Two-step Approach to Unsupervised Morpheme Segmentation

Stefan Bordag
University of Leipzig
sbordag@informatik.uni-leipzig.de

Abstract

This paper describes two steps of a morpheme boundary segmentation algorithm. The task is solely to find boundaries between morphemes bar any further analysis such as phoneme deletions, insertions or alternations that may occur between or within morphemes. The algorithm presented here was designed under the premise that it is not supposed to utilize any knowledge about the language it should analyse. Neither is it supposed to rely on any kind of human supervision. The first step is to use a high-precision, low-recall algorithm to find a relatively small number of mostly correct segmentations, see (Bordag, 2005). In the second step, these segmentations are used to train a classificator, which is then applied to all words to find morpheme boundaries within them.

1 Related Work

The first step of the algorithm presented in this paper is a revised version of the *letter successor variety* (LSV) based algorithm (Harris, 1955; Hafer and Weiss, 1974) (see also (Feng et al., 2004) for an application of that idea to word splitting in Chinese) described and implemented previously by Bordag (2005). This part of the algorithm makes use of contextual information such as cooccurrences of words (the term 'word' will be used synonymously to 'word forms' throughout this paper) within sentences or next to each other. That makes this algorithm comparable to but not identical with another existing algorithm, which also takes a semantic approach (Schone and Jurafsky, 2001).

As it became clear in (Bordag, 2005), the LSV algorithm does not achieve sufficient levels of recall while having a high precision score. Consequently, another algorithm is necessary that can generalize the knowledge produced by the LSV algorithm and apply it to each word in order to increase recall while trying to keep precision high. This second step of the algorithm is based on an implementation of the PATRICIA tree (Morrison, 1968) as a classificator (Witschel and Biemann, 2005), although any other machine learning method could be applied. It is a variant of the trie memory (Fredkin, 1960) that is especially well suited for natural language tasks due to its low memory usage. Each boundary found by the first step is used as a training instance for a PATRICIA tree based classificator. The classificator then, applied to an unanalysed word, marks the most probable prefix or suffix of that word. Given a few simple constraints, the combination of the two algorithms yields a slight precision drop, (probably due to overlearning) versus a strong recall increase compared to the first LSV algorithm alone. Additionally, the classificator is applied recursively to the found affix and the remaining part of the word in order to find morpheme boundaries even within very long words - thus alleviating another problem of the LSV algorithm.

2 The algorithm in two steps

2.1 The First Step: Letter Successor Variety

The letter successor variety has been introduced as *word-splitting* that *occurs if the number of distinct letters* after a given substring rises significantly or above a certain threshold, given a list of words to compare with (Harris, 1955). However, the exact set of words to compare against when measuring the amount of various letters encountered after substrings remains unclear. Especially since there is evidence (Hafer and Weiss, 1974) that the results can be quite noisy, if simply the entire word list is used. In fact, using the **plain global LSV** method (plain stands for not using any weights introduced later on, global stands for comparing the morphology of any word against all other words) with for example the cut-off strategy yields only a maximal F-measure of 41% on the word list used in the experiments below (see Figure 1). Furthermore, it is rather plausible that although for example the word *clearly* has some similarity in letters (measurable by the edit distance) to the word form *early*, there is no doubt that this similarity does not help at all to explain the morphological structure of *clearly* because both words are unrelated otherwise. Thus, a method is needed that can provide a list of words both similar by edit distance and by contexts in which they appear - i.e. either semantically or syntactically related.

One possibility to obtain semantically and/or syntactically related words for a given input word w, is to compute sentence or neighbour cooccurrences (Quasthoff and Wolff, 2002). For example, the word *clearly* cooccurs with *been, said, now, ...* This infor-

mation in the form of a high-dimensional vector space can be used to compute a list of (at most 150 in this work) similar words using any similarity measure such as cosine or 2-norm. Finally, the most similar words for a given input word, ranked both by edit distance and contextual similarity, are used to compute an individual LSV-score for each position within the word. In the English corpus used, the list of most similar words for *clearly* contains *closely, greatly, legally, linearly, really, weakly, ...* The final LSV-score is computed for each position in the input word w by mutliplying the **original LSV-score** with two normalization factors, a **weighted average of parts-frequencies** and the **inverse bigram ranking weight**.

The original LSV-score: The LSV-score for each position between two letters of the input word is computed directly by counting the number of different letters that follow after or before a substring of the word. Table 1 shows that before the substring *-ly* 4 different letters were encountered within the 150 most similar words to *early*. On the other hand, Table 2 shows that for *clearly* 16 different letters were seen before *-ly*. Thus, according to the original LSV idea it is possible to conclude that in the word *early* the syllable *-ly* is not a suffix, whereas in the word *clearly* it is.

The weighted average of substring frequency: However, there are a few quirks with the plain LSV approach. One is that the frequency of the respective substrings within the similarity list plays an important role: For *early* only 6 words out of 19 ending with *-y* ended with *-ly*, compared to 76 out of 90 words for *clearly*. As an improvement over the original LSV method this ratio is used to obtain a confidence weight for how 'trustworthy' the computed LSV at that particular position is. For *ear-ly* it would be $6/19 = 0.3$ compared to $76/90 = 0.8$ for *clear-ly*, further widening the difference between the two types of *-ly*.

Another quirk is that some phonemes are represented by more than one letter such as *th* in English. This results in wrong splittings, because the frequency weight denominator is 'carried away'. But it can be safely assumed that these letter combinations are of a much higher frequency compared to other common combinations of letters. The assumption is safe because single letters have a higher frequency spectrum compared to letter combinations. But if some letter combinations are essentially single phonemes represented by several letters, then they also belong to the higher frequency spectrum of single letters. The corrected frequency weight is computed as a weighted average of frequency weights using a bi- and tri-gram weight computed globally over the entire wordlist. The weights are distributed uniformly along the continumm between 0.0 and 1.0 according to their corresponding frequencies for each n of n-grams individually. Thus, the most frequent bigram receives a weight of 1.0 and the least frequent bigram 0.0.

For the word *thing*, for example, there is a LSV-value of 4 for *th-*. The frequency of *th-* in the 150 most similar words is 12 as opposed to *t-* with 23. The bigram weight of 0.3 allows to down-weight the resulting $12/23 = 0.5$ to $(1.0 * 12/23 + 0.3 * 12/150)/(1.0 + 0.3) = 0.4$. Since for example in German even three letters can represent a single phoneme, the same can be applied to 3-grams and in each case the larger n-gram weight is chosen.

The inverse bigram weight: Taking the inverse bigram weight of the position for which a score is computed can help to weight down such positions that are very improbable to represent a morpheme boundary. In the example of *early*, the bigram *rl* is very rare, thus the weight is 0.0 and the inverse weight is $1.0 - 0.0 = 1.0$. This means that it is quite probable for a boundary to be at that position.

The combination of LSV-score and weights: The final score for each position is computed as the sum of the scores for LSV from left and from right, multiplied with the two weights, the corrected frequency weight and the inverse bigram probability. The resulting score for the example *ear-ly* is the origibal LSV-score for that position, 4, multiplied by the weighted average and the inverse bigram ranking: $4 * (1.0 * 12/23 + 0.3 * 12/150)/(1.0 + 0.3) * (1.0 - 0.0) = 1.2$ as opposed to *clear-ly* with $16 * (1.0 * 76/90 + 0.3 * 76/150)/(1.0 + 0.3) * (1.0 - 0.0) = 13.4$, as can be seen in Tables 1 and 2.

It is possible to label an algorithm as **local** (contrary to global LSV), if it is based on computing the LSV-scores for each word using its similar words instead of simply all words. As can be seen in Figure 1, when using the introduced weights the local variant reaches a maximum peak at approximately the same 64% as the global variant. However, it has a much higher precision at that peak of 71% compared to 59% of the global variant while having a much lower recall value of 56% compared to 70% of the global variant.

For any of the proposed methods using an arbitrarily chosen threshold such as 5, it is possible to decide, whether a given score is a morpheme boundary, or not. This threshold should theoretically depend only on the number of different letters in a given language. It may be a mere coincidence, but as can be seen from Figure 1, the optimum choice of the treshold seems to roughly correspond to the natural logarithm of the number of possible letters except for the case of the plain global LSV-algorithm. However, the lack of cooccurrence observations if the corpus is not large enough, can effectively prevent the discovery of a valid morpheme boundary with an otherwise correctly set threshold.

In fact, the size of the corpus is a rather essential problem for this algorithm. From Zipfs law (Zipf, 1949) follows that in any corpus most words will have a frequency of less then x for some low x such as 10, for example. But if a given word occurs only a dozen of times, then only a few words will be significant neighbour cooccurrences and almost no words can be com-

input word:	#	e	a	r	l	y	#
LSV left:		40	5	1	1	2	1
LSV right:		1	2	1	4	6	19
freq. left:		150	9	2	2	2	1
freq. right:		1	2	2	6	19	150
bigram left:			0.2	0.2	0.5	0.0	
trigram left:				0.0	0.1	0.0	
bigram right:			0.5	0.0	0.1	0.3	
trigram right:			0.0	0.0	0.2		
bigram weight:			0.2	0.5	0.0	0.1	
score left:			0.0	0.0	0.5	1.7	
score right:			1.0	0.0	0.7	0.2	
final score :			1.0	0.1	1.2	2.0	

Table 1: Sample computation of the local LSV algorithm for *early*. Weights were rounded and the given scores and weights refer to the position to the left of the respective letter.

input word:	#	c	l	e	a	r	l	y	#
LSV left:		28	5	3	1	1	1	1	1
LSV right:		1	1	2	1	3	16	10	14
freq. left:		150	11	4	1	1	1	1	1
freq. right:		1	1	2	2	5	76	90	150
bigram left:			0.4	0.1	0.5	0.2	0.5	0.0	
trigram left:				0.1	0.1	0.1	0.1	0.0	
bigram right:			0.5	0.2	0.5	0.0	0.1	0.3	
trigram right:			0.1	0.1	0.0	0.0	0.2		
bigram weight:			0.1	0.5	0.2	0.5	0.0	0.1	
score left:			0.1	0.3	0.0	0.4	1.0	0.9	
score right:			0.3	0.9	0.1	0.0	12.4	3.7	
final score :			0.4	1.2	0.1	0.4	**13.4**	4.6	

Table 2: Sample computation of the local LSV algorithm for *clearly*. Weights were rounded and the given scores and weights refer to the position to the left of the respective letter.

puted as similar to the input word. Thus, in a small corpus even common words might be represented insufficiently for this algorithm. Furthermore, for languages such as Finnish this problem is intensified - due to the large amount of various word forms, each one occurs substantially less frequently in a similar sized corpus and thus it is less probable to obtain a sensible set of semantically similar words for any given input word unless the corpus size is significantly increased.

2.2 The Second Step: A Generalisation using a Trie-Based Classificator

One possibility to circumvent the representativity problems of the local LSV-based algorithm is to use its result in an attempt to generalize them by other means. For this it is feasible to use affix trees such as a trie (Fredkin, 1960) or a PATRICIA compact tree (PCT) (Morrison, 1968). Variations of this data structure have already been widely used for many applications and also for classifications of word strings and their affixes (Cucerzan and Yarowsky, 2003; Sjoeberg and Kann, 2004). The particular implementation used here is the same as in (Witschel and Biemann, 2005).

A PCT can be trained to classify affixes in the following manner: An input consists of the string to be classified, i.e. *clearly*, and the classification class, such as *ly* or 2. This either means that the suffix *-ly* has to be cut, or more simply that the boundary is the second position from the right side of the word. However, the latter variant is more susceptible for overlearning, thus the whole substrings instead of just substring lengths were used as classes. From the examples used in the previous section one valid training instance can be acquired: *clearly ly*. The corresponding reversed uncompressed tree structure would have one node, *y* with one possible decision *ly=1* (with the frequency of 1). This node would have a child node *l* with the same information.

In order to use such a tree for classification, first the deepest possible node in the tree structure has to be retrieved. For the example *daily* it would be the second node *l*, because the next child node is a mismatch between *i* of *daily* and *a* stored in the tree. The probability for any class of the found node is the frequency of that class divided by the sum of all frequencies of all classes of that node. A threshold (in the experiments conducted here it was set to 0.51) can be used to

discern too unclear decisions from clear cases and effectively prevent too much overlearning. In the example *daily*, the probability is 1.0 since there are no other classes stored in the found node. It is noteworthy that such classificator trees have strong generalization abilities while retaining all exceptions. Such a suffix tree, trained on three items *clearly ly*, *strongly ly* and *early NULL* is able to correctly annotate hundreds of words ending with *-ly* while remembering the single exception of *early*. However, it will only be able to produce this single exception, so overlearning is still possible. Pruning, a common technique to cut seemingly redundant branches of the trie for higher efficiency, has not been used here.

For the current special case of affix classification it is important to decide whether the class to be trained is a prefix or a suffix. This is because it does not help much to know that a word begins with *mo* in order to guess whether its trailing *s* is a suffix or not. Therefore, a simple strategy is used to train two distinct classificators: Given an input string with n boundaries, the outermost is selected recursively as a class and cut off for the next training item. If it is more to the right side, then the suffix classificator is trained with that and otherwise the prefix classificator is trained. For example the word *dis-similar-ly* results in the one training item for the suffix classificator *dissimilarly ly*, and one for the prefix classificator *dis similar*.

After training both classificators in the described way, they can be used as a morpheme boundary detection algorithm. For any input word both classificators are used to retrieve their most probable classification. In rare cases this can produce unfitting classifications, such as *-ly* for the input word *May*. This can happen if the lowest common node is *y* and the strongest class at that node is *ly* - such cases are discarded. Then the longer of the two classes (from forward or backward classificator) is taken and a morpheme boundary is introduced according to that classification. Thus, for the example *undertaken* the affix *under-* will be favored over the affix *-en*. A length threshold of 3 is used to determine a valid classification, which means that either the new affix or the remaining word must be longer or equal in length to this threshold in order to avoid degenerated analyses such as *s-t-i-l-l*. After that, this classification algorithm is recursively applied to both parts again. This results in long words such as *hydro-chem-ist-ry* to be analysed completely, where the initial local LSV-based algorithm failed altogether.

3 Quality assessment

A first assessment of the quality of the results can be made by utilizing information available from CELEX (Baayen et al., 1995). However, without any modifications such as introduced to the gold-standard of the MorphoChallenge 2005, analyses such as *lur-ed* will be marked as wrong using this method.

The languages used for this evaluation were German and English. The corpora used were available from the 'Projekt Deutscher Wortschatz' (Quasthoff, 1998). The German corpus contains about 24 million sentences and the English corpus contained 13 million sentences. For the MorphoChallenge 2005 additionally two smaller corpora of 4 million Finnish sentences and 1 million Turkish sentences were used.

Analogically to the evaluation of the MorphoChallenge 2005, in this evaluation the overlap between the manually tagged morpheme boundaries in CELEX and the computed ones is measured. Precision is the number of found correct boundaries divided by the total number of found boundaries. Recall is the number of found correct boundaries divided by the total number of boundaries present in the gold standard, restricted to the words that were in the corresponding corpus.

There are two categories to be measured: the performance of the first part of the algorithm, labeled **LSV** (local LSV in Figure 1) in the tables, and both algorithms combined, labeled **combined** (local LSV+trie in Figure 1). Precision, Recall and the F-measure for the threshold 5 are depicted in Table 3. Additionally Figure 1 shows the performance of the plain global LSV algorithm as well as the global LSV with the introduced weights.

Several observations can be pointed out. The algorithms perform better on the German data than on English. This might be explained by the small size of the English corpus. But another explanation is more plausible: English is morphologically poorer than German. Thus a systematic error either by the algorithm or in the evaluation data would have severe effects on the measured performance. One such systematic mistake is the analysis of the *-ed* affix which in cases such as *lur-ed* is marked as wrong. In a manual error analysis this single error amounts to almost 50% of all reported mistakes for the *LSV* part, followed by other supposedly wrong cases such as *plopp-ed* or *arrang-e-s*. The preliminary results available from the MorphoChallenge 2005 indicate that these considerations were at least partly true since the results reported there had an F-value of 61.7%. The remaining 11% difference to the German results can be more easily explained by the differences in corpus sizes as well as random variation.

	German	English
LSV Precision	80,20	70,35
LSV Recall	34,52	10,86
LSV F-measure	48,27	18,82
combined Precision	68,77	52,87
combined Recall	72,11	52,56
combined F-measure	**70,40**	**55,09**

Table 3: Precision and recall of morpheme boundary detection for both the LSV-based algorithm only and the combination with the PATRICIA tree classifier, based on an unmodified CELEX.

Another interesting point is that for the combined al-

gorithm in both languages there is a medium Precision drop, traded for a large Recall gain: about 38% Recall for German, compared to a loss of 12% in Precision. Thus, the intended effect of increasing Recall without hampering Precision too much by using the tree classifiers has been partly achieved. Nevertheless the resulting precision of 69% for German and merely 53% for English seem to be inhibitively low, albeit the effects of false negatives as reported above are not yet quantified.

It is further interesting that when attempting to let the trie-based classifier learn from the global LSV algorithm the results were almost exactly the same. This indicates that treating each word in its own context and then letting a global algorithm (the trie-based classificator) learn from that indeed improves performance. At the same time, using two global algorithms (global LSV and then the trie) and letting the one learn from the results of the other cannot help because in fact they at best will do the same.

The corpora available to the author for the Finnish and Turkish entries to the MorphoChallenge 2005 are small - an order of magnitude smaller than for German and English. Additionally both languages have almost an order of magnitude more different word forms for the same amount of text when compared to English. Based on these considerations, the LSV threshold was lowered to 2.5 in both cases whereas it was kept at 5 for German and English as reported in (Bordag, 2005) to be a good guess for high precision. Nevertheless, the preliminary results from the MorphoChallenge 2005 indicate that especially for Finnish the corpus size was simply insufficient.

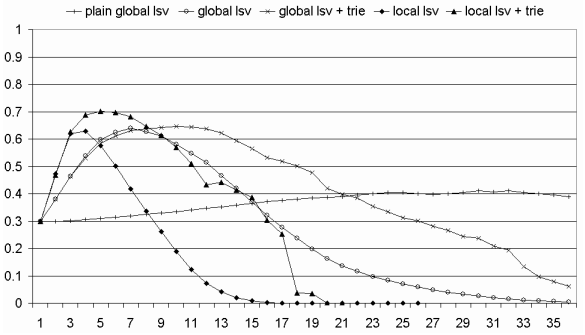

Figure 1: Comparison of global LSV vs. local LSV and after application of the trie-based classificator for a variety of thresholds. Baseline is plain global LSV without normalisations.

3.1 Conclusions

The described experiments show that the combination of one algorithm learning from another is a viable way to increase overall performance, although the results are still far from perfect. Additionally it seems that any single algorithm might work well only for certain (morphological) types of languages and worse for other languages. For example, the local LSV algorithm works quite well for the more flective German but worse for the isolative English and even worse for agglutinative languages such as Finnish or Turkish (at least when it comes to Recall). Other algorithms (Creutz and Lagus, 2005) seem to be inherently better suited for these languages, but might perform worse for e.g. German. One of the main reasons might be the treatment of irregular words: they are usually rather few, but have a high frequency. At the same time their formation is irregular with respect to the majority of other words. That means that comparing them to all other words tends to result in wrong analyses, whereas comparing them to their most similar words should have a greater chance to capture their irregularity because even irregularities tend to be regular in the correct context.

Since most algorithms can provide a confidence score to each decision they make it would be interesting to combine them into a voting system, effectively improving results and broadening the applicability to a wider range of languages.

References

R. Harald Baayen, Richard Piepenbrock, and Léon Gulikers. 1995. *The CELEX lexical database (CD-ROM)*. Linguistic Data Consortium, University of Pennsylvania, Philadelphia, PA, http://www.ldc.upenn.edu/Catalog/CatalogEntry.jsp?catalogId=LDC96L14.

Stefan Bordag. 2005. Unsupervised knowledge-free morpheme boundary detection. In *Proceedings of RANLP 05*, Borovets.

Mathias Creutz and Krista Lagus. 2005. Unsupervised morpheme segmentation and morphology induction from text corpora using morfessor 1.0. In *Publications in Computer and Information Science, Report A81*. Helsinki University of Technology, March.

Silviu Cucerzan and David Yarowsky. 2003. Minimally supervised induction of grammatical gender. In *Proceedings of HLT-NAACL 2003: Main Conference*, pages 40–47.

Haodi Feng, Kang Chen, Chunyu Kit, and Xiaotie Deng. 2004. Unsupervised segmentation of chinese corpus using accessor variety. In *Proceedings of IJCNLP 2004*, pages 694–703, Hainan Island, China, March. Springer.

Edward Fredkin. 1960. Trie memory. *Comm. ACM*, 3(9):490–499, September.

Margaret A. Hafer and Stephen F. Weiss. 1974. Word segmentation by letter successor varieties. *Information Storage and Retrieval*, 10:371–385.

Zellig S. Harris. 1955. From phonemes to morphemes. *Language*, 31(2):190–222.

David R. Morrison. 1968. Patricia - practical algorithm to retrieve information coded in alphanumeric. *Journal of the ACM*, 15(4):514–534, October.

Uwe Quasthoff and Christian Wolff. 2002. The poisson collocation measure and its applications. In *Second International Workshop on Computational Approaches to Collocations*.

Uwe Quasthoff. 1998. Projekt: Der Deutsche Wortschatz. In Gerhard Heyer and Christian Wolff, editors, *Tagungsband zur GLDV-Tagung*, pages 93–99, Leipzig, March. Deutscher Universitätsverlag.

Patrick Schone and Daniel Jurafsky. 2001. Knowledge-free induction of inflectional morphologies. In *Proceedings of NAACL 2001*, Pittsburgh, USA.

Jonas Sjoeberg and Viggo Kann. 2004. Automatic indexing based on bayesian inference networks. In *Proceedings of LREC-2004*, Lisbon, Portugal.

Hans Friedrich Witschel and Christian Biemann. 2005. Rigorous dimensionality reduction through linguistically motivated feature selection for text categorisation. In *Proceedings of NODALIDA-05*, Joensuu, Finland.

George Kingsley Zipf. 1949. *Human Behaviour and the Principle of Least-Effort*. Addison-Wesley, cambridge ma edition.