An Inductive-Learning Approach to Morphological Analysis *

Antal van den Bosch
antal@cs.rulimburg.nl

Walter Daelemans
walter@kub.nl

Ton Weijters
weijters@cs.rulimburg.nl

Abstract

Morphological analysis is an important subtask in text-to-speech conversion, hyphenation, and other language engineering tasks. The traditional approach to performing morphological analysis is to combine a morpheme lexicon, sets of (linguistic) rules, and heuristics to find a most probable analysis. In contrast, we present an inductive learning approach in which morphological analysis is reformulated as a segmentation task. We report on a number of experiments in which five inductive learning algorithms are applied to three variations of the task of morphological analysis. Results show (i) that the generalisation performance of the algorithms is good, and (ii) that the *lazy learning* algorithm IB1-IG performs best on all three tasks. We conclude that lazy learning of morphological analysis as a classification task is indeed a viable approach; moreover, it has the strong advantages of avoiding the knowledge-acquisition bottleneck, being fast and deterministic in learning and processing, and being language-independent.

1 Introduction

Morphological analysis is often deemed to be an important, if not essential subtask in linguistic modular systems for text-to-speech processing (Allen, Hunnicutt, and Klatt, 1987) and hyphenation (Daelemans, 1989). In text-to-speech processing, it serves to prevent incorrect application of grapheme–phoneme conversion rules across morpheme boundaries (e.g. preventing *carelessly* from being pronounced as /kə'rɛləslai/). In hyphenation, it guides the placement of hyphens at certain morphological boundaries (e.g. preventing *looking* from being hyphenated as *loo-king*). Morphological analysis also

*This paper is a revised, extended version of Van den Bosch *et al.* This research was performed while the second author was a visiting fellow at NIAS (Netherlands Institute for Advanced Studies, Wassenaar) plays a crucial role in applications such as part-of-speech tagging (assigning the correct morpho-syntactic category to words in context), for obtaining a reasonable analysis of words not present in the lexicon.

The traditional approach to performing morphological analyses presupposes the availability of a morpheme lexicon, spelling rules, morphological rules, and heuristics to prioritise possible analyses of a word according to their plausibility (e.g. see the DECOMP module in the MITtalk system (Allen, Hunnicutt, and Klatt, 1987)). In contrast, the approach described in this paper presupposes a morphologically analysed corpus of words (rather than a corpus of morphemes), and an inductive learning algorithm which is trained to segment spelling words into morphemes in the form of a simple classification task.

In this paper, we will first outline what we mean by rephrasing a linguistic problem as a classification task, and introduce five inductive-learning algorithms capable of learning classification tasks. Then, in Section 2, we give an overview of the traditional approach to morphological analysis and introduce our alternative reformulation. In Section 3 we present and analyse the results of the application of the learning algorithms to the morphological analysis task. We conclude this paper with a summary of the obtained results and a discussion of the differences between the traditional approach to morphological analysis and an inductive-learning approach, in Section 4.

1.1 Reformulating linguistic problems as classification tasks

Most linguistic problems can be seen as context-sensitive mappings from one representation to another (e.g. from text to speech; from a sequence of spelling words to a parse tree; from a parse tree to logical form; from source language to target language, etc.). The typical traditional approach to language-engineering problems is to build a description of the general rules governing these mappings, describe additional subregularities, and list the remaining exceptions to the rules and subregularities. The acquisition of this knowledge is labour-intensive and costly. In contrast to this handcrafting approach, an inductive machine-learning method approaches a linguistic problem in an data-oriented way, i.e. it automatically gathers the knowledge needed for solving the problem by considering instances of the problem. By 'instance' we mean a data structure containing an input and its associated 'solution': its classification. The knowledge implicitly present in the collection of instances is used to classify new instances of the same problem.

Most linguistic tasks can be described as classification tasks, i.e. given a description of an input in terms of a number of feature-values, a classification of the input is performed. Two types of classification tasks can be discerned (Daelemans, 1995):

Identification given a set of possible classifications and an input of feature values, determine the correct classification for this input. For example, given a letter surrounded by a number of neighbours (e.g. *a* in *have*), determine the phonemic transcription of that letter.

Segmentation given a set of possible boundary classes and an input consisting of a focus position in its immediate context, determine whether a boundary is associated with the focus position, and if so, which one. For example, determine if the *b* in *table* marks the boundary of a syllable.

Once a task is reformulated as a classification task, it can be learned by an inductive-learning algorithm. Differences exist in the ways inductive algorithms extract knowledge from the available instances. In *lazy learning* (such as memory-based learning, (Stanfill and Waltz, 1986; Daelemans, 1995)), there is no abstraction of higher-level data structures such as rules or decision trees at learning time; learning consists of simply storing the instances in memory. A new instance of the same problem is solved by retrieving those instances from memory that match the new instance best (according to a similarity metric), and by extrapolating from the solutions of these 'nearest neighbours'. The *memory-based learning approach* therefore does not distinguish between regularities and individual exceptions; rule-like behavior automatically emerges from the interaction between the memory contents and the similarity metric used. In *eager learning* approaches (such as C4.5 (Quinlan, 1993) or connectionist learning), abstract data structures (matrices of connection weights in connectionist networks, decision trees in C4.5) are extracted from the learning material during learning. In contrast with lazy learning, eager learning devotes a significant amount of effort to abstracting from instances, rather than simply storing them in memory.

In previous research we have demonstrated the application of a memory-based (lazy) learning approach to several linguistic problems, e.g. segmentation as in hyphenation and syllabification (Daelemans and Van den Bosch, 1992; Van den Bosch et al., 1995), and identification as in grapheme–phoneme conversion (Weijters, 1991; Van den Bosch and Daelemans, 1993; Daelemans and Van den Bosch, 1994), and stress assignment (Daelemans, Gillis, and Durieux, 1994). In most cases, the memory-based (lazy) approach outdid the more eager inductive algorithms. We believe that in a 'noisy' domain such as natural language, abstracting from the training instances is a bad idea because any one instance (however exceptional from the point of view of the learning algorithm) can potentially be a model for new instances.

In this paper, we demonstrate that the memory-based learning approach is also applicable to morphological parsing, by reformulating it as a senmentation task. We compare the approach to alternative inductive machine-learning algorithms. First, we provide a brief summary of the inductive-learning algorithms used in the experiments reported in this paper.

1.2 Algorithms and methods for inductive learning

Inductive learning in its most straightforward form is exhibited by memory-based lazy learning algorithms such as IB1 (Aha, Kibler, and Albert, 1991) and variations, e.g. IB1-IG (Daelemans and Van den Bosch, 1992; Daelemans, Van den Bosch, and Weijters, 1996), in which all instances are fully stored in memory, and in which classification involves a pass along all stored instances. To optimise memory lookup and minimise memory usage, more eager learning algorithms are available that compress the memory in such a way that most relevant knowledge is retained and stored in a quickly accessible form, and redundant knowledge is removed. Examples of such algorithms are the decision-tree algorithms IGTREE (Daelemans, Van den Bosch, and Weijters, 1996) and C4.5 (Quinlan, 1993). Another popular inductive algorithm is the connectionist Back-propagation (BP) (Rumelhart, Hinton, and Williams, 1986) learning algorithm. We provide a summary of the basic functions of these learning algorithms.

1. IB1 (Aha, Kibler, and Albert, 1991) constructs a data base of instances (the *instance base*) during learning. An instance consists of a fixed-length vector of n feature-value pairs, and an information field containing the classification(s) of that particular feature-value vector. When the feature-value vector is associated to more than one classification (i.e. when its classification is ambiguous), the distributions (occurrences) of the different classifications in the learning material are counted and stored with the instance. After the instance base is built, new instances are classified by IB1 by matching them to all instances in the instance base, and calculating with each match the *distance* between the new instance X and the memory instance Y, $\Delta(X,Y)$, using the function in Equation 1.

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

where $W(fi)$ is the weight of the ith feature (in IB1, this weight is equal for all features), and $\delta(x_i,y_i)$ is the distance between the values of the ith feature in instances X and Y. When the values of the instance features are symbolic, as with our linguistic tasks, a simple distance function for $\delta(x_i,y_i)$ is used (Equation 2).

$$\delta(x_i,y_i) = 0 \; if \; x_i = y_i, \; else \; 1 \tag{2}$$

The classification of the memory instance Y with the smallest $\Delta(X, Y)$ is then taken as the classification of X. New to IB1, as compared to the algorithm proposed by Aha, Kibler, and Albert (1991), is that when the single best matching instance is an ambiguous instance (i.e. the instance carries information on the distribution of more than one classifications, rather than a single classification), IB1 selects the classification with the highest occurrence in the instance's distribution. We have furthermore added to the IB1 algorithm a function that, in case of *more than one* best matching memory instance, merges (i.e. sums) the distributions (occurrences) of the classifications of all of these best-matching memory instances, and produces as output the classification that has the highest occurrence in the merged classification distribution. In cases of occurrence ties, random selections are made.

2. IB1-IG (Daelemans and Van den Bosch, 1992; Daelemans, Van den Bosch, and Weijters, 1996) differs from IB1 in the weighting function $W(f_i)$ (cf. Equation 1). This function computes for each feature, over the full instance base, its *information gain*, a function from information theory that is also used in ID3 (Quinlan, 1986) and C4.5 (Quinlan, 1993) (for more details, see Daelemans and Van den Bosch (1992)). In short, the information gain of a feature expresses its relative relevance compared to the other features in performing the mapping from input to classification. This weighting function gives right to the fact that for some tasks, some features are far more important than other features. When information gain is used as the weighting function in the distance function (Equation 1), instances that match on an important feature are regarded as less distant (more alike) than instances that match on an unimportant feature.

3. IGTREE (Daelemans, Van den Bosch, and Weijters, 1996) compresses an instance base into a decision tree. Instances are stored in the tree as paths of connected nodes and leaves contain classification information. Nodes are connected via arcs denoting feature values. Information gain is used in IGTREE to determine the order in which instance feature values are added as arcs to the tree. The reasoning behind this compression is that when the computation of information gain points to one feature clearly being the most important in classification, search can be restricted to matching a test instance to those memory instances that have the same feature value as the test instance at that feature. Instead of indexing all memory instances only once on this feature, the instance memory can then be optimised further by examining the second most important feature, followed by the third most important feature, etc. A considerable compression is

obtained as similar instances share partial paths. The tree structure is compressed even more by restricting the paths to those input feature values that disambiguate the classification from all other instances in the training material. The idea is that 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. In applications to linguistic tasks, IGTREE is shown to obtain compression factors of 90% or more as compared to IB1/IB1-IG (Van den Bosch and Daelemans, 1993; Daelemans and Van den Bosch, 1994).

IGTREE also stores with each non-terminal node information concerning the most probable or *default* classification given the path thus far, according to the classification bookkeeping information maintained by the tree construction algorithm. This extra information is essential when processing new instances. Processing a new instance involves traversing the tree (i.e. matching all feature-values of the test instance with arcs in the order of the overall feature information gain), and either retrieving a classification when a leaf is reached (i.e. an exact match was found), or retrieving the default classification on the last matching non-terminal node if an exact match fails. For more details on IGTREE, see Daelemans, Van den Bosch, and Weijters (1996).

4. C4.5 (Quinlan, 1993) is a well-known decision-tree algorithm which basically uses the same type of strategy as IGTREE to compress an instance base into a compact tree. To this purpose, standard C4.5 also uses information gain, or *gain ratio* (Quinlan, 1993) to select the most important feature in tree building; however, in contrast to IGTREE, C4.5 recomputes this function for each node in the tree. Another difference with IGTREE is that C4.5 implements a pruning stage, in which parts of the tree are removed as they are estimated to contribute to instance classification below a certain threshold.

5. BP (Rumelhart, Hinton, and Williams, 1986) is an artificial-neural-network learning rule, which operates on multi-layer feed-forward networks (MFNs). In these networks, feature-values of instances are encoded as activation patterns in the input layer, and the network is trained to produce an activation pattern at the output layer representing the desired classification. In contrast to the previously described algorithms, BP does not accumulate its knowledge by literally storing (parts of) instances in memory or by constructing a decision tree on the basis of them. Rather, BP tunes the connections between units in the input layer and the hidden layer, and between units of the hidden layer and the output layer, during a training phase in which all training instances are presented several times to the network. The BP learning algorithm, which is a gradient-descent algorithm, attempts

to set the connections between the layers with increasing subtlety, aiming at minimisation of the error on the training material. After training, the units at the hidden layer encode an intermediary representation that captures some essential information from both the input (the feature-values) and the output (the desired classification). These representations are non-symbolic, and do not lend themselves easily for inspection, in contrast to the previously described symbolic algorithms.

When one plans to apply learning algorithms to classification tasks, it is important to establish a method for interpreting the results from such experiments beforehand. In our experiments, we are primarily interested in the *generalisation accuracy* of trained models, i.e. the ability of these models to use their accumulated knowledge to classify new instances that were not in the training material. A method that gives a good estimate of the generalisation performance of an algorithm on a given instance base, is *10-fold cross-validation* (Weiss and Kulikowski, 1991). Using this method, 10 partitionings into a training set (90%) and a test set (10%) are generated on the basis of an instance base, leading to 10 experiments and 10 results per learning algorithm and instance base. Significance tests such as one-tailed t-tests can be applied to the outcomes of 10-fold cross-validation experiments with several learning algorithms trained on the same data.

2 Morphological analysis

2.1 Traditional approaches

The traditional approach to morphological analysis basically presupposes three components: (i) a morpheme lexicon, (ii) a set of spelling rules and morphological rules to discover possible analyses of morphologically complex words, and (iii) prioritising heuristics to choose the most probable analysis from sets of possible analyses. We briefly illustrate the functioning of this type of analysis by taking DECOMP's processing of the word *scarcity* as an example (Allen, Hunnicutt, and Klatt, 1987):

1. In a morpheme lexicon covering the English language, a first analysis divides *scarcity* into *scar* and *city*.

2. The analysis *scar*|*city* is validated by a finite-state automaton covering the possible sequences of morphemes in English words; furthermore, an analysis-cost heuristic assigns an integer-valued cost to the combination of two noun stems.

3. Using spelling rules of letter deletion in inflection and compounding in English, the system suspects that the analysis *scarce*|*ity* is also

possible, as *ity* may have deleted the *e* of *scarce*. This analysis, which is validated by the morpheme-sequence finite-state automaton, yields a lower cost than *scar|city*, as the analysis-cost heuristic assigns a lower value to a derivational affix than to a second stem.

4. As no further spelling-change rules can be applied to the analysis with the lowest cost, *scarce|ity*, the process ends by producing this analysis.

It is argued in Allen, Hunnicutt, and Klatt (1987) that a morpheme lexicon containing 10,000 morphemes is effective in a text-to-speech system. Neologisms, a problem for purely lexicon-based approaches, seldomly contain new morphemes. The morpheme-sequence finite-state automaton, the spelling rules, and the analysis-cost heuristic are in principle not very complex in terms of processing. They demand, however, a considerable amount of knowledge acquisition and fine-tuning. Another serious problem with these analysis components is that the number of analyses of morphologically complex words may become very much larger (near exponential in the number of morphemes) for longer words.

Morphological analysis on a probabilistic basis, using only a morpheme lexicon, an analyses generator, and a probabilistic function to determine the analysis with the highest probability (Heemskerk, 1993) does not suffer from the disadvantageous knowledge acquisition and fine-tuning phase, but is nevertheless also confronted with an explosion of the number of generated analyses.

2.2 Inductive-learning approach

In contrast to this decomposition into three components, we reformulate the task of morphological analysis as a one-pass segmentation task, in which an input (a sequence of letters with a focus position) is to be classified as marking a morpheme boundary at that focus position. This classification approach demands that the number of input features be fixed, hence we cannot use whole words as input. Instead, we convert a word into fixed-sized instances of which the middle letter is mapped to a class denoting a morpheme boundary decision. To generate fixed-sized instances, we adopt the windowing scheme proposed by Sejnowski and Rosenberg (1987) which generates fixed-sized snapshots of words. Surrounding the focus letter, we choose a fixed context of three letters to the left, and three letters to the right of the middle position. This context may prove to be too small to disambiguate between certain instances; we will investigate the impact of this choice (motivated by our earlier work with inductive learning of morphophonological tasks (Weijters, 1991; Van den Bosch and Daelemans, 1993), in which this specific contextual scope proved to disambiguate between instances nearly exhaustively) in Section 3.

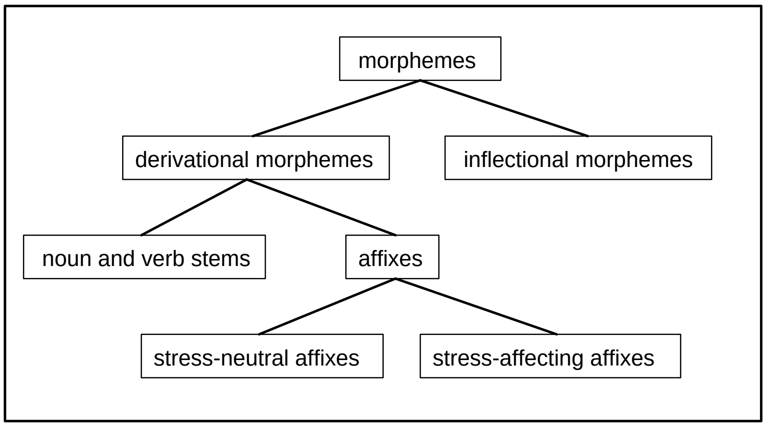

Figure 1: Family tree of English morphemes

In its most basic form, the classification of each instance denotes whether the focus letter of the instance maps to a morpheme boundary ('yes', or '1') or not ('no', or '0'). However, distinguishing between only '1' and '0' does not take into account that morphological theory generally distinguishes between several types of morphemes. For the case of English, a family tree of morphemes would for example be the one displayed in Figure 1.

Distinguishing between, for example, stress-neutral and stress-affecting affixes would be directly helpful as input knowledge for performing the stress-assignment task in a text-to-speech system. However, distinguishing between types of morphemes according to this theory also introduces a certain amount of pre-wired linguistic knowledge. With this in mind we extended the task of morphological analysis into three different tasks, with increasing implicit linguistic knowledge encoded in the classes:

task M1: decide whether the focus letter marks the beginning of

- a morpheme: map to class '1',
- no morpheme: class '0'.

task M2: decide whether the focus letter marks the beginning of

- a derivational morpheme: class 'd',
- an inflectional morpheme: class 'i',
- no morpheme: class '0'.

task M3: decide whether the focus letter marks the beginning of

- a noun or verb stem: class 's',
- a stress-neutral affix: class '1',
- a stress-affecting affix: class '2',

- an inflectional morpheme: class 'i',
- no morpheme: '0'.

Applying the windowing method to the example word *abnormalities* leads to the instances displayed in Table 1, listing for each of the three tasks their appropriate classifications. The morphological analysis of the full word is simply the concatenation of the instance classifications, in which all classifications other than '0' mark morpheme boundaries.

As can be seen from Table 1, a morphological boundary is assigned to the position at which a new morpheme begins, regardless of the spelling changes that may have occurred in the vicinity of that position. For example, the analysis displayed in Table 1 states that the 'surface' form *iti* is a stress-affecting affix, although its 'deep' form is *ity*. A second characteristic of our representation of morphological boundaries, is that it is non-hierarchic. Although morpheme hierarchy may be important in determining the part-of-speech of a word (Allen, Hunnicutt, and Klatt, 1987), it is not necessary to have a full hierarchical analysis when the morphological analysis is used as input to a text-to-speech system.

3 Experiments

3.1 Data collection and algorithmic parameters

The source for the morphological data used in our experiments is CELEX (Burnage, 1990), a large lexical data base of English, Dutch, and German. We extracted from the English data base all available information on wordforms relating to spelling and morphology, and created a lexicon of 65,558 morphologically analysed words (please note that CELEX does not provide a morphological analysis for all of its wordforms: it does not provide obscure and undeterminable analyses of exceptional (loan) words, and, more importantly, it does not analyse words containing etymologically old roots not current in English, such as *ad* in *addict*; these words are included as monomorphemic words in our lexicon). On the basis of the word token frequency information contained in CELEX, we computed that this lexicon covers approximately 65% of the word tokens in the 16,6 million word COBUILD corpus of written text (Burnage, 1990); 26% of the words in our corpus do not occur in the COBUILD corpus. This 65,558-word lexicon was used to create instance bases for the M1, M2, and M3 tasks, each containing 573,544 instances.

For completeness, the learning parameters of the five algorithms described in Section 1, viz. IB1, IB1-IG, IGTREE, C4.5, and BP, as used in our experiments, are the following: (i) IB1 and IB1-IG implement 1-nearest neighbour matching; (ii) C4.5 uses the gain ratio criterion, default pruning, and no subsetting of feature-values; (iii) BP uses a network with 294 input units (letters are locally coded), 50 hidden units, and 2, 3, or 5 output units (classes are locally coded), a learning rate of 0.1, a momentum of 0.4, and an update tolerance of 0.2. IGTREE's functioning is not governed by parameters.

3.2 Results

We applied the five algorithms to the three tasks, performing with each algorithm and each task a 10-fold cross-validation experiment (Weiss and Kulikowski, 1991). We computed for each 10-fold cross-validation experiment the average percentage of incorrectly processed test words. A word is incorrectly processed when *one or more* instance classifications associated with the instances derived from the word are incorrect (i.e. when one or more of the segmentations is incorrect). Figure 2 displays these generalisation errors. The algorithms are ordered on their performance on task M1.

The best performing algorithm on tasks M1, M2, and M3 is IB1-IG. Its performance is significantly better compared to all other algorithms in all three tasks with $p < 0.001$. On task M1, the algorithm performing second best to IB1-IG (12.04% incorrectly processed test words) is IGTREE (14.27%) (level of significance $t(19) = 13.56, p < 0.001$). On task M2, the second best algorithm is IB1 (15.74%) ; IB1-IG processes 14.40% test words incorrectly ($t(19) = 7.64, p < 0.001$). On task M3, IB1-IG incorrectly processes 17.63% of the test words, again followed by IB1 with 18.94% ($t(19) = 6.95, p < 0.001$).

Interesting is the fact that IGTREE performs well on M1, but performs relatively badly on M2 and M3. IGTREE is known to perform worse when the information gain of the input features displays a low variance (Daelemans, Van den Bosch, and Weijters, 1996), i.e. when there is little difference between the relative relevance of the input features. This suggests that the information-gain values of the features with tasks M2 and M3 have less outspoken differences than with M1, which is indeed the case, as is displayed in Figure 3. For all three tasks, Figure 3 displays the fact that the letter immediately preceding the focus letter is the most important one in the segmentation task.

A more general observation on the basis of the results displayed in Figure 2 is that tasks M1, M2, and M3 are increasingly difficult to learn for all algorithms. Distinguishing between more output classes with a finer linguistic granularity obviously increases the difficulty of learning the task. The results in Figure 2 also provide an indication that the performance of the best algorithms is quite good, considering (i) the test words are not seen by the algorithms during training, and (ii) the test words are dictionary words, rather than words from a written text corpus: they are on the average morphologically more complex than words from a free-text corpus.

instance number	left context			focus letter	right context			classification M1	M2	M3
1	_	_	_	*a*	*b*	*n*	*o*	1	d	1
2	_	_	*a*	*b*	*n*	*o*	*r*	0	0	0
3	_	*a*	*b*	*n*	*o*	*r*	*m*	1	d	s
4	*a*	*b*	*n*	*o*	*r*	*m*	*a*	0	0	0
5	*b*	*n*	*o*	*r*	*m*	*a*	*l*	0	0	0
6	*n*	*o*	*r*	*m*	*a*	*l*	*i*	0	0	0
7	*o*	*r*	*m*	*a*	*l*	*i*	*t*	1	d	1
8	*r*	*m*	*a*	*l*	*i*	*t*	*i*	0	0	0
9	*m*	*a*	*l*	*i*	*t*	*i*	*e*	1	d	2
10	*a*	*l*	*i*	*t*	*i*	*e*	*s*	0	0	0
11	*l*	*i*	*t*	*i*	*e*	*s*	_	0	0	0
12	*i*	*t*	*i*	*e*	*s*	_	_	1	i	i
13	*t*	*i*	*e*	*s*	_	_	_	0	0	0

Table 1: Instances with morphological analysis classifications derived from the word *ab|norm|al|iti|es*. The three classification fields belong to tasks M1, M2, and M3, respectively. Denotations of the classification labels is as follows: 0 = no morpheme boundary; 1 = morpheme boundary with M1, and stress-neutral affix with M3; 2 = stress-affecting affix; d = derivational boundary; i = inflectional boundary; s = stem boundary.

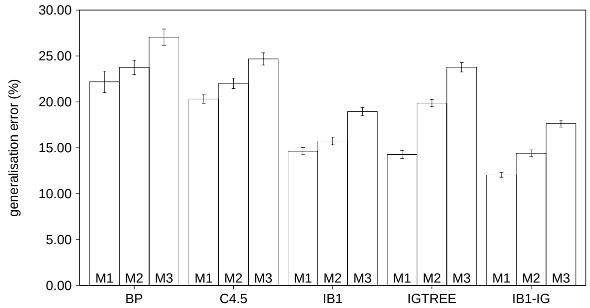

Figure 2: Generalisation errors in terms of the percentage of incorrectly classified test words, with standard deviations (error bars) of five algorithms applied to the three variations of the task of English morphological analysis M1, M2, and M3

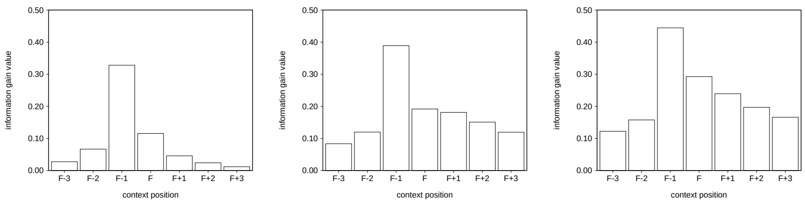

Figure 3: Information-gain values of the features of tasks M1 (left), M2 (middle), and M3 (right), computed over the full instance bases

When the generalisation performance is expressed in terms of incorrectly classified instances, low error rates are obtained. For example, trained on M1, IB1-IG classifies only 1.65% of all test instances incorrectly (1.97% on M2, and 2.46% on M3).

A close inspection of the errors generated by IB1-IG shows that most classification errors occur when IB1-IG is forced to choose between two or more best-matching instances, i.e. when the algorithm decides to choose the most frequently occurring classification summed over the classification distributions of the best-matching instances. Inspecting the application of IB1-IG on the first partitioning of task M1, it was found that only 4.8% of all matches between test instances and stored instances retrieve more than one best-matching instance; in 81% of these ambiguous cases, the classification produced by IB1-IG is correct. However, the remaining 19% incorrect classifications contribute to 60% of the total generalisation error of IB1-IG applied to M1. For task M2, this percentage is 59%; for M3, it is 65%.

A matter related to IB1-IG's problems with similar best-matching instances with different classifications, is the question whether the window size of three left context letters and three right context letters is enough to disambiguate between all instances. A detailed inspection of the decision trees generated by IGTREE applied to the three tasks shows that there are indeed instances that cannot be disambiguated with this context: they are stored in the trees as non-ending (unresolved) nodes at the deepest level of the tree. These nodes carry as classification label the most frequently occurring class for that ambiguous path, which acts as a 'last best' guess when classifying new instances: actually, for IGTREE's application to the first partitioning of M1, 81% of these 'last best' guesses turn out to be correct (a performance which is closely similar to IB1-IG's performance on classifying similar best-matching instances with different classifications). The 19% incorrect best guesses contribute to no less than 52% of the total generalisation error on test instances of the decision tree generated by IGTREE on M1. For task M2 as well as for task M3, this percentage is 68%. These error levels suggest that there might be some gain in the generalisation performance of IGTREE when the context is expanded.

As a test, we applied IGTREE to tasks M1, M2, and M3 (with a 10-fold cross-validation setup), using instances containing *five* left context letters and *five* right context letters, adding up to a window containing eleven letters per instance. The trees generated by IGTREE on these eleven-letter instances contained no non-terminal nodes at the bottom level, indicating that the eleven-letter window is wide enough to disambiguate between all instances. Surprisingly, however, we found that the generalisation performance of IGTREE was *worse* for the three tasks than the performances of IGTREE reported above, on the smaller seven-letter instances: IGTREE's performances on the three tasks were 15.0% incorrectly processed test words for M1 (14.3% previously on the seven-letter task), 20.6% for M2 (19.9% previously), and 25.0% for M3 (23.8% previously). The differences between the performances on the seven-letter and eleven-letter windows are significant for all three tasks, with $p < 0.01$ for M1 and $p < 0.001$ for M2 and M3. It can be concluded from these results that no performance gain can be obtained when the window is expanded to disambiguate between all instances: apparently, the default classification information stored at the bottom of the trees generated on the basis of the seven-letter instances is *more reliable when classifying new instances* than the unambiguous class labels stored at the end nodes of the trees generated on the basis of the eleven-letter instances. This justifies our usage of the seven-letter instances, and points to the interesting fact that morphological analysis can be performed successfully to a large extent using only the small, local contextual scope of seven letters.

As a final illustration, we provide some examples of segmentations generated by IB1-IG on the first partitioning of task M1. Most errors are related to (apparent) morphological ambiguities: incorrect boundary insertions in *ear|ly*, *nav|y*, and *co|al|ed*, and missed boundaries in *printable*, *upland*, and *manslaught|er*. Some examples of correctly segmented words that are morphologically complex are *horse|whip*, *nut|ti|est*, *steep|en*, *veto|es*, and *dis|agree|able|ness*.

Comparing the performance of the algorithms under investigation, applied here to the English CELEX data, with other morphological analysis systems, one is faced with two problems: (i) a proper comparison can only be made when the systems to be compared are tested on the same data, and (ii) we are currently not aware of any system tested on the English CELEX data. Further research should investigate the possibilities of making comparisons with other systems by performing tests on commonly used test sets[1].

4 Conclusions

We have demonstrated the applicability of an inductive machine-learning approach to morphological analysis, by reformulating the problem as a segmentation task in which letter sequences are classified as marking different types of morpheme boundaries. The generalisation performance of inductive-learning algorithms to the task is good.

An interesting result is that within the class of inductive learning algorithms, generalisation accuracy correlates with the degree of eagerness of the inductive algorithm used; best results are obtained with memory-based

[1]Please note that it is possible to train the inductive-learning algorithms on the complete CELEX dataset, rather than on a 90% subset, when the test set is not derived from CELEX, but from free text or any other corpus. The generalisation performance of the algorithms can be expected to be more accurate on this kind of testing material than the performances reported in this paper. Future tests will have to corroborate this hypothesis.

learning (IB1-IG), a lazy learning algorithm retaining full memory of all training instances with a classification-task-related feature-weighting similarity function. The methods abstracting most from the instances perform worst. This corroborates our hypothesis that because of the intricate interaction of regularities, subregularities and exceptions present in this task as well as in most other linguistic problems we studied, lazy learning methods are superior to eager learning methods.

In comparison with the traditional approach, in which morphological analysis is performed by a system containing several components, the inductive learning approach applied to a reformulation of the problem as a classification task of the segmentation type, has a number of advantages:

- it presupposes no more linguistic knowledge than explicitly present in the corpus used for training, i.e. it avoids a knowledge-acquisition bottleneck
- it is language-independent, as it operates on any morphologically analysed corpus in any language
- learning is automatic and fast
- processing is non-recurrent, i.e. it does not retry analysis generation, and is only linearly related to the length or morphological complexity of words.

Nevertheless, it also displays two disadvantages:

- produces an analysis that lacks hierarchy of morphemes
- it does not recover the 'deep' form of morphemes.

Future work on inductive learning of morphological analysis should include a thorough performance comparison with existing traditional systems for morphological analysis, based on linguistic theory and heuristics such as DECOMP (Allen, Hunnicutt, and Klatt, 1987) as well as with probabilistic systems (Heemskerk, 1993). Secondly, we aim at integrating trained learning models of morphological analysis into larger systems, to investigate whether the enrichment of spelling input with morphological boundary information improves the generalisation performance of other learning systems trained on, e.g. stress assignment, grapheme–phoneme conversion, and part-of-speech prediction of unknown words.

References

Aha, D., D. Kibler, and M. Albert. 1991. Instance-based learning algorithms. *Machine Learning*, 7:37–66.

Allen, J., S. Hunnicutt, and D. Klatt. 1987. *From Text to Speech: The* MIT*alk System.* Cambidge University Press, Cambridge, UK.

Burnage, G., 1990. CELEX*: A Guide for Users.* Centre for Lexical Information, Nijmegen.

Daelemans, W. 1989. Automatic hyphenation: Linguistics versus engineering. In F. J. Heyvaert and F. Steurs, editors, *Worlds behind Words.* Leuven University Press, Leuven, pages 347–364.

Daelemans, W. 1995. Memory-based lexical acquisition and processing. In P. Steffens, editor, *Machine Translation and the Lexicon*, number 898 in Springer Lecture Notes in Artificial Intelligence. SPRINGER, pages 85–98.

Daelemans, W., S. Gillis, and G. Durieux. 1994. The acquisition of stress, a data-oriented approach. *Computational Linguistics*, 20:421–451.

Daelemans, W. and A. Van den Bosch. 1992. Generalisation performance of backpropagation learning on a syllabification task. In M. F. J. Drossaers and A. Nijholt, editors, TWLT*3: Connectionism and Natural Language Processing*, pages 27–37, Enschede. Twente University.

Daelemans, W. and A. Van den Bosch. 1994. A language-independent, data-oriented architecture for grapheme-to-phoneme conversion. In *Proceedings of the Second* ESCA/IEEE *Workshop on Speech Synthesis, New York*, pages 199–203. sc esca/ieee.

Daelemans, W., A. Van den Bosch, and A. Weijters. 1996. IGTree: using trees for classification in lazy learning algorithms. *Artificial Intelligence Review.* To appear.

Heemskerk, J. S. 1993. A probabilistic context-free grammar for disambiguation in morphological parsing. In *Proceedings of the 6th Conference of the* EACL, pages 183–192.

Quinlan, J. R. 1986. Induction of decision trees. *Machine Learning*, 1:81–206.

Quinlan, J. R. 1993. C4.5*: Programs for Machine Learning.* MORKAU.

Rumelhart, D. E., G. E. Hinton, and R. J. Williams. 1986. Learning internal representations by error propagation. In D. E. Rumelhart and J. L. McClelland, editors, *Parallel Distributed Processing: Explorations in the Microstructure of Cognition*, volume 1: Foundations. MIT, pages 318–362.

Sejnowski, T. J. and C. S. Rosenberg. 1987. Parallel networks that learn to pronounce English text. *Complex Systems*, 1:145–168.

Stanfill, C. and D. Waltz. 1986. Toward memory-based reasoning. *Communications of the* ACM, 29:1213–1228.

Van den Bosch, A. and W. Daelemans. 1993. Data-oriented methods for grapheme-to-phoneme conversion. In *Proceedings of the 6th Conference of the* EACL, pages 45–53.

Van den Bosch, A., A. Weijters, H. J. Van den Herik, and W. Daelemans. 1995. The profit of learning exceptions. In *Proceedings of the 5th Belgian-Dutch Conference on Machine Learning,* BENELEARN*'95*, pages 118–126.

Weijters, A. 1991. A simple look-up procedure superior to NETtalk? In *Proceedings of the International Conference on Artificial Neural Networks -* ICANN*-91, Espoo, Finland.*

Weiss, S. and C. Kulikowski. 1991. *Computer Systems That Learn.* MORKAU.