946	0,927	29,980
IB1	Feature-aligned 38	0,921	0,945	0,926	30,989
IB1	Syllable count 13	0,919	0,943	0,924	25,969
IB1	Syllable count 20	0,921	0,945	0,926	42,924
IB1	Syllable count 30	0,921	0,946	0,927	59,500
IB1	Syllable count 38	0,922	0,946	0,927	66,467
IB1	Syllables 13	0,909	0,939	0,915	87,705
IB1	Syllables 20	0,910	0,940	0,916	120,733
IB1	Syllables 30	0,907	0,938	0,913	180,433
IB1	Syllables 38	0,909	0,938	0,914	232,317
IB1	Last letter probability 13	0,919	0,943	0,924	12,856
IB1	Last letter probability 20	0,921	0,945	0,926	26,802
IB1	Last letter probability 30	0,923	0,946	0,928	29,980
IB1	Last letter probability 38	0,923	0,946	0,927	35,915
IB1	Morpheme probability 13	0,891	0,929	0,893	18,317
IB1	Morpheme probability 20	0,906	0,938	0,909	25,010
IB1	Morpheme probability 30	0,906	0,938	0,909	30,851
IB1	Morpheme probability 38	0,908	0,940	0,911	39,624

Table 16: Results obtained with *PSearch* for IB1

Algorithm	Data Representation	Average F-score	Average AUC	Average Accuracy	Average Execution time
TRIBL2	Feature-aligned 13	0,892	0,922	0,898	6,184
TRIBL2	Feature-aligned 20	0,899	0,930	0,903	8,852
TRIBL2	Feature-aligned 30	0,909	0,936	0,914	29,421
TRIBL2	Feature-aligned 38	0,911	0,938	0,916	33,562
TRIBL2	Syllable count 13	0,890	0,921	0,895	7,757
TRIBL2	Syllable count 20	0,894	0,926	0,899	10,217
TRIBL2	Syllable count 30	0,902	0,932	0,906	13,106
TRIBL2	Syllable count 38	0,904	0,933	0,908	15,453
TRIBL2	Syllables 13	0,889	0,921	0,896	12,283
TRIBL2	Syllables 20	0,892	0,923	0,898	15,972
TRIBL2	Syllables 30	0,899	0,929	0,904	21,361
TRIBL2	Syllables 38	0,899	0,930	0,904	22,798
TRIBL2	Last letter probability 13	0,919	0,943	0,925	11,907
TRIBL2	Last letter probability 20	0,921	0,945	0,926	18,753
TRIBL2	Last letter probability 30	0,920	0,944	0,926	23,699
TRIBL2	Last letter probability 38	0,921	0,945	0,926	30,727
TRIBL2	Morpheme probability 13	0,887	0,927	0,888	9,591
TRIBL2	Morpheme probability 20	0,890	0,930	0,892	20,621
TRIBL2	Morpheme probability 30	0,890	0,930	0,892	29,962
TRIBL2	Morpheme probability 38	0,891	0,931	0,893	34,011

Table 17: Results obtained with *PSearch* for TRIBL2

Algorithm	Data Representation	Average F-score	Average AUC	Average Accuracy	Average Execution time
TRIBL	Feature-aligned 13	0,883	0,917	0,892	3,509
TRIBL	Feature-aligned 20	0,891	0,922	0,897	5,209
TRIBL	Feature-aligned 30	0,901	0,930	0,907	7,370
TRIBL	Feature-aligned 38	0,904	0,932	0,910	8,888
TRIBL	Syllable count 13	0,884	0,919	0,893	5,667
TRIBL	Syllable count 20	0,890	0,922	0,897	6,848
TRIBL	Syllable count 30	0,896	0,927	0,902	9,432
TRIBL	Syllable count 38	0,897	0,926	0,903	10,413
TRIBL	Syllables 13	0,904	0,932	0,910	12,224
TRIBL	Syllables 20	0,887	0,921	0,893	13,493
TRIBL	Syllables 30	0,889	0,923	0,896	12,798
TRIBL	Syllables 38	0,889	0,923	0,896	13,238
TRIBL	Last letter probability 13	0,900	0,928	0,905	5,929
TRIBL	Last letter probability 20	0,541	0,705	0,605	5,592
TRIBL	Last letter probability 30	0,541	0,705	0,605	7,718
TRIBL	Last letter probability 38	0,857	0,899	0,865	9,872
TRIBL	Affix probability 13	0,686	0,791	0,727	4,541
TRIBL	Affix probability 20	0,686	0,792	0,726	5,756
TRIBL	Affix probability 30	0,686	0,792	0,726	7,966
TRIBL	Affix probability 38	0,686	0,792	0,727	9,802

Table 18: Results obtained with *PSearch* for TRIBL

Algorithm	Data Representation	Average F-score	Average AUC	Average Accuracy	Average Execution time
IB2	Feature-aligned 13	0,908	0,936	0,924	177,863
IB2	Feature-aligned 20	0,911	0,938	0,924	215,397
IB2	Feature-aligned 30	0,913	0,939	0,926	326,663
IB2	Feature-aligned 38	0,914	0,941	0,927	340,288
IB2	Syllable count 13	0,907	0,935	0,921	233,797
IB2	Syllable count 20	0,910	0,938	0,925	321,727
IB2	Syllable count 30	0,910	0,938	0,925	394,205
IB2	Syllable count 38	0,908	0,937	0,923	409,411
IB2	Syllables 13	0,898	0,930	0,915	315,234
IB2	Syllables 20	0,902	0,933	0,917	431,366
IB2	Syllables 30	0,898	0,930	0,914	1239,375
IB2	Syllables 38	0,897	0,929	0,913	1983,447
IB2	Last letter probability 13	0,902	0,933	0,916	78,723
IB2	Last letter probability 20	0,909	0,937	0,921	123,798
IB2	Last letter probability 30	0,893	0,924	0,909	137,564
IB2	Last letter probability 38	0,910	0,938	0,922	163,891
IB2	Affix probability 13	0,891	0,928	0,902	105,823
IB2	Affix probability 20	0,894	0,931	0,904	166,258
IB2	Affix probability 30	0,888	0,928	0,900	178,619
IB2	Affix probability 38	0,892	0,930	0,903	239,829

Table 19: Results obtained with *PSearch* for IB2

Unlike the results in Table 16 to 19, which were based on the ten best performing algorithmic settings for the various classification algorithms and feature representations, Table 20 displays the results obtained with an exhaustive search throughout all of the possible parameter settings for IGTree for the 20 different data representations. The exhaustive search was possible since IGTree is a relatively fast algorithm with only five different combinations of algorithmic parameter settings.

Algorithm	Data Representation	Average F-score	Average AUC	Average Accuracy	Average Execution time
IGTree	Feature-aligned 13	0,779	0,840	0,795	6,371
IGTree	Feature-aligned 20	0,780	0,841	0,796	11,842
IGTree	Feature-aligned 30	0,784	0,844	0,801	20,935
IGTree	Feature-aligned 38	0,785	0,844	0,801	30,014
IGTree	Syllable count 13	0,801	0,859	0,815	9,416
IGTree	Syllable count 20	0,805	0,862	0,818	20,758
IGTree	Syllable count 30	0,808	0,864	0,821	25,217
IGTree	Syllable count 38	0,809	0,864	0,825	38,923
IGTree	Syllables 13	0,801	0,859	0,815	9,416
IGTree	Syllables 20	0,804	0,862	0,818	20,758
IGTree	Syllables 30	0,808	0,864	0,821	25,217
IGTree	Syllables 38	0,809	0,864	0,822	38,923
IGTree	Last letter probability 13	0,482	0,618	0,596	6,544
IGTree	Last letter probability 20	0,482	0,619	0,596	9,604
IGTree	Last letter probability 30	0,482	0,619	0,596	13,349
IGTree	Last letter probability 38	0,482	0,619	0,596	16,751
IGTree	Affix probability 13	0,637	0,743	0,695	25,627
IGTree	Affix probability 20	0,636	0,743	0,694	9,414
IGTree	Affix probability 30	0,636	0,743	0,695	13,282
IGTree	Affix probability 38	0,637	0,743	0,695	16,618

Table 20: Exhaustive Search Results for IGTree

Table 21 indicates the best data representations, together with the best combination of algorithmic parameter settings for each of the classification algorithms selected from Table 16 to 20. The best data representation was selected on linguistic accuracy, since the objective of this study is first off all to construct a lemmatiser that achieves the highest possible accuracy rate. Average Execution time was the deciding factor when a tie (based on average accuracy) occurred between two or more data representations. The best performing parameter settings for each of the 5 classification algorithms, as computed by *PSearch* (except the parameter setting for IGTree) are also displayed in Table 21. For example, the best performing parameters for IB1 (MVDM distance metric, Information Gain weighting, 11 nearest neighbours, a frequency threshold of 2 and Inverse Linear class weights), was

determined by selecting the best performing combination of parameter settings for IB1 with feature-aligned data with 20 features.

Algorithm	Data Representation	Distance Metric	Feature-weight	Nearest neighbours	Frequency Threshold	Class Weights	TRIBL Offset
IB1	Feature-aligned 20	MVDM	Information Gain	11	2	Inverse Linear	-
IB2	Feature-aligned 38	Jeffrey	Information Gain	15	2	Inverse Linear	-
TRIBL	Feature-aligned 38	Jeffrey	Gain Ratio	5	2	Inverse Distance	27
TRIBL2	Last letter probability 20	MVDM	Shared Variance	7	1	Inverse Linear	-
IGTree	Syllable count 38	Overlap	Information Gain	-	-	-	-

Table 21: The best data representation for the different TiMBL algorithms

The different algorithmic parameters obtained for each of the five different settings in Table 21 are now evaluated in on the full data set comprising 72 226 instances. Ten-fold cross-validation was used in the evaluation and the average linguistic accuracy figures and execution time were computed.

Algorithm	Average Accuracy	Average Execution Time (in seconds)	Memory usage (MB)
IB1	0,928	18,317	14,888
IB2	0,928	350,171	10,305
IGTree	0,903	14,810	0,267
TRIBL2	0,927	18,197	18,717
TRIBL	0,911	9,162	22,552

Table 22: Results obtained with the best parameter settings for the different algorithms

Table 22 indicates that IB2 has an execution time of 350,171 seconds, which deviates from the execution times of the other algorithms. This long execution time can be ascribed to the incremental addition of misclassified instances to memory (see Section 5.2.2). Another value that clearly deviates from the other in

Table 22 is the memory usage of IGTree. This much reduced figure can be accounted to the optimised storage of training instances in a tree structure as employed by IGTree (see Section 5.2.3).

The 5 algorithms (trained with their best parameter settings and data representations) are now ranked according to the accuracies achieved on the Afrikaans lemmatisation task. Remember that it was indicated in Chapter 1 that the performance metrics are considered in the following order of importance:

a. Accuracy
b. Execution time
c. Memory usage

The ranks produced by Friedman's ANOVA are displayed in
Table 23. The results produced by Friedman's ANOVA indicate that there is a significant difference between the accuracy scores obtained with the different classification algorithms (F = 32,240; p<0,05; df = 4). Results obtained with the Wilcoxon Signed-rank test indicates that significant differences exist between all the of the algorithms except for IB1 and IB2 (T = 21,5; p<0,005; r = -0,19), IB1 and TRIBL2 (T = 11; p<0,005; r = -0,531), and IB2 and TRIBL2 (T = 17; p<0,005; r = -0,338). A Bonferroni correction was applied and therefore all the effects are reported at a significance level of 0,005.

Algorithm	Mean Rank
IB1	4,20
IB2	3,90
IGTree	1,00
TRIBL2	3,90
TRIBL	2,00

Table 23: Ranks for the different classification algorithms

The results from the Wilcoxon Signed-rank test indicate that there are no significant differences in the accuracy figures obtained by the top 3 algorithms, namely IB1, IB2 and TRIBL2. In order to decide on the best algorithm between IB1, IB2 and TRIBL2, the execution time was considered.

Table 22 indicates that the execution time of IB2 (350,171 s) is much higher than the execution time of IB1 (18,317 s) and TRIBL2 (18,197 s), and IB2 is therefore not further considered in the process of finding the best algorithm for Afrikaans lemmatisation.

In order to make an informed selection (based on execution time) between IB1 and TRIBL2, another Wilcoxon Signed-rank test was computed to determine if significant differences in the execution times of IB1 and TRIBL2 exist. Results from the Wilcoxon Signed-rank test indicate that there is no significant difference between the execution times of IB1 (Mdn = 18,223) and TRIBL (Mdn = 18,188), T = 0, p>0,05, r = -0,142). There is no significant difference between the accuracies and the execution times of IB1 and TRIBL2; therefore we were forced to decide between IB1 and TRIBL on the basis of memory usage.

Table 22 indicates that IB1 uses significantly less memory than TRIBL2, which indicates that IB1 is the best algorithm for Afrikaans lemmatisation. The difference in memory usage between IB1 and TRIBL2 can be ascribed to the fact that TRIBL was trained on data with more features (feature-aligned data with 38 features) than the data on which IB1 was trained (feature-aligned data with 20 features).

Based on the preceding comparisons and statistical significance tests, the classification algorithms can be ranked as in
Table 24 in terms of suitability for Afrikaans lemmatisation based on linguistic accuracy. The rankings of
Table 24 were based on accuracy, while execution time and memory usage were used to distinguish between options when no significant differences existed. Note that Table 24 indicates the ranking of the classification algorithms and not the 5 best classifiers.

Rank	Classification Algorithm	Data Representation	Distance Metric	Feature-Weight	Nearest neighbours	Frequency Threshold	Class Weights	TRIBL Offset
1	IB1	Feature-aligned 20	MVDM	Information Gain	11	2	Inverse Linear	-
2	TRIBL2	Last letter probability 20	MVDM	Shared Variance	7	1	Inverse Linear	-
3	IB2	Feature-aligned 38	Jeffrey	Information Gain	15	2	Inverse Linear	-
4	TRIBL	Feature-aligned 38	Jeffrey	Gain Ratio	5	2	Inverse Distance	27
5	IGTree	Syllable count 38	Overlap	Information Gain	-	-	-	-

Table 24: Rankings based on linguistic accuracy

Table 24 indicates that IB1 trained on feature-aligned data with MVDM as distance metric, Information Gain as feature weight, 11 nearest neighbours, a frequency threshold of 2 and Inverse Linear class weights achieve the highest accuracy figure of 92,8%. The high accuracy figure of the classifier is confirmed by the high figures for F-Score (0,923) and AUC (0,946) that the classifier obtained.

5.6 Conclusion

This chapter commenced with an introduction to the classification algorithms available in TiMBL. We first described the two instance-based algorithms, IB1 (that is the basic instance-based algorithm) and IB2, the incremental editing algorithm. This was followed by descriptions of IGTree, the decision tree-based algorithm, and the two hybrid algorithms TRIBL and TRIBL2. The introduction of the classification algorithms was followed by an overview of the various algorithmic parameter settings available in TiMBL. We first considered 4 feature-weighting possibilities that included *Information-Gain*, *Gain Ratio*, *Chi-squared* and *Shared Variance* weighting. Descriptions of the distance metrics, namely *Overlap*, *Modified Value Difference* and *Jeffrey Divergence* were next provided. We then

proceeded to explain the process of class voting in TiMBL, where we focused on normal majority voting and various methods of distance weighted class voting. We also discussed the frequency threshold parameter for *MVDM* and *Jeffrey divergence*. We also clarified the classification process in the event of the occurrence of a tie among two or more neighbours in the instance space.

Section 5.4 introduced the process of Wrapped Progressive Sampling, where we explained its operation as analogous to the event of a race between mountaineers to reach the top of a mountain. WPS forms the basis for the operation of *Paramsearch*, the software package that performs automatic parameter selection. *Paramsearch* was unfortunately not available for all the TiMBL classification algorithms, so we had to write our own implementation of *Paramsearch*, named *PSearch*, in order to enable us to use it for all of the TiMBL algorithms. *Paramsearch* also did not utilize the relatively large data set of *Lia* to its full potential, as too many algorithmic parameter combinations that could possibly have performed well on the complete data set were filtered out early on in the process. We therefore proposed a method for generating larger data sets early on in the process to improve the performance of *Paramsearch*. This method for generating larger data sets was accordingly implemented in *PSearch*, before a comparison was done between *PSearch* and *Paramsearch* on the basis of the results delivered by an exhaustive search to determine if *PSearch* delivered better results than *Paramsearch*. *PSearch* delivered higher ranking algorithmic parameter combinations than *PSearch*. We also indicated that a statistically significant difference exists between the accuracy figures obtained with the best combination of algorithmic parameter settings that *PSearch* and *Paramsearch* yielded.

The chapter concluded with Section 5.5 where **Research Question 3 (Which algorithm and parameters settings deliver the best performance in terms of linguistic accuracy, execution time and memory usage?)** was addressed. *PSearch* was used to determine the best data representation and algorithmic parameters for the IB1, IB2, TRIBL and TRIBL2 algorithms. The best data representation and algorithmic parameters for IGTree was determined by means of an exhaustive search. We determined that the classification algorithms can be ranked (on the basis of accuracy scores) in the following order: IB1, TRIBL2, IB2, TRIBL, IGTree. The best classifier (considering accuracy as the most

important performance metric) was IB1 trained on feature-aligned data with MVDM as distance metric, Information Gain as feature weight, 11 nearest neighbours, a frequency threshold of 2 and Inverse Linear class weights. This classifier obtained the following results: Accuracy - 92,8%; F-Score - 0,923; AUC - 0,946; Memory-usage - 10,305 MB; Execution time – 18,317 seconds.

We indicated that alternative rankings than those based on accuracy can be produced by assigning weights to the different performance metrics. The order of the rankings of the classification algorithms is dependent on the performance metrics considered to be the most important for the classification task, since the top settings of the classification algorithms all produced classifiers with accuracy figures of above 90%.

Chapter 6: Conclusion and Future Directions

6.1 Summary

The availability of comprehensive Human Language Technology (HLT) resources is of cardinal importance to the survival of any language in the technological era. South Africa has 11 official languages and therefore the promotion of multilingualism is a very important focus of various South African organisations such as universities and the government. In spite of this strong focus on multilingualism, the fields of HLT and Natural Language Processing (NLP) are relatively unexplored research territories in South Africa. It is therefore not surprising that the 11 languages of our country lag behind other languages such as English and Dutch in terms of the availability of HLT resources. The main theme of this research has been the development of an automatic lemmatiser for Afrikaans, which forms part of the effort to bridge the digital divide between the resource-scare languages of South Africa and other languages that have a large number of available HLT resources.

The automatic lemmatiser developed in this study is named *Lia* ("*Lemma-identifiseerder for Afrikaans*" 'Lemmatiser for Afrikaans'). In order to construct *Lia*, the following research objectives were set in Chapter 1:

1) To define the classes for Afrikaans lemmatisation;
2) To determine the influence of the data size and various feature options on the performance of the system; and
3) To automatically determine the algorithm and parameter settings that deliver the best performance in terms of linguistic accuracy, classification time and memory usage.

Considering the results obtained for lemmatisation in various other languages and the extent of the lemmatisation task in Afrikaans, we defined a desired accuracy figure of 90% for Afrikaans lemmatisation.

Chapter 2 provided information on memory-based learning in general. The architecture of *Lia* was presented in graphical format and the lemmatisation task was related to memory-based learning by extending Mitchell's' definition [20] of machine learning to lemmatisation learning. It was motivated that lemmatisation is ideally suited to be resolved by means of memory-based learning, since the lemmatisation task is complex with many exceptions from a linguistic point of view. The operation of the most basic memory-based learning algorithm, the *k*-NN algorithm, was explained to provide insight into the operation of memory-based learning in general. The strengths and weaknesses of memory-based learning were also discussed to provide a better idea of the problems that may be encountered in the process of constructing an automatic lemmatiser for Afrikaans.

Chapter 2 concluded with a section where TiMBL, the memory-based learning system applied in the development of *Lia,* was introduced. TiMBL has a large variety of different classification algorithms and parameter options available that can be set in the construction of the classifier. Determining the algorithm and parameter options that deliver the best results in terms of linguistic accuracy, classification time and memory usage formed one of the 3 objectives of this study.

The purpose of Chapter 3 was to define the classes for Afrikaans lemmatisation, which was also the first objective of this study. The chapter started with an investigation into the process of inflection in Afrikaans, since automatic lemmatisation requires a clear distinction between inflectional and derivational affixes. Distinguishing between inflectional and derivational affixes was however not a simple process, since there is no agreement among Afrikaans linguists on the affixes that should be considered to be of an inflectional nature. We therefore provided a short overview of the viewpoints of a number of authoritative linguists on the subject of inflection and the categories thereof in Afrikaans. We accordingly proceeded to define every inflectional category mentioned by the linguists for the purposes of this study.

The focus of Chapter 4 was initially on the various facets of *Lia*'s training data. We explained how the training data for *Lia* was generated, since we did not have training data available in the desired format (due to Afrikaans being a resource-scarce language) at the start of the project. The training data for *Lia* was generated by extracting the word-forms from CTexT's

Afrikaans lexicon, which correspond to the affixes identified by the defined inflectional categories. The extracted word-forms were then manually annotated by providing the linguistic accurate lemma for each word-form.

The extent of the challenge presented to *Lia* was realised when we considered the fact the *Lia* is expected to achieve an accuracy figure of 90%, in spite of the various complexities and exceptions that the lemmatisation task in Afrikaans entails. One of the problems experienced in the development of *Lia* was a lack of training data. It is required of *Lia* to achieve the highest possible accuracy figure (or at least an accuracy figure of 90%), while being trained with the smallest amount of training data. We made a prediction, based on a logarithmic trendline inserted into a graph of accuracy versus the number of instances in the training data, that *Lia* would require a training dataset containing approximately 100 000 instances to achieve an accuracy figure of 90%. We did not have the resources available to annotate more data; therefore we considered alternative options other than enlarging the number of training instances in the dataset to achieve the standard of 90%.

We experimented with various feature options to determine if this could improve the accuracy of *Lia*. The influence of appending the features on the performance of *Lia* in terms of accuracy, classification time and memory usage was determined by evaluating the feature options by training *Lia* with the default algorithmic parameter settings of TiMBL. The purpose of these experiments was to address the second research objective of this study, namely to determine the influence of the data size and various feature options on the performance of the system. The results reported in Chapter 4 were based on the default algorithmic parameter settings of TiMBL, and therefore these results should not be generalised as valid for other algorithmic parameter settings.

Chapter 4 concluded with a visualisation of the nearest neighbour relations in an extraction of the training set. The visualisation of the training set showed definite structure in terms of the positioning of nodes with the same colour, which further motivated that the problem of lemmatisation in Afrikaans should be successfully resolved by means of MBL.

Chapter 5 addressed the third objective of this study, namely to automatically determine the algorithm and parameter settings that deliver the best performance in terms of linguistic accuracy, classification time and memory usage. The operation of the 5 classification algorithms and the various parameter options in TiMBL were described.

The process of Wrapped Progressive Sampling (WPS) was introduced as an efficient alternative to an exhaustive search for finding the data representation (for each classification algorithm) and the parameter settings that deliver the best performance in terms of the considered performance metrics. *Paramsearch*, a software package that implements WPS, was evaluated on *Lia*'s training data by comparing it with the results obtained from an exhaustive search. It was evident from these results that too many algorithmic parameter settings (that could possibly have performed well on larger datasets) were filtered out early on in the WPS process by *Paramsearch*. We improved on the results of *Paramsearch* by creating *PSearch*, our own program implementing WPS. The difference between *Paramsearch* and *PSearch* is that *PSearch* determines the sizes of the progressive datasets used in the WPS procedure on prior knowledge about the size of the available training set.

PSearch was used to obtain 10 (possibly optimal) parameter settings for every combination of algorithm (except for IGTree) and data representation that are estimated to do well in the lemmatisation task. IGTree was evaluated by means of an exhaustive search, since there was no need to perform WPS on IGTree, due to the small number of parameter options available to IGTree. These 10 settings were then evaluated on the full training set and the average results were reported. The best data representation and parameter settings for each classification algorithm were determined from these results.

6.2 Main Findings

Results obtained during this study indicate that the lemmatisation task in Afrikaans can be successfully resolved by means of memory-based learning, since an accuracy figure of 92,8% was achieved. This accuracy figure is well beyond the standard set for Afrikaans lemmatisation and compares well to the results achieved by similar studies [3,8,13]. The accuracy figure of 92,8% represents a error reduction percentage of 78,2% on the accuracy

figure obtained by *Ragel*, which indicates that memory-based learning is more suitable than rule-based methods for Afrikaans lemmatisation. This improvement in the accuracy figure can be attributed to the fact that memory-based learning has the ability to model complex target functions such as performing lemmatisation.

The **first research question** of this study, namely: "**What are the classes for Afrikaans lemmatisation?",** was addressed in Chapter 3. In order to define the classes, the following eight categories for inflection in Afrikaans were identified:

1) Plural (e.g. the *-s* in "*tafels*", 'tables' and the *-e* in "*mense*", 'humans');
2) Degrees of comparison (e.g. the *-er* and *-ste* in "*kleiner*" 'smaller' and "*kleinste*" 'smallest');
3) Diminutive (e.g. the *-jie* in "*hondjie*" 'puppy');
4) Past Tense (e.g. the *ge-* in "*geloop*" 'walked' and the -ge- in "*uitgeloop*" 'walked out');
5) Past Participle (e.g. the *ge- -te* in "*getrapte*" 'trampled');
6) Infinitive (e.g. the *-e* in "*drinke*" 'drink');
7) Attributive (e.g. the *-e* in "*pragtige*" 'exquisite'); and
8) Partitive Genitive (e.g. the *-s* in "*pragtigs*" 'exquisite').

These eight categories of inflection represent a number of inflectional affixes (see Addendum B). The training data was constructed by extracting the words that correspond to the inflectional affixes in the eight categories. The classes were derived on the basis of a comparison between the word-forms and their corresponding lemmas in the annotated training data. The complete list of derived classes, together with their frequency in the training data is provided in Addendum D.

Chapter 4 focused on the **second research question** of this study: "**What is the influence of the data size and various options on the performance of Lia?**". We indicated that the performance of *Lia* improved, but that the execution time and memory usage increased when the number of instances in the training data was enlarged. It was also indicated that each of the feature options had an effect on the accuracy figure of *Lia*.

Chapter 5 addressed the **third research question in** this study, namely "Which algorithm and parameter settings deliver the best **performance in terms of linguistic accuracy, execution time and memory usage?**". The results reported in this chapter indicated that *PSearch* is a useful tool for determining the algorithmic parameter settings and data representations that deliver the best results for Afrikaans lemmatisation. Enlarging the sizes of the progressive datasets at the start of the WPS process significantly improved the obtained results.

The number of features in the training data had a significant effect on the performance of *Lia*. Data containing 38 features proved to be the best data representation for IB2, TRIBL and IGTree, while data with 20 features proved to be the best for IB1 and TRIBL2. Appending additional features (i.e. features other than just letter sequences) to the training data also had a significant influence on the performance. In the cases of TRIBL2 (data appended with last letter probability information) and IGTree (data appended with information on the syllable count) we proved that additional features improved the performance in terms of accuracy.

The IB1 algorithm (trained with feature-aligned data with 20 features) proved to be the most effective of all the evaluated algorithms for Afrikaans lemmatisation in terms of linguistic accuracy, while IGTree (trained with syllable count appended data with 20 features) proved to be the most memory efficient algorithm.

It is generally hard to understand why certain parameter choices lead to better performance. In our study some indication of parameter choices leading to good results are given. However, we were not able to explicitly determine the exact effect of each parameter selection in a specific algorithm. This is a well known problem [66], the resolution of which can lead to the construction of better machine learning algorithms.

6.3 Future Directions

The following aspects could be considered in future:

- The training data should be further evaluated to improve the quality thereof. *Lia* can also be used to generate more training data that can in turn be used to further improve accuracy.

- *Lia* can be expanded to include dictionary lookup to evaluate the existence of a lemma before it is produced. Dictionary lookup procedures have improved the results of rule-based lemmatisers/stemmers like SteDL [9] and therefore we should experiment with dictionary lookup to determine if this can also improve the accuracy of *Lia*.
- Further research is necessary to see if the accuracy of the decision tree based classifiers can be improved. The results reported in Chapter 5 indicate that the IGTree algorithm is superior to the other algorithms in terms of execution time and memory usage, but it obtained the lowest accuracy figure of all the classification algorithms. A good starting point will be to experiment with the C4.5 decision tree algorithm, since the training data is already in C4.5 format.
- The features of *Lia* can be expanded to include part-of-speech tags, since similar studies in the past have shown that part-of-speech tags can significantly increase accuracy figures.
- Enlarging the sizes of the progressive datasets generated during the WPS procedure (the approach of *PSearch*), significantly improved the results obtained by *Paramsearch*. Further research is necessary to determine if this improvement in accuracy can also be achieved when alternative classification tasks than lemmatisation is considered.
- The sizes of the progressive datasets generated by *PSearch* are based on prior knowledge about the size of the available training data. The ideal solution would be to devise a formula for generating the sizes of the progressive datasets in order to make *PSearch* more generic.
- *Paramsearch* currently determines the best performing parameter settings on the basis of accuracy. *Paramsearch* can be expanded to determine the best performing parameter settings on the basis of weighted performance metrics as introduced in Section 5.5.
- *Lia* can be implemented in the spelling checker for Afrikaans to enhance the lexical recall ability (i.e. the number of words recognised as valid words). Another interesting application where *Lia* may be implemented is the WordNet for Afrikaans, which is currently being developed at the North-West University. WordNet is an electronic lexical database that is an important tool to researchers in computational linguistics, text analysis and other related areas [69].

- *PSearch* currently estimates a combination of possibly optimal algorithmic parameters based on the accuracy figures achieved by the classifier to which it is applied. *PSearch* can therefore be expanded to base its estimation on a trade-off score between accuracy, classification time and memory usage as in equation 5.20.

6.4 CONCLUSION

Lia delivers satisfactory results and can be used as a foundation for a variety of Afrikaans HLT applications such as morphological analysers and information extraction systems. This study contributes to the growth and development of Afrikaans in the technological environment, and to the general HLT and NLP industry in South Africa. The methodology used in the construction of *Lia* can in principle also be applied to create lemmatisers for other South African languages such as Setswana, Sesotho sa Lebowa, isiZulu and isiXhosa, in order to empower and contribute to the technological status of these languages.

ADDENDUM A

Technical Report: Manual for the annotation of training data for the development of *Lia*

SUMMARY

This document serves as a manual to generate data for the development of *Lia* (Lemmatiser for Afrikaans).

BACKGROUND

The purpose of this experiment is to describe the data annotation process for generating training data for the development of *Lia*. *Lia* deviates from the traditional rule-based methods of lemmatisation, as machine-learning techniques form the basis of *Lia*.

Lia will be an intelligent lemmatiser, which means she will be able to learn to perform lemmatisation. This consists of *Lia* being trained in the process of lemmatisation, and the more she is trained the smarter she becomes.

Thus, *Lia* will need extensive data with which to train her, which is precisely the purpose of this experiment. It is extremely important that the data used to train *Lia* is correct, as it can actively influence the accuracy of *Lia*. This project is inordinately important to the advancement of the technological status of Afrikaans. Each person participating in this experiment will thus be instrumental to the development of the first lemmatiser for Afrikaans.

METHOD

To each person participating in this experiment, a spreadsheet is supplied, consisting of two identical columns with words. The first column (column A in Figure 26) of the spreadsheet contains the words of which the respective lemmas must be indicated. The task is then to indicate the **lemma** of each word in the second column (column B in Figure 26), by changing the existing word through elimination and/or addition of a character(s), or leaving it

unchanged. Where there is any uncertainty concerning the meaning of a word or the lemma thereof, it is indicated by a question mark in column C. Any comments are inserted in column D.

Microsoft Excel - eersteduisendreggemaak

File Edit View Insert Format Tools Data Window Help

Arial 10 B I U $ % ,

F33 Rkies>g

	A	B
22	meisiekinders	meisiekind
23	dekspeletjie	dekspel
24	draffies	draffie
25	gordelbroekie	gordelbroek
26	barsies	bars
27	driepootpotjie	driepootpot
28	gebedjies	gebed
29	gebiedsuitsprake	gebiedsuitspraak
30	koffiesakkie	koffiesak
31	tekshakie	tekshakie
32	bokkies	bok
33	versterkinkies	versterking
34	getalle	getal
35	gebaretale	gebaretaal
36	maagwalletjie	maagwal
37	rolletjies	rol
38	geelkeurbome	geelkeurboom
39	strammer	stram
40	wissellammers	wissellam
41	ysblommetjie	ysblom
42	gevegsplanne	gevegsplan
43	uitgesprokener	uitgesproke
44	geesvermoëns	geesvermoë
45	dwarsskoppie	dwarsskop
46	armpies	arm
47	aandagstrepies	aandagstreep
48	gedweër	gedweë
49	buidelbere	buidelbeer

Figure 26: Example data that has been correctly annotated

INSTRUCTIONS

Lemmatisation comprises the reduction of words in a corpus, to their corresponding lexemes/lemmas. A lemma is the grammatically correct root of the word. To obtain a valid

Afrikaans word, there are eight different ways to transform lemmas by means of inflection, to wit:

1. Plurals (example "*tafels*")
2. Past tense (example "*geskop*")
3. Diminutives (example "*tafeltjie*")
4. Attributive -e (example "*interessante persoon*")
5. Infinitive -e (example "*iets te drinke*")
6. Partive genitive (example "*iets moois*")
7. Degrees of comparison (example "*mooier, mooiste*")
8. Past participle -t/-d (example "*gemengd, gelapt*")

The affixes represented by the eight above-mentioned categories of inflection (and which thus need to be removed to obtain the lemma), can be found in Addendum B.

Use the fourth edition of the "*Verklarende Woordeboek van die Afrikaanse Taal*" (HAT) [70] as a guideline if any doubts exist (see Figure 27). Another source would be the eighth rendition of the "*Verklarende Afrikaanse Woordeboek*" (VAW) [71]. The latter is available in electronic format, which facilitates the search process.

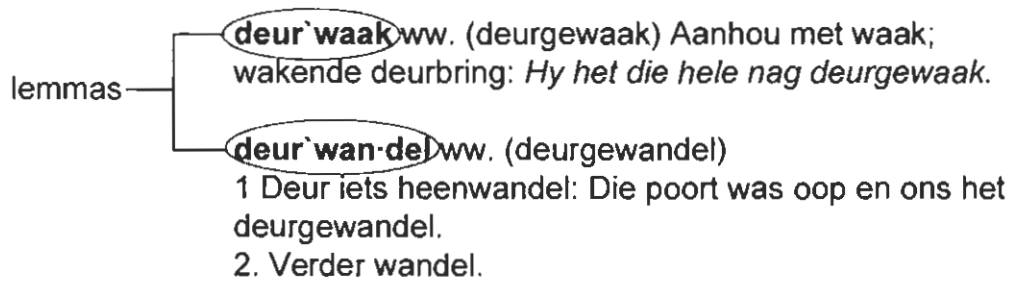

Figure 27: Extract from the HAT [70]

A golden rule to remember is that the part of speech of the original word may not differ from that of the lemma. This is especially evident in words such as "*speler*", 'player', where the person might indicate the lemma as "*speel*" 'play'. However, this cannot be permitted, as "*speler*" is a noun, which differs from the part of speech of "*speel*", which is a verb. Thus, the lemma of "*speler*" 'player' is "*speler*" 'player'.

DIMINUTIVES

There are seven categories by which diminutives' lemmas are identified:

1) Regular Diminutives (unaltered)

The diminutive forms of these lemmas are common knowledge to the average Afrikaans speaker (examples: "*plankie*" 'small plank', "*varkie*" 'small pig', "*kindjie*" 'small child', "*karretjie*" 'small car').

2) Lexicalised (unaltered)

These words were mostly meant to be diminutives in the original Dutch, but with time, the diminutive form has become lexicalised. As a result, these words are now only found as diminutives in Afrikaans (examples: "*polfyntjie*" 'souvenir', "*kolwyntjie*" 'cup-cake', "*uientjie*" 'onion', "*babetjie*" 'baby', "*dankie*" 'thank you').

3) Institutionalised (unaltered)

In this category, the words usually consist of a composition, coupled with the diminutive suffix. These words are commonly only used in their diminutive form (examples: "*mooinooientjie*" 'bamboo-fish', "*bobbejaantoontjie*" 'wild vine', "*middeloorbeentjie*" 'incus').

4) Utilised frequency (altered)

These words are commonly used in the diminutive form, but it would be correct to use them in the non-diminutive form (examples: "*hondepisbossie*" (a shrud species), "*karoobossie*" 'karoo bush', "*rosyntjie*" 'raisin', "*boontjie*" 'bean', "*klontjie*" 'small lump'). The word "*glippertjie*" (type of grape) should be included in this category. It would be correct to use the word "*glipper*", as it is a position on the cricket field. However, the word "*glipper*" 'slip' is antiquated, and therefore, the decision was made that in cases such as these, the words are considered under the institutionalised category.

5) **Semantic (altered)**

The meanings of these words differ when used in diminutive form (examples: "*graatjie*" 'fish-bone', "*hokkie*" 'hockey').

6) **Diminutive as noun-former (unaltered)**

(Examples: "*bruintjie*" 'brown one', "*kleintjie*" 'little one').

7) **Diminutive as adverbial (unaltered)**

Usually, these are words with an emotive charge (examples: "*fyntjies*" 'cleverly', "*effentjies*" 'slightly', "*saggies*" 'softly', "*skoontjies*" 'neatly', "*koffietjies*" 'a little coffee').

ADDENDUM B

Complete list of affixes, grouped under the inflectional categories defined in Section 3.4.

- **Plural**

Affix	Example	Translation	Lemma
-e	*"mense"*	humans	*"mens"*
-i	*"politici"*	politicians	*"politikus"*
-ers	*"kinders"*	children	*"kind"*
-dere	*"beendere"*	bones	*"been"*
-s	*"mans"*	males	*"man"*
-a	*"stadia"*	stadiums	*"stadion"*
-'s	*"pa's"*	fathers	*"pa"*
-'e	*"m'e"*	m's	*"m"*
-ns	*"vermoëns"*	abilities	*"vermoë"*
-ens	*"kwajongens"*	naughty boys	*"kwajong"*

- **Degrees of comparison**

Affix	Example	Translation	Lemma
-er	*"verder"*	further	*"ver"*
-ër	*"hoër"*	higher	*"hoog"*
-ste	*"mooiste"*	most beautiful	*"mooi"*
-stes	*"grootstes"*	geatest ones	*"groot"*

- **Diminutive**

Affix	Example	Translation	Lemma
-etjie	*"balletjie"*	small ball	*"bal"*
-etjies[6]	*"mannetjies"*	small men	*"man"*
-ie	*"huisie"*	small house	*"huis"*
-ies	*"koekies"*	small cakes	*"koek"*
-jie	*"ritjie"*	short journey	*"rit"*
-jies	*"gebedjies"*	short prayers	*"gebed"*
-kie	*"koninkie"*	little king	*"koning"*
-kies	*"garinkies"*	small threads	*"garing"*
-pie	*"boompie"*	small tree	*"boom"*
-pies	*"pypies"*	small pipe	*"pyp"*
-tjie	*"vroutjie"*	little woman	*"vrou"*

[6] The *-s* does not form part of the diminutive affix, it actually indicates the plural form. It is presented in the diminutive category, since the lemmatisation task in this study is seen as a once-off process where the *-s* is removed together with the rest of the affix during lemmatisation. The same principle applies to *-jies*, *-ies*, *-kies*, *-pies*, *-tjies* and *-'tjies*

-tjies	*"miertjies"*	small ants	*"mier"*
-'tjie	*"foto'tjie"*	small photo	*"foto"*
-'tjies	*"tablo'tjies"*	small tableaux	*"tablo"*

- **Past Tense**

Affix	Example	Translation	Lemma
ge-	*"geloop"*	walked	*"loop"*
aange-	*"aangebied"*	offered	*"aanbied"*
alleenge-[7]	*"alleengeloop"*	walked alone	*"alleenloop"*
aanmekaarge-	*"aanmekaargetimmer"*	knocked together	*"aanmekaartimme"*
agterge-	*"agtergebly"*	remained behind	*"agterbly"*
agternage-	*"agternagesit"*	pursued	*"agternasit"*
bangge-	*"banggemaak"*	frightened	*"bangmaak"*
binnege-	*"binnegekom"*	entered	*"binnekom"*
buitege-	*"buitegespeel"*	played outside	*"buitespeel"*
byge-	*"bygegooi"*	added	*"bygooi"*
deurge-	*"deurgebeur"*	forced through	*"deurbeur"*
deurmekaarge-	*"deurmekaargemaak"*	confused	*"deurmekaarmaak"*
droogge-	*"drooggekook"*	boiled dry	*"droogkook"*
inge-	*"ingeblaas"*	blowed into	*"inblaas"*
klaarge-	*"klaargespeel"*	hashed	*"klaarspeel"*
koudge-	*"koudgewals"*	cold rolled	*"koudwals"*
kwaadge-	*"kwaadgepraat"*	slandered	*"kwaadpraat"*
losge-	*"losgemaak"*	loosened	*"losmaak"*
misge-	*"misgeskiet"*	miss (when shooting)	*"misskiet"*
nage-	*"nagedink"*	considered	*"nadink"*
natge-	*"natgegooi"*	watered	*"natgooi"*
omge-	*"omgestamp"*	knocked over	*"omstamp"*
onderge-	*"ondergedruk"*	pushed under	*"onderdruk"*
onge-	*"ongedraaide"*	unturned	*"ongedraai"*
oopge-	*"oopgemaak"*	opened	*"oopmaak"*
oorge-	*"oorgeboek"*	over booked	*"oorboek"*
openbaarge-	*"openbaargemaak"*	revealed	*"openbaarmaak"*
opge-	*"opgebel"*	called up (phone)	*"opbel"*
platge-	*"platgestoot"*	flattened	*"platstoot"*
rondge-	*"rondgery"*	drove about	*"rondry"*
saamge-	*"saamgegooi"*	threw together	*"saamgooi"*
singe-	*"singemaak"*	made sense	*"sinmaak"*
skerpge-	*"skerpgemaak"*	sharpened	*"skerpmaak"*
skrikge-	*"skrikgemaak"*	frightened	*"skrikmaak"*
stilge-	*"stilgesit"*	sat still	*"stilsit"*
stomge-	*"stomgeslaan"*	astonished	*"stomslaan"*

[7] *"alleen"* does not form part of the affix and is therefore not removed from the word during lemmatisation. The same principal applies to the rest of the past tense affixes.

teenge-	"*teengegaan*"	thwarted	"*teengaan*"
tegemoetge-	"*tegemoetgery*"	drove out to meet	"*tegemoetrys*"
tentoonge-	"*tentoongestel*"	displayed	"*tentoonstel*"
terugge-	"*teruggedruk*"	pushed back	"*terugdruk*"
toege-	"*toegemaak*"	covered	"*toemaak*"
tussenge-	"*tussengevoeg*"	inserted	"*tussenvoeg*"
uitge-	"*uitgespoel*"	washed out	"*uitspoel*"
vasge-	"*vasgedraai*"	tightend	"*vasdraai*"
verbyge-	"*verbygesteek*"	overtaked	"*verbysteek*"
voorge-	"*voorgeloop*"	walked in front	"*voorloop*"
voortge-	"*voorgeveg*"	fought in front	"*voorveg*"
vryge-	"*vrygemaak*"	freed	"*vrymaak*"
wegge-	"*weggee*"	gave away	"*weggee*"

- **Past Participle**

Affix	Example	Translation	Lemma
ge-…-d	"*gemengd*"	mixed	"*meng*"
ge-…-t	"*gelapt*"	mended	"*lap*"
ge-…-de[8]	"*gedraaide*"	turned	"*draai*"
ge-…-te	"*gevlegte*"	braided	"*vleg*"

- **Infinitive**

Affix	Example	Translation	Lemma
-e	"*kope*"	buy	"*koop*"
-e	"*drinke*"	drink	"*drink*"

- **Attributive**

Affix	Example	Translation	Lemma
-ë	"*leë*"	empty	"*leeg*"
-e	"*moeilike*"	difficult	"*moeilik*"

- **Partitive Genitive**

Affix	Example	Translation	Lemma
-s	"*moois*"	beautiful	"*mooi*"

[8] The -e does not indicate the past participle form; it is only included in the past participle category as it forms part of the affix to be removed during lemmatisation.

ADDENDUM C

Lemmatisation probabilities of words containing the inflectional affixes used in this study.

Morpheme	Probability	Morpheme	Probability
-a	0,10299	*koudge-*	0,833333
alleenge-	0,920635	*kwaadge-*	1
aange-	1	*losge-*	0,90625
agterge-	1	*misge-*	0,727273
agternage-	1	*nage-*	0,910112
bangge-	0,888889	*natge-*	1
binnege-	0,666667	*-ns*	0,387805
buitege-	0,864865	*omge-*	0,818792
byge-	1	*onderge-*	0,714286
-ci	0,981132	*onge-*	0,492611
-dere	0,83427	*oopge-*	1
deurge-	1	*oorge-*	0,924812
deurmekaarge-	1	*openbaarge-*[9]	0,5
droogge-	0,835625	*opge-*	0,933333
-e	0,760494	*-pie*	0,412766
-'e	1	*-pies*	0,78481
-ë	0,434211	*platge-*	1
-ens	0,142792	*rondge-*	0,9
-er	0,047619	*-s*	0,560192
-ër	0,83427	*-'s*	1
-ers	0,830189	*saamge-*	0,884615
-etjie	0,967742	*singe-*	1
-etjies	0,685603	*skerpge-*	0,5
ge-	0,098257	*skrikge-*	0,5
ge-...-d	0,404018	*stilge-*	0,5
ge-...-t	0,452174	*stomge-*	0,5
ge-...-de	0,941385	*-ste*	0,924081
ge-...-te	0,824	*-stes*	0,957447
-i	0,14483	*teenge-*	1
-ie	0,525977	*tegemoetge-*	0,5
-ies	0,842566	*tentoonge-*	0,5
inge-	0,520833	*terugge-*	0,958333
-jie	0,990566	*-tjie*	0,728643
-jies	0,692053	*-'tjie*	1
-kie	0,932292	*-tjies*	0,982885

[9] "*openbaarge-*" is unfortunately not found in the training data, therefore we gave it a probabililty score of 0,5. The same applies to "*skerpge-*", "*skrikge-*","*stilge-*" and "*stomge-*".

-kies	1	*-'tjies*	1
klaarge-	1	*toege-*	0,878788
tussenge-	0,5	*voorge-*	0,714286
uitge-	0,889764	*voortge-*	1
vasge-	0,956522	*vryge-*	0,583333
verbyge-	0,666667	*wegge-*	0,941176

ADDENDUM D

Frequency of the classes, arranged in descending order.

Class	Example of Input Word	Lemma	Frequency
0	"*geel*" 'yellow'	"*geel*" 'yellow'	36414
Re>	"*diere*" 'animals'	"*dier*" 'animal'	9099
Rs>	"*koeëls*" 'bullets'	"*koeël*" 'bullet'	9025
Lge>	"*geloop*" 'walked'	"*loop*" 'walk'	1606
Rte>	"*slotte*" 'locks'	"*slot*" 'lock'	1560
Mge>	"*ingeklim*" 'climbed into'	"*inklim*" 'climb into'	1336
Rste>	"*droogste*" 'driest'	"*droog*" 'dry'	1001
Rer>	"*realistieser*" 'more realistic'	"*realisties*" 'realistic'	717
Rse>	"*musse*" 'caps'	"*mus*" 'cap'	576
Rre>ar	"*are*" 'veins"	"*aar*" 'vein'	514
Rke>	"*plekke*" 'places'	"*plek*" 'place'	487
Rde>	"verbeterde" 'improved'	"*verbeter*" 'improve'	483
Rwe>f	"*erwe*" 'stands'	"*erf*" 'stand'	451
Rle>al	"*krale*" 'beads'	"*kraal*" 'beads'	408
Rtjie>	"*seuntjie*" 'little boy'	"*seun*" 'boy'	375
Lge>Rde>	"*gevriesde*" 'frozed'	"*vries*" 'froze'	368
Rle>	"*velle*" 'skins'	"*vel*" 'skin'	320
Rde>id	"*eenhede*" 'units'	"*eenheid*" 'unit'	274
Rte>at	"*soldate*" 'soldiers'	"*soldaat*" 'soldier'	263
Rpe>	"*koppe*" 'heads'	"*kop*" 'head'	242
R's>	"*risiko's*" 'risks'	"*risiko*" 'risk'	233
Rtjies>	"*diertjies*" 'small animals'	"*dier*" 'animal'	209
Rne>	"*sinne*" 'sentences'	"*sin*" 'sentence'	207
Rie>	"*vlerkie*" 'small wing'	"*vlerk*" 'wing'	189
Mge>Rde>	"*ingelyfde*" 'incorporated'	"*inlyf*" 'incorporate'	187
Rme>	"*bomme*" 'bombs'	"*bom*" 'bom'	187
Rle>el	"*voordele*" 'advantages'	"*voordeel*" 'advantage'	180
Rjie>	"*eendjie*" 'little duck'	"*eend*" 'duck'	175
Rme>om	"*drome*" 'dreams'	"*droom*" 'dream'	167
Rse>os	"*rose*" 'roses'	"*roos*" 'rose'	162
Lge>Rd>	"*gedefinieerd*" 'defined'	"*definieer*" 'define'	124
Rne>an	"*loopbane*" 'careers'	"*loopbaan*" 'career'	122
Rpe>op	"*knope*" 'buttons'	"*knoop*" 'button'	120
Rjies>	"*landjies*" 'small countries'	"*land*" 'country'	120
Rë>	"*fobieë*" 'phobias'	"*fobie*" 'phobia'	115
Re>g	"*vliegtuie*" 'aeroplanes'	"*vliegtuig*" 'aeroplane'	113
Rte>ot	"*bote*" 'boats'	"*boot*" 'boat'	112
Rre>ur	"*ure*" 'hours'	"*uur*" 'hour'	112
Rne>on	"*hormone*" 'hormones'	"*hormoon*" 'hormone'	110

Rme>am	"*name*" 'names'	"*naam*" 'name'	102
Mge>Rte>	"*ingesakte*" 'collapsed'	"*insak*" 'collapse'	102
Rë>og	"*oë*" 'eyes'	"*oog*" 'eye'	101
Ries>	"*dorpies*" 'small towns'	"*dorp*" 'town'	97
Lge>Rte>	"*gekraakte*" 'cracked'	"*kraak*" 'crack'	94
Rre>or	"*kantore*" 'offices'	"*kantoor*" 'office'	93
Rwe>af	"*slawe*" 'slaves'	"*slaaf*" 'slave'	89
Rsie>	"*vissie*" 'small fish'	"*vis*" 'fish'	89
Rde>ad	"*dade*" 'deeds'	"*daad*" 'deed'	87
Rkie>	"*sakkie*" 'small bag'	"*sak*" 'bag'	80
Rre>er	"*vere*" 'feathers'	"*veer*" 'feather'	78
Rne>en	"*grafstene*" 'tombstones'	"*grafsteen*" 'tombstone'	74
Rder>	"*donkerder*" 'darker'	"*donker*" 'dark'	74
Rle>ol	"*skole*" 'schools'	"*skool*" 'school'	73
Ra>um	"*akwaria*" 'aquariums'	"*akwarium*" 'aquarium'	73
Rens>	"*treinwaens*" 'railway carriages'	"*treinwa*" 'railway carriage'	70
Re>ag	"*vrae*" 'questions'	"*vraag*" 'question'	70
Raie>d	"*paaie*" 'roads'	"*pad*" 'road'	68
Rke>ek	"*biblioteke*" 'libraries'	"*biblioteek*" 'library'	66
Rpie>	"*besempie*" 'small broom'	"*besem*" 'broom'	65
Rke>ak	"*moordsake*" 'murder cases'	"*moordsaak*" 'murder case'	63
Lgeï>i	"*geïnkarneer*" 'incarnated'	"*inkarneer*" 'incarnate'	61
Lgeë>e	"*geëvalueer*" 'evaluated'	"*evalueer*" 'evaluate'	59
Rë>g	"*oorloë*" 'wars'	"*oorlog*" 'war'	56
Rkie>g	"*verhoginkie*" 'small increase'	"*verhoging*" 'increase'	52
Rwwe>f	"*gifstowwe*" 'toxins'	"*gifstof*" 'toxin'	51
Rkies>	"*takkies*" 'small twigs'	"*tak*" 'twig'	48
Rci>kus	"*politici*" 'politicians'	"*politikus*" 'politician'	48
Lgeï>iRde>	"*geïnspekteerde*" 'inspected'	"*inspekteer*" 'inspect'	45
Rse>as	"*vase*" 'vases'	"*vaas*" 'vase'	44
Rter>	"*natter*" 'wetter'	"*nat*" 'wet'	42
Rd>	"*gewerweld*" 'vertebrated'	"*gewerwel*" 'vertbrate'	41
Rpies>	"*probleempies*" 'small problems'	"*probleem*" 'problem'	38
Rre>	"*karre*" 'cars'	"*kar*" 'car'	34
Rletjie>	"*watervalletjie*" 'small waterfall'	"*waterval*" 'waterfall'	34
Rte>et	"*komete*" 'comets'	"*komeet*" 'comet'	34
Rletjies>	"*brilletjies*" 'small glasses'	"*bril*" 'glasses'	32
Re>d	"*maaltye*" 'meals'	"*maaltyd*" 'meal'	32
Rme>em	"*probleme*" 'problems'	"*probleem*" 'problem'	31
Mge>Re>	"*aangemelde*" 'reported'	"*aanmeld*" 'report'	28
Rede>id	"*klublede*" 'club members'	"*klublid*" 'club member'	28
Repe>ip	"*skepe*" 'ships'	"*skip*" 'ship'	28
Rgie>	"*seiljaggie*" 'small yacht'	"*seiljag*" 'yacht'	28
Rmetjie>	"*blommetjie*" 'little flower'	"*blom*" 'flower'	27
Rsies>	"*fesies*" 'small festivals'	"*fees*" 'festival'	27
Retjie>	"*leerlingetjie*" 'small pupil'	"*leerling*" 'pupil'	26

Rnetjie>	"*kannetjie*" 'small can'	"*kan*" 'can'	25
Rns>	"*hawens*" 'harbours'	"*hawe*" 'harbour'	24
R'tjie>	"*balju'tjies*" 'small sherrifs'	"*balju*" 'sherrif'	23
Rpe>ep	"*staatsgrepe*" 'coups'	"*staatsgreep*" 'coup'	23
Rde>ed	"*wrede*" 'cruel'	"*wreed*" 'cruel'	23
Rwe>ef	"*krewe*" 'lobsters'	"*kreef*" 'lobster'	23
Rpe>ap	"*skape*" 'sheep'	"*skaap*" 'sheep'	23
Rde>od	"*pitabrode*" 'pita breads'	"*pitabrood*" 'pita bread'	23
Rmer>am	"*behulpsamer*" 'more helpful'	"*behulpsaam*" 'helpful'	23
Lgeë>eRde>	"*geëlimineerde*" 'eliminated'	"*elimineer*" 'eliminate'	22
Rë>e	"*houtsneë*" 'wood engravings'	"*houtsnee*" 'wood engraving'	22
Rgies>	"*gesiggies*" 'small faces'	"*gesig*" 'face'	20
Rers>	"*kinders*" 'children'	"*kind*" 'child'	19
Rkies>g	"*poedinkies*" 'small dessert'	"*poeding*" 'dessert'	19
R's>	"*skoonma's*" 'mothers in law'	"*skoonma*" 'mother in law'	19
Ri>us	"*stimuli*" 'stimuli'	"*stimulus*" 'stimulus'	18
Rûe>ug	"*brûe*" 'bridges'	"*brug*" 'bridge'	18
Rner>	"*besetener*" 'more possessed'	"*besete*" 'posessed'	17
Rwe>of	"*stowe*" 'stoves'	"*stoof*" 'stove'	17
Rële>eel	"*sekretariële*" 'secretarial'	"*sekretarieel*" 'secretarial'	17
Lge>Re>	"*gewonde*" 'wounded'	"*wond*" 'wound'	17
Rwer>f	"*sensitiewer*" 'more sensitive	"*sensitief*" 'sensitive'	16
Rler>al	"*radikaler*" 'more radical'	"*radikaal*" 'radical'	15
Rdens>	"*baddens*" 'bathtubs'	"*bad*" 'bathtub'	15
R'e>	"*crèche'e*" 'crèche's'	"*crèche*" 'crèche'	13
Rere>	"*liedere*" 'songs'	"*lied*" 'song'	12
Rmetjies>	"*vlammetjies*" 'small flames'	"*vlam*" 'flame'	12
Rnetjies>	"*nonnetjies*" 'small nuns'	"*non*" 'nun'	11
Rfie>	"*skoffie*" 'short shift'	"*skof*" 'shift'	11
Mge>Rd>	"*aangeleerd*" 'acquired'	"*aanleer*" 'acquire'	11
Rlui>man	"*ambagslui*" 'craftsmen'	"*ambagsman*" 'craftsman'	11
Rler>	"*stiller*" 'more silent'	"*stil*" 'silent'	11
Retjies>	"*wieletjies*" 'small wheels'	"*wiel*" 'wheel'	11
Rde>p	"*hemde*" 'shirts'	"*hemp*" 'shirt'	10
Mgeë>e	"*oorgeënt*" 'revaccinated'	"*oorent*" 'revaccinate'	10
Rnste>	"*volwassenste*" 'most matured'	"*volwasse*" 'mature'	10
R'tjies>	"*kommando'tjies*" 'small commando'	"*kommando*" 'commando'	10
Rsie>as	"*plasie*" 'small farm'	"*plaas*" 'farm'	9
Rede>ad	"*voorstede*" 'suburbs'	"*voorstad*" 'suburb'	9
Rte>ut	"*institute*" 'institutes'	"*instituut*" 'institute'	9
Rentjie>	"*waentjie*" 'small wagon'	"*wa*" 'wagon'	9
Radjies>d	"*paadjies*" 'small path'	"*pad*" 'path'	9
Rge>	"*sogge*" 'sows'	"*sog*" 'sow'	8
Rbe>	"*saadlobbe*" 'seedlobes'	"*saadlob*" 'seedlobe'	8
Rë>eg	"*parapleë*" 'paraplegics'	"*parapleeg*" 'paraplegic'	8
Rler>el	"*geler*" 'more yellow'	"*geel*" 'yellow'	8

Ratjie>t	"*gaatjie*" 'small hole'	"*gat*" 'hole'	8
Rke>ok	"*glimstroke*" 'reflectors'	"*glimstrook*" 'reflector'	8
Res>	"*bloeses*" 'blouses'	"*bloes*" 'blouse'	8
Rta>	"*stomata*" 'stomata'	"*stoma*" 'stoma'	7
Rser>	"*knusser*" 'comfier'	"*knus*" 'comfy'	7
Rfie>of	"*stofie*" 'small stove'	"*stoof*" 'stove'	7
Rse>es	"*vrese*" 'fears'	"*vrees*" 'fear'	7
Rser>os	"*roekeloser*" 'more reckless'	"*roekeloos*" 'reckless'	7
Radjie>d	"*bergpaadjie*" 'small mountain road'	"*bergpad*" 'mountain road'	6
Rsie>os	"*appelkosie*" 'small apricot'	"*appelkoos*" 'apricot'	6
Ratjies>t	"*vaatjies*" 'small barrels'	"*vat*" 'barrel'	6
Rpie>op	"*knopie*" 'small button'	"*knoop*" 'button'	5
Rretjie>	"*sparretjie*" 'small rafter'	"*spar*" 'rafter'	5
Rdste>	"*befoeterdste*" 'most troublesome'	"*befoeter*" 'troublesome'	5
Rfies>of	"*kolestofies*" 'small colestoves'	"*kolestoof*" 'colestove'	4
Lgeï>iRd>	"*geïsoleerd*" 'isolated'	"*isoleer*" 'isolate'	4
Rsies>os	"*rosies*" 'small roses'	"*roos*" 'rose'	4
Rkie>ak	"*aptekie*" 'small pharmacy'	"*apteek*" 'pharmacy'	4
Rgies>ag	"*toelagies*" 'small grant'	"*toelaag*" 'grant'	4
Rpie>ap	"*skapie*" 'little sheep'	"*skaap*" 'sheep'	4
Ra>on	"*hiperbata*" 'hyperbata'	"*hiperbaton*" 'hyperbaton'	4
Rgie>ag	"*plagie*" 'small plague'	"*plaag*" 'plague'	4
Rtes>	"*uitkomstes*" 'outcomes'	"*uitkoms*" 'outcome'	4
Rtia>s	"*deponentia*" 'deponents'	"*deponens*" 'deponent'	4
Rgie>og	"*bogie*" 'small arch'	"*boog*" 'arch'	4
Rwe>ut	"*splinternuwe*" 'brand-new'	"*splinternuut*" 'brand-new'	4
Rretjies>	"*sterretjies*" 'little stars'	"*ster*" 'star'	4
Rwer>	"*ruwer*" 'rougher'	"*ru*" 'rough'	3
Lge>Rdes>	"*gedaagdes*" 'defendents'	"*daag*" 'summon'	3
Rgoed>ding	"*speelgoed*" 'toys'	"*speelding*" 'toy'	3
Rentjies>	"*waentjies*" 'small wagons'	"*wa*" 'wagon'	3
Rle>ul	"*ridikule*" 'ridicule'	"*ridikuul*" 'ridicule'	3
Ruste>	"*skuuste*" 'shiest'	"*sku*" 'shy'	3
Rkie>ek	"*aptekie*" 'small pharmacy'	"*apteek*" 'pharmacy'	3
Rbe>ob	"*halofobe*" 'halophobes'	"*halofoob*" 'halophobe'	3
Ries>y	"*whiskies*" 'whiskies'	"*whisky*" 'whisky'	3
Rpies>ep	"*aandagstrepies*" 'small dashes'	"*aandagstreep*" 'dash'	3
Rntjies>	"*wesentjies*" 'small creatures'	"*wese*" 'creature'	3
Rë>ön	"*sporosoë*" 'sporozoans'	"*sporosoön*" 'sporozoan'	3
Rpies>op	"*winskopies*" 'bargains'	"*winskoop*" 'bargain'	3
Rertjies>	"*kindertjies*" 'small children'	"*kind*" 'child'	3
Rbetjie>	"*lamsribbetjie*" 'small lamb rib'	"*lamsrib*" 'lamb rib'	3
Rwwer>f	"*rowwer*" 'rougher'	"*rof*" 'rough'	3
Res>ag	"*bylaes*" 'addendums'	"*bylaag*" 'addendum'	3
Rmers>	"*lammers*" 'little lambs'	"*lam*" 'lamb'	3
Rker>	"*brakker*" 'more brackish'	"*brak*" 'brackish'	3

Rpies>ap	"*apies*" 'monkeys'	"*aap*" 'monkey'	3
Rner>an	"*spontaner*" 'more spontaneous'	"*spontaan*" 'spontaneous'	2
Rner>on	"*ongewoner*" 'more unusual'	"*ongewoon*" 'unusual'	2
Rntjie>	"*wesentjie*" 'small creature'	"*wese*" 'creature'	2
Rële>eël	"*parsiële*" 'partial'	"*parsieël*" 'partial'	2
Rfies>	"*skoffies*" 'short shifts'	"*skof*" 'shift'	2
Rôe>og	"*sôe*" 'sows'	"*sog*" 'sow'	2
Rwers>f	"*kalwers*" 'calves'	"*kalf*" 'calf'	2
Rdes>	"*verstoordes*" 'disturbed people'	"*verstoor*" 'disturb'	2
Rfies>af	"*skafies*" 'small planes'	"*skaaf*" 'plane'	2
Rfe>	"*karaffe*" 'jugs'	"*karaf*" 'jug'	2
Rpie>ep	"*sepie*" 'small soap'	"*seep*" 'soap'	2
Lge>eRe>	"*geëelte*" 'covered with bunions'	"*eelt*" 'bunion'	2
Lgeë>eRd>	"*geëmaljeerd*" 'enamelled'	"*emaljeer*" 'enamel'	2
Rder>ed	"*wreder*" 'more cruel'	"*wreed*" 'cruel'	2
Rliede>man	"*edelliede*" 'noblemen'	"*edelman*" 'nobleman'	2
Rler>ul	"*ridikuler*" 'more ridicule'	"*ridikuul*" 'ridicule'	2
Lgeü>u	"*geüsurpeer*" 'usurped'	"*usurpeer*" 'usurp'	2
Rdes>ed	"*alkaloïdes*" 'alkaloids'	"*alkaloïed*" 'alkaloid'	2
Rper>	"*papper*" 'more flaccid'	"*pap*" 'flaccid'	2
Rkies>ek	"*strekies*" 'small regions'	"*streek*" 'region'	2
Rfie>af	"*stafie*" 'small bar'	"*staaf*" 'bar'	2
Reë>a	"*regalieë*" 'regalia'	"*regalia*" 'regalia'	2
Rter>ot	"*groter*" 'bigger'	"*groot*" 'big'	2
Rie>ig	"*wie*" 'wedges'	"*wig*" 'wedge'	2
R'etjie>	"*m'etjie*" ' 'small letter m'	"*m*"	2
Rmer>	"*slimmer*" 'cleverer'	"*slim*" 'clever'	2
Rter>at	"*akkurater*" 'more accurate'	"*akkuraat*" 'accurate'	2
Raie>t	"*meerkaaie*" 'mongooses'	"*meerkat*" 'mongoose'	2
Rrie>	"*tjorrie*" 'small ramshackle car'	"*tjor*" 'ramshackle car'	2
Rr>	"*opgewonder*" 'more exited'	"*opgewonde*" 'excited'	1
Rmie>om	"*omie*" 'uncle'	"*oom*" 'uncle'	1
Mge>Rste>	"*uitgestrekste*" 'most extended'	"*uitstrek*" 'extend'	1
Lge>Rdste>	"*gedistingeerdste*" 'most distinguished'	"*distingeer*" 'distinguish'	1
Rer>ag	"*vaer*" 'more vague'	"*vaag*" 'vague'	1
Lgeë>eRe>	"*geënte*" 'engrafted'	"*ent*" 'graft'	1
Roie>d	"*gebooie*" 'commandments'	"*gebod*" 'commandment'	1
Reë>og	"*boeë*" 'arches'	"*boog*" 'arch'	1
Rëler>eël	"*essensiëler*" 'more essential'	"*essensieel*" 'essential'	1
Re-ter-see>-ter-see	"*luitenante-ter-see*" 'naval lieutenants'	"*luitenant-ter-see*" 'naval lieutenant'	1
Rtie>	"*skattie*" 'darling'	"*skat*" 'darling'	1
Rwer>af	"*brawer*" 'more brave'	"*braaf*" 'brave'	1
Rles>	"*hoopvolles*" 'hope	"*hoopvol*" 'hopeful'	1
Rter>ad	"*kwater*" 'more annoyed'	"*kwaad*" 'annoy'	1

Mgeë>eRte>	"*opgeëiste*" 'claimed'	"*opeis*" 'claim'	1
Rëler>eel	"*offisiëler*" 'more official'	"*offisieel*" 'official'	1
Rina>en	"*abdomina*" 'abdomina'	"*abdomen*" 'abdomen'	1
Rgies>eg	"*stegies*" 'small alleys'	"*steeg*" 'alley'	1
Rsie>es	"*fesie*" 'small festival'	"*fees*" 'festival'	1
Rwer>ut	"*nuwer*" 'newer'	"*nuut*" 'new'	1
Lgeü>uRde>	"*geüniformde*" 'clad in uniform'	"*uniform*" 'uniform'	1
Rsies>es	"*fesies*" 'small festivals'	"*fees*" 'festival'	1
Rër>ed	"*breër*" 'wider'	"*breed*" 'wide'	1
R'etjies>	"*n'etjies*" 'small n's'	"*n*"	1
Mge>Rdes>	"*aangesteldes*" 'appointed persons'	"*aanstel*" 'appointed'	1
Rnetjie>en	"*enetjie*" 'little one'	"*een*" 'someone'	1
Rër>	"*toeër*" 'more closed'	"*toe*" 'closed'	1
Rsies>as	"*vasies*" 'small vases'	"*vaas*" 'vase'	1
R'ies>	"*x'ies*" 'small x'es'	"*x*"	1
Mge>Rne>	"*ingespanne*" 'harnessed'	"*inspan*" 'harness'	1
Rfie>ef	"*nefie*" 'cousin'	"*neef*" 'cousin'	1
Rmertjie>	"*lammertjie*" 'little lamb'	"*lam*" 'lamb'	1
Rdere>	"*beendere*" 'bones'	"*been*" 'bone'	1
Rbes>	"*ribbes*" 'ribs'	"*rib*" 'rib'	1
Lgeë>eRte>	"*geëiste*" 'demanded'	"*eis*" 'demand'	1
Rper>op	"*goedkoper*" 'cheaper'	"*goedkoop*" 'cheap'	1
Lgeü>uRd>	"*geüniformd*" 'clad in uniform'	"*uniform*" 'uniform'	1
Rëte>eet	"*diëte*" 'diets'	"*dieet*" 'diet'	1
Mge>Rdste>	"*uitgebreidste*" 'most expanded'	"*uitbrei*" 'expand'	1
Repie>ip	"*skepie*" 'small ship'	"*skip*" 'ship'	1
Rora>us	"*korpora*" 'corpora'	"*korpus*" 'corpus'	1
Rwer>ef	"*skewer*" 'more crooked'	"*skeef*" 'crooked'	1
Rries>	"*tjorries*" 'small ramshackle cars'	"*tjor*" 'ramshackle car'	1
Ri>o	"*concerti*" 'concerti'	"*concerto*" 'concerto'	1
Rde>ud	"*ongehude*" 'unmarried'	"*ongehuud*" 'unmarried'	1
Lgeï>iRdste>	"*geïnteresseerdste*" 'be most interested in'	"*interesseer*" 'be interested in'	1
Rde>e	"*onaangeklede*" 'naked'	"*onaangeklee*" 'naked'	1
Mge>Rle>	"*neergevalle*" 'falled down'	"*neerval*" 'fall down'	1
Rbetjies>	"*varkribbetjies*" 'pork cutlets'	"*varkrib*" 'pork cutlet'	1
Retjies>a	"*babetjies*" 'little babies'	"*baba*" 'baby'	1
Rse>us	"*ekskuse*" 'excuses'	"*ekskuus*" 'excuse'	1
Rëer>og	"*droëer*" 'drier'	"*droog*" 'dry'	1
Lge>eRd>	"*geëerd*" 'honoured'	"*eer*" 'honour'	1
Rfies>ef	"*tefies*" 'small bitches'	"*teef*" 'bitch'	1
Rgies>og	"*ogies*" 'small eyes'	"*oog*" 'eye'	1
Rkies>ok	"*rokies*" 'wisps of smoke'	"*rook*" 'smoke'	1
Rker>ek	"*weker*" 'weaker'	"*week*" 'weak'	1
Rmer>om	"*vromer*" 'more pious'	"*vroom*" 'pious'	1
Rër>g	"*moeër*" 'more tired'	"*moeg*" 'tired'	1

Mgeë>eRde>	"*aangeërfde*" 'obtained by inheritance'	"*aanerf*" 'obtain by inheritance'	1
Rer>d	"*wyer*" 'wider'	"*wyd*" 'wide'	1

BIBLIOGRAPHY

[1] W. Lezius, R. Rapp and M. Wettler, "A Freely Available Morphological Analyzer, Disambiguator and Context Sensitive Lemmatizer for German," in *Proceedings of the COLING-ACL*, 1998, pp. 743-747.

[2] G. Minnen, J. Carrol and D. Pearce, "Applied Morphological Processing of English," in *Natural Language Engineering*, 2001, vol. 7, no.3, pp. 207-223.

[3] A. van den Bosch and W. Daelemans, "Memory-Based Morphological Analysis," in *Proceedings of the 37th Annual Meeting of the ACL*, 1999, pp. 285-292.

[4] R. Sproat, *Morphology and Computation*. London: The MIT Press, 1992.

[5] J. Plisson, N. Lavrac and D. Mladenic, "A rule-based approach to word lemmatization," in *Proceedings C of the 7th International Multi-Conference Information Society IS 2004*, 2004, pp. 83-86.

[6] T. Erjavec and S.Džeroski, "Machine Learning of Morphosyntactic Structure: Lemmatising Unknown Slovene Words," in *Applied artificial intelligence*, 2004, vol.18 no.1, pp. 17-40.

[7] M. Porter, "An Algorithm for Suffix Stripping," in *Program*, 1980, vol. 14 no. 3, pp.130-137.

[8] D. Mladenić, "Automatic Word Lemmatisation", Ljubljana :Jozef Stefan Institute, 2000.

[9] T. Gaustad and G. Bouma, "Accurate Stemming of Dutch for Text Classification", in *Language and Computers*, 2002, vol. 45, no. 1, pp. 104-117.

[10] B. Jongejan and D. Haltrup. (2005, August 23). The CST Lemmatiser. [Online]. Available: http://cst.dk/online/lemmatiser/cstlemma.pdf.

[11] RAGEL ("Reëlgebaseerde Afrikaanse Grondwoord- En Lemma-identifiseerder" 'Rule-based Afrikaans Stemmer and Lemmatiser'). [Online]. Available: http://www.puk.ac.za/opencms/export/PUK/html/fakulteite/lettere/ctext/ragel.html.

[12] MBLEM: Memory-Based Lemmatisation Demo, [Online]. Available: http://ilk.uvt.nl/mblem/.

[13] G. Chrupala, "Simple Data-Driven Context-Sensitive Lemmatization", in *Proceedings of SEPLN 2006*, 2006.

[14] J. Gustafson, N. Lindberg and M. Lundeberg, "The August Spoken Dialogue System," in *Proceedings of Eurospeech*, 1999.

[15] W. Daelemans and H. Strik. (2002). Actieplan Voor Het Nederlands in de Taal- en Spraaktechnologie: Prioriteiten Voor Basisvoorzieningen 'Action Plan for Dutch in Language and Speech Technology: Priorities for Basic Provisions.' [Online]. Available: http://www.cnts.ua.ac.be/Publications/2002/DS02.

[16] W. Daelemans, A. Van den Bosch, J. Zavrel and K. Van der Sloot. "TiMBL: Tilburg Memory Based Learner", Version 5.1, Reference Guide", ILK Technical Report 04-02, 2004.

[17] A.J. Carlson, C.M. Cumby, J.L. Rosen and D. Roth. "SNoW User's Guide", UIUC Tech report UIUC-DCS-R-99-210. [Online]. Available: 2r.cs.uiuc.edu/~danr/Papers/userguide.ps.gz.

[18] T. Joachims, *Learning to Classify Text using Support Vector Machines*, Kluwer Academic Publishers, 2002.

[19] North-West University, *Afrikaanse Speltoetser 3.0, Tesourus 1.0 en Woordafbreker* 'Afrikaans Spelling

Checker 3.0, Thesaurus 1.0 and Hyphenator,' Potchefstroom: CTexT, 2005.

[20] T. Mitchell, *Machine Learning*. London: The MIT Press, 1997.

[21] A. van den Bosch. (2005). Paramsearch 1.0 Beta Patch 24. [Online]. Available: http://ilk.uvt.nl/software.html#paramsearch.

[22] A. Field, *Discovering Statistics using SPSS* (2nd Edition). London: Sage Publishers, 2005.

[23] J. Cohen, *Statistical Power Analysis for the Behavioural Sciences* (2nd Edition). New York: Academic Press, 1988.

[24] R. van Hout and T. Rietveld, *Statistics in Language Research*. New York: Walter De Gruyter Inc., 2005.

[25] *SPSS 14 for Windows*, SPSS Inc., 2005.

[26] D. Jurafsky and J. Martin, *Speech and Language Processing*. New York: Prentice-Hall, 2000.

[27] C.J. van Rijsbergen, *Information Retrieval*. London: Butterworths, 1979.

[28] T. Fawcett, "ROC graphs: Practical Considerations for Researchers," Hewlett-Packard Labs, Tech. Rep. HPL-2003-4, 2004.

[29] S. Kübler, *Memory-Based Parsing*. Amsterdam: John Benjamins Publishing Company, 2004.

[30] W. Daelemans, A. van den Bosch and A. Weijters, "IGTree: Using Trees for Compression and Classification in Lazy Learning Algorithms," in *Artificial Intelligence Review*, 1997, vol. 11, pp. 407-423.

[31] E. Aloaydin. *Introduction to Machine Learning*. Cambridge: MIT Press, 2004.

[32] J.M. Dake. (2003, February 2). Explorations of the Speed-accuracy Trade-off in Memory Based Learning Algorithms. *ILK Technical Report 03-02*. [Online]. Available: http://ilk.kub.nl/downloads/pub/papers/ilk0302.pdf.

[33] P. Dayan, "Unsupervised Learning," in *The MIT Encyclopedia of the Cognitive Sciences*, R. Wilson and F. Keil, Ed. London: The MIT Press, 1999.

[34] P.W. Wagacha, "Instance-Based Learning: *k*-Nearest Neighbour," Nairobi:University of Nairobi, 2003.

[35] D. Aha, D. Kibler and M. Albert, "Instance-Based Learning Algorithms," in *Machine Learning*, 1991, vol. 6, pp. 37-66.

[36] J. Quinlan, *C4.5: Programs for machine learning*. San Mateo, CA: Morgan Kaufmann, 1993.

[37] GB van Huyssteen, M.M. van Zaanen, "A Spelling Checker for Afrikaans based on morphological analysis," in *Proceedings of the 6th International Terminology in Advanced Management Applications Conference (TAMA2003)*, 2003, pp. 189-194.

[38] A.G. Jenkinson, "The Problem of Inflection and Derivation in Afrikaans," in *South African Journal of Linguistics Supplement*, 1993, vol. 18, pp. 100-122.

[39] R.H.Gouws, *Leksikografie* 'Lexicography.' Cape Town: Academica, 1989.

[40] M. de Villiers, *Afrikaanse klankleer: Fonetiek, Fonologie en Woordbou* 'Afrikaans Phonetics: Phonetics, Phonology and Word Construction.' Cape Town: A.A. Balkema, 1969.

[41] L.Bauer, *English Word-formation*. Cambridge: Cambridge University Press, 1983.

[42] D. Tuggy, "The inflectional/derivational distinction," in *Workpapers of the Summer Institute of Linguistics at University of North Dakota*, 1985, vol. 29, pp. 209-222.

[43] J.G.H. Combrink, *Afrikaanse Morfologie* 'Afrikaans Morphology.' Stellenbosch: Academica, 1990.

[44] W.J. de Klerk, *Die Aard van Dialektiese Verskeidenheid in Afrikaans* 'The Nature of Dialectic Diversity in

Afrikaans.' Unpublished D. Litt Thesis, Pretoria: University of Pretoria, 1968.

[45] C.P. van der Walt, C.P. van Aardt and L.C. Eksteen, *Taalkunde vir die Middelbare Skool, standards 9 en 10* 'Language Study for the Secondary School, standards 9 and 10.' Johannesburg: Voortrekkerpers, 1969.

[46] M.J. Posthumus, "Die Woordstruktuur van Afrikaans" 'The Word Structure of Afrikaans,' in *Inleiding tot die Taalkunde* 'Introduction to Linguistics.' H.J.J.M. van der Merwe, Ed. Pretoria: J.L. Van Schaik Bpk., 1964, pp. 135-148.

[47] M.J. Posthumus, "Morfologie" 'Morphology,' in *Inleiding tot die Taalkunde* 'Introduction to Linguistics.' H.J.J.M. van der Merwe, Ed. Pretoria: J.L. Van Schaik Bpk., 1969, pp. 101-123.

[48] A.G. Jenkinson "Aspekte van die Morfologie van Afrikaans" 'Aspects of Afrikaans Morphology,' in *Kongresreferate LVSA 1983*, 1983, pp. 127-149.

[49] P.J.du Toit, "Taalleer vir Onderwyser en Student" 'Language Learning for Teacher and Student.' Pretoria: Academica, 1986.

[50] M.J. Posthumus, "Probleme rondom die morfologie" 'Problems surrounding morphology.' in *Inleiding tot die Taalkunde* 'Introduction to Linguistics.' H.J.J.M. van der Merwe, Ed. Pretoria: J.L. Van Schaik Bpk., 1971, pp. 101-122.

[51] A.G. Jenkinson,"Die diminutief: fleksiemorfeem of afleidingsmorfeem?" 'The diminuitive: inflection or derivation morpheme?,' in *S.A. Tydskrif vir Taalkunde* 'S.A. Linguistics Magazine', 1986, vol.4 no.3, pp. 13-46.

[52] C. van Heerden, *Inleiding to die Semantiek* 'Introduction to semantics.' Johannesburg: Willem Gouws (Edms.) Bpk, 1965.

[53] H.J.J.M. van der Merwe, *Inleiding tot die Taalkunde* 'Introduction to Linguistics.' Pretoria: J.L. van Schaik Bpk., 1964.

[54] J. G. H. Combrink, "Soek: Afrikaans se fleksie" 'Wanted: The inflectional morphemes of Afrikaans,' in *Taalkunde - 'n Lewe* 'Linguistics - a life.' F.F. Odendal, Ed. Cape Town: Tafelberg, 1974, pp. 21-30.

[55] Geert Booij, *The Grammar of Words*. New York: Oxford University Press, 2005.

[56] V.I. Levenshtein, "Binary Codes Capable of Correcting Deletions, Insertions and Reversals," in *Dokl. Akad. Nauk SSR*, 1965, vol 163, no.4, pp. 845-848.

[57] O. Streiter and E.W. de Luca, "Example-based NLP for Minority Languages: Tasks, Resources and Tools," in *Proceedings of TALN 2003*, 2003.

[58] Calomo ("C5 Afrikaanse Lettergreepverdeler vir Outomatiese Morfologiese Ontleding" 'C5 Afrikaans Hyphenator for Automatic Morphological Analysis '). [Online]. Available: http://www.puk.ac.za/opencms/export/PUK/html/fakulteite/lettere/ctext/calomo.html

[59] E. Marsi (2004), "Knngraph version 1.0 beta." [Online]. Available: http://ilk.uvt.nl/~marsi/software/knngraph.html.

[60] W. Daelemans and A. van den Bosch, *Memory-based Language Processing*. Cambridge: Cambridge University Press, 2005.

[61] F. Aurenhammer and R. Klein. (2000). Voronoi Diagrams. [Online]. Available: http://www.pi6.fernuni-hagen.de/publ/tr198.pdf.

[62] A. van den Bosch, *Learning to Pronounce Written Words: A Study in Inductive Language Learning*, Cadier

and Keer: Uitgeverij Phidippides, 1997.

[63] W. Daelemans, A. van den Bosch and J. Zavrel, "A Feature-relevance Heuristic for Indexing and Compressing Large Case Bases," in *Poster Papers of the Ninth European Conference on Machine Learning*. 1997, pp. 29-38.

[64] S.A. Dudani, "The Distance-weighted *k*-Nearest Neighbor rule," in *IEEE Trans. Systems, Man and Cybernetics*. 1976, vol. SMC-6, pp. 325-327.

[65] R.N. Shepard, "Toward a universal law of generalization for psychological science", in *Science,* 1987, vol 237, pp. 11317-1323.

[66] A. van den Bosch, "Wrapped Progressive Sampling Search for Optimizing Learning Algorithm Parameters," in *Proceedings of the 16th Belgian-Dutch Conference on Artificial Intelligence*, R. Verbrugge, N. Taatgen and L. Schomaker, Eds. 2004.

[67] R. Kohavi and G. John, "Wrappers for Feature Subset Selection," in *Artificial Intelligence Journal*, 1997, pp. 273-324.

[68] PSearch. [Online]. Available: http://www.puk.ac.za/opencms/export/PUK/html/fakulteite/lettere/ctext/psearch.html.

[69] C. Fellbaum, *Wordnet - An electronic lexical database*. London: MIT Press, 1998.

[70] F.F. Odendal and R.H. Gouws, *Handwoordeboek van die Afrikaanse Taal* 'Desk Dictionary of Afrikaans.' Midrand: Perskor, 2003.

[71] L.C. Eksteen and F.J. Labuschagne, *Verklarende Afrikaanse Woordeboek* 'Explanatory Afrikaans Dictionary.' Cape Town: Pharos Dictionaries, 1993.