Unsupervised segmentation of words into morphemes – Challenge 2005
An Introduction and Evaluation Report

Mikko Kurimo, Mathias Creutz, Matti Varjokallio
Adaptive Informatics Research Centre
Helsinki University of Technology
P.O.Box 5400, FIN-02015 HUT, Finland
`Mikko.Kurimo@tkk.fi`

Ebru Arisoy and Murat Saraclar
Bogazici University
Electrical and Electronics Eng. Dept.
34342 Bebek, Istanbul, TURKEY

Abstract

The objective of the challenge for the unsupervised segmentation of words into morphemes, or shorter the *Morpho Challenge*, was to design a statistical machine learning algorithm that segments words into the smallest meaning-bearing units of language, morphemes. Ideally, these are basic vocabulary units suitable for different tasks, such as speech and text understanding, machine translation, information retrieval, and statistical language modeling. The segmentations were evaluated in two complementary ways: *Competition 1:* The proposed morpheme segmentation were compared to a linguistic morpheme segmentation gold standard. *Competition 2:* Speech recognition experiments were performed, where statistical n-gram language models utilized the proposed word segments instead of entire words. Data sets were provided for three languages: Finnish, English, and Turkish. Participants were encouraged to apply their algorithm to all of these test languages.

1 Introduction

Segmentation is a common problem in the analysis of data from many modalities such as gene sequences, image analysis, time series, and segmentation of text into words. It is conceivable that similar machine learning methods could work well in different segmentation tasks.

The task proposed here was to design a statistical machine learning algorithm that segments words into the smallest meaning-bearing units of language, morphemes. The purpose is to obtain a set of basic vocabulary units for different tasks, such as speech and text understanding, machine translation (Lee, 2004), information retrieval (Zieman and Bleich, 1997), and statistical language modeling (Geutner, 1995; Hirsimäki et al., 2006).

In many European languages this task is both difficult and necessary, due to the large number of different word forms found in text. In highly-inflecting languages, such as Finnish and Hungarian, there may be thousands of different word forms of the same root, which makes the construction of a fixed lexicon for any reasonable coverage hardly feasible. Also in compounding languages, such as German, Swedish, Greek and Finnish, complex concepts can be expressed in one single word, which considerably increases the number of possible word forms and calls for the use of sub-word segments as vocabulary units.

The discovery of meaningful word segments has already shown to be relevant for language modeling for speech recognition in Finnish, Turkish and Estonian (Hirsimäki et al., 2006; Kurimo et al., 2006), where language models based on statistically discovered sub-word units have rivaled language models that utilize words. However, any of the research fields dealing with natural language of any kind, as well as multimodal integration, is expected to benefit from the discovery of general meaning-bearing units. For example, a machine translator could have a vocabulary based on minimal meaningful units and generate output words and sentences using them (e.g., translation from English to Finnish: `fact+s about our car+s` / `tieto+a auto+i+sta+mme`). In information retrieval, some of the units (the word roots or stems) might be utilized as key words whereas others might be discarded (e.g., `tietoa` autoistamme $\rightarrow$ tieto auto; Engl. fact car).

A good segmentation algorithm should be able to find units that are meaningful (that is, usable for representing text for many different tasks), that cover as much of the naturally occurring language as possible (including unseen words), and that can be used to generate the totality of the language. The field of linguistics has attempted to capture these properties by the concept of “morpheme”, the difference being that a morpheme may not correspond directly to a particular word segment but to an abstract class. However, in this challenge the task was to uncover concrete word segments.

In obtaining such a segmentation, the use of linguistic analysis and manual coding may be an option for some languages, but not all, due to being very labor-intensive. Furthermore, statistical machine learning methods might eventually discover models that rival even the most carefully linguistically designed morphologies.

In order to be a morphology-discovery method the method should be very language-general, that is, applicable to many different languages without the manual coding of language dependent rules, etc. An example of a general morphology-discovery method is described in (Creutz and Lagus, 2005a).

The main challenge in the task is the sparsity of language data: A significant portion of the words may occur only once even in the largest corpora. Thus, the algorithm should learn meaningful word segments (i.e., inner structures of words) and be capable of generalizing to previously unseen words.

2 Task

The task was the unsupervised segmentation of word forms into sub-word units (segments) given a data set that consists of a long list of words and their frequencies of occurrence in a corpus. The number of unique segments was restricted to the range 1000 - 300,000 (type count). Most of the participants, however, failed to keep the number of segments below 300,000, so it was decided to disregard this limitations and accept all submissions.

Data sets were provided for three languages: Finnish, English, and Turkish. Participants were encouraged to apply their algorithm to all of these test languages. Solutions, in which a large number of parameters must be ”tweaked” separately for each test language were discouraged, since the aim of the challenge was the unsupervised (or very minimally supervised) segmentation of words into morphemes. It was required that the participants submitted clear descriptions of which steps of supervision or parameter optimization were involved in the algorithms.

The segmentations were evaluated in two complementary ways: *Competition 1:* The proposed morpheme segmentation were compared to a linguistic morpheme segmentation gold standard (Creutz and Linden, 2004). *Competition 2:* Speech recognition experiments were performed, where statistical n-gram language models utilized the proposed word segments instead of entire words. Competition 1 included all three test languages. Winners were selected separately for each language. As a performance measure, the F-measure of accuracy of discovered morpheme boundaries was utilized. Should two solutions have produced the same F-measure, the one with higher precision would win. Competition 2 included speech recognition tasks in Finnish and Turkish. The organizers trained a statistical language model based on the segmentations and performed the required speech recognition experiments. As a performance measure, the phoneme error rate in speech recognition was utilized.

3 Data sets

The data sets provided by the organizers consisted of word lists. Each word in the list was preceded by its frequency in the corpora used. The participants’ task was to return exactly the same list(s) of words, with spaces inserted at the locations of proposed morpheme boundaries.

For instance, a subset of the supplied English word list looked like this:

```
6755 sea
1 seabed
1 seabeds
2 seabird
34 seaboard
1 seaboards
```

Submission for this particular set of words might have looked like this:

```
sea
sea bed
sea bed s
sea bird
sea board
```

`sea board s`

The Finnish word list was extracted from newspaper text and books stored at the Language Bank of CSC [1]. Additionally, newswires from the Finnish National News Agency were used.

The English word list was based on publications and novels from the Gutenberg project, a sample of the English Gigaword corpus, as well as the entire Brown corpus.

The Turkish word list was based on prose and publications collected from the web, newspaper text, and sports news.

The desired segmentations, according to the gold standard (Creutz and Linden, 2004), for a small sample of words (500 – 700 words) in each language were provided for download and inspection by the participants. For some words there were multiple correct segmentations, e.g., English: `pitch er s, pitcher s`.

The Finnish gold standard is based on the two-level morphology analyzer FINTWOL from Lingsoft, Inc. The English gold standard is based on the CELEX English data base and the Comprehensive Grammar of the English Language by Quirk et al. (1985) The Turkish linguistic segmentations were obtained from a morphological parser developed at Bogazici University (Cetinoglu, 2000; Dutagaci, 2002). The Turkish parser is based on Oflazer's finite-state machines, with a number of changes.

4 Participants and their submissions

By the deadline of January 15, 2006, 12 research groups had submitted the segmentation results obtained by their algorithms. Totally 14 different algorithms were submitted and 10 of them ran experiments on all three test languages. It is noteworthy that half of the algorithms were designed by groups from the University of Leeds, where participation to this challenge was part of a course in computational linguistics. All the submitted algorithms are listed in Table 1.

In general, the submission were all interesting and relevant. Some of them failed to meet the exact specifications given, but after clarifications were requested, everyone succeeded to provide data that could be properly evaluated. The stipulated maximum count of different segments was exceeded by most of the participants, but after it turned out that this did not impede the evaluation, this restriction was removed.

In addition to the competitors' 14 segmentation algorithms, we evaluated a public baseline method called Morfessor (Creutz and Lagus, 2002; Creutz and Lagus, 2005b) organizers as well as its two more recent versions "Categories-ML" (Creutz and Lagus, 2004) and "Categories-MAP" (Creutz and Lagus, 2005a). Mikko lisaa viitteen .bib-tiedostoonsa. Together with one of the challenge participants, Eric Atwell, the organizers also extended Atwell's original committee classifier algorithm "Cheat" to utilize the segmentations of all the other submissions ("Cheat-all") in addition to only the segmentations from Leeds. Naturally, these later extensions as well as the Morfessor versions competed outside the main competition and the results were included only as reference.

5 Competition 1

5.1 Evaluation

In Competition 1, for each language, the morpheme segmentations proposed by the participants' algorithm were compared against a linguistic gold standard. In the final evaluation, only a subset of all words in the data were included. For each language, a random subset consisting of 10 % of all unique word forms were picked, and the segmentations of these words were compared to the reference segmentations in the gold standard. The exact constitution of this subset was not revealed to the participants. In the evaluation, word frequency played no role. All words were equally important, were they frequent or rare.

The evaluation program, written in Perl, was provided beforehand in order to let the participants evaluate their segmentations relative to the gold standard samples provided in the Challenge. The evaluation was based on the placement of morpheme boundaries.

Example. Suppose that the proposed segmentation of two English words are:

```
boule vard
cup bearer s'
```

The corresponding desired (gold standard) segmentations are:

```
boulevard
cup bear er s '
```

Taken together, the proposed segmentations contain 2 hits (correctly placed boundaries between `cup` and `bear`, as well as between `er` and

[1] `http://www.csc.fi/kielipankki/`.

Table 1: The submitted algorithms.

Name	Authors	Affiliation
A1 “Summaa”	Choudri and Dang	Univ. Leeds, UK
A2a	Bernhard	TIMC-IMAG, F
A2b	Bernhard	TIMC-IMAG, F
A3 “A.A.”	Ahmad and Allendes	Univ. Leeds, UK
A4a “Comb”	Bordag	Univ. Leipzig, D
A4b “Lsv”	Bordag	Univ. Leipzig, D
A5	Rehman and Hussain	Univ. Leeds, UK
A6 “RePortS”	Pitler and Keshava	Univ. Yale, USA
A7 “Bonnier”	Bonnier	Univ. Leeds, UK
A8	Kitching and Malleson	Univ. Leeds, UK
A9 “Pacman”	Manley and Williamson	Univ. Leeds, UK
A10	Johnsen	Univ. Bergen, NO
A11 “Swordfish”	Jordan, Healy and Keselj	Univ. Dalhousie, CA
A12a “Cheat”	Atwell and Roberts	Univ. Leeds, UK
M1 “Baseline”	Morfessor	Helsinki Univ. Tech, FI
M2 “Categories-ML”	Morfessor	Helsinki Univ. Tech, FI
M3 “Categories-MAP”	Morfessor	Helsinki Univ. Tech, FI
A12b “Cheat-all”	Atwell and the organizers	Leeds and Helsinki
A12c “Cheat-top5”	Atwell and the organizers	Leeds and Helsinki

`s`). There is 1 *insertion* (the incorrect boundary between `boule` and `vard`) and 2 *deletions* (the missed boundaries between `bear` and `er`, and between the plural `s` and the apostrophe `'` marking the possessive).

Precision is the number of hits H divided by the sum of the number of hits and insertions I:

$$\text{Precision} = H/(H + I)\ . \tag{1}$$

Recall is the number of hits divided by the sum of the number of hits and deletions D:

$$\text{Recall} = H/(H + D)\ . \tag{2}$$

F-Measure is the harmonic mean of precision and recall, which equals:

$$\text{F-Measure} = 2H/(2H + I + D)\ . \tag{3}$$

According to the rules, the participant achieving the highest F-measure was to be the winner of Competition 1. In case of a tie, higher precision wins. Winners are selected separately for each language.

5.2 Results of Competition 1

The obtained F-measure percentages in the different tasks of Competition 1 are shown in Table 2. The corresponding precision and recall figures are shown in Tables 3 and 4, respectively.

For the Finnish task the winner (measured by F-measure) was the algorithm A2b from TIMC-IMAG in France. Next came A2a also from TIMC-IMAG and A1 from the University of Leeds. The best overall score was obtained by Morfessor M2.

A2b from TIMC-IMAG won also the Turkish task by a clear marginal. Next came A4a from the University of Leipzig and the committee classifier A12a from Leeds. The best overall score was obtained by Morfessor M3.

In the English task, the clear winner was the algorithm A6, i.e., “RePortS” from the University of Yale, who did not participate in any other language. Next came A2a and A2b from TIMC-IMAG, of which A2a scored better in this task. The A6 algorithm succeeded to beat also all Morfessors.

For English, the committee classifiers A12a, A12b, A12c from Leeds dominated all the other participants that were utilized as committee members. In Finnish only A12c and in Turkish A12a and A12c managed to do the same. Thus, the best score was always obtained by A12c, the committee of the top 5 of the other segmentation algo-

Table 2: The obtained F-measure % on different languages (Competition 1).

Name	Finnish	Turkish	English
A1	61.3	55.4	49.8
A2a	63.3	55.3	66.6
A2b	*64.7*	*65.3*	62.4
A3	n.a.	n.a.	32.0
A4a	48.3	57.0	61.7
A4b	3.8	5.2	58.5
A5	43.4	45.2	53.8
A6	n.a.	n.a	*76.8*
A7	40.8	43.5	48.0
A8	n.a.	n.a.	36.2
A9	28.2	40.0	28.5
A10	n.a.	n.a.	43.7
A11	35.2	26.3	45.7
A12a	61.2	55.9	55.7
Winner	A2b: 64.7	A2b: 65.3	A6: 76.8
M1	54.2	51.3	66.0
M2	*67.0*	69.2	69.0
M3	66.4	*70.7*	66.2
A12b	62.0	59.7	77.4
A12c	*68.3*	*71.7*	*78.6*

Table 3: The obtained precision % on different languages (Competition 1).

Name	Finnish	Turkish	English
A1	66.2	58.8	44.7
A2a	73.6	77.9	67.7
A2b	63.0	65.4	55.2
A3	n.a.	n.a.	24.1
A4a	*74.8*	*79.9*	62.6
A4b	52.4	70.3	61.2
A5	66.3	60.4	50.6
A6	n.a.	n.a.	*76.2*
A7	49.3	55.6	47.1
A8	n.a.	n.a.	32.5
A9	25.2	38.1	22.9
A10	n.a.	n.a.	37.5
A11	70.2	59.4	57.1
A12a	67.2	61.0	57.6
Best	A4a: 74.8	A4a: 79.9	A6: 76.2
M1	*84.4*	79.1	63.1
M2	70.1	73.7	64.1
M3	75.0	77.5	*85.1*
A12b	84.1	*86.7*	*86.0*
A12c	76.3	78.4	83.2

rithms.

5.3 Discussion

It is not that surprising that the same algorithm (A2b) wins in both the Finnish and Turkish task of Competition 1, whereas another algorithm (A6) outperforms the others in the English task. Word forming is different in Finnish and Turkish, on the one hand, and in English, on the other hand. Since English words consist of fewer morphemes, English data tends to be less sparse.

Unfortunately, the A6 algorithm, which performs extremely well on English, has not been evaluated "officially" on the two other languages. However, in their paper in these proceedings, the designers of A6 (Keshava and Pitler) report segmentation accuracies for all three languages on the small development sets provided in the challenge. It turns out that their algorithm reaches only average performance on the agglutinative languages Finnish and Turkish. Since the recall is not very high, one might assume that their algorithm suffers from the higher data sparseness of Finnish and Turkish when attempting to "peel off" prefixes and suffixes from word stems.

The committee classifier (A12a, A12b, and A12c) is an interesting approach, which generally obtains very good results. The committee classifier compares the outputs of several other systems and selects for each word the segmentation that the majority of the systems have proposed. If the majority vote results in a tie, the segmentation of the system with the highest F-measure is chosen. Thus, in order for the committee classifier to work, it seems necessary to have access to some reliable gold standard, as the performance of the other systems needs to be assessed. However, the gold standard can be fairly small, as demonstrated by the use of the segmentation samples (development sets) provided in the challenge.

6 Competition 2

6.1 Evaluation

In Competition 2, the organizers utilized the segmentations provided by the participants in order to segment the words in large corpora of Finnish as well as Turkish text. An n-gram language model was trained for this segmentation and this language model used in speech recognition experiments.

The winner of Competition 2 is the participant

Table 4: The obtained recall % on different languages (Competition 1).

Name	Finnish	Turkish	English
A1	57.0	52.3	56.1
A2a	55.6	42.8	65.5
A2b	*66.4*	*65.2*	71.6
A3	n.a.	n.a.	47.6
A4a	1.9	2.7	54.9
A4b	44.8	47.9	62.2
A5	32.2	36.1	57.3
A6	n.a.	n.a.	*77.4*
A7	34.8	35.8	49.0
A8	n.a.	n.a.	40.9
A9	32.0	42.0	37.9
A10	n.a.	n.a.	52.3
A11	23.5	16.8	38.1
A12a	56.1	51.5	53.8
Best	A2b: 66.4	A2b: 65.2	A6: 77.4
M1	39.9	37.9	69.2
M2	64.2	65.1	74.6
M3	59.7	65.0	54.2
A12b	49.1	45.6	70.4
A12c	61.9	*66.1*	74.6

that provides the segmentation that produces the lowest letter error rate in speech recognition. The letter error is calculated as the sum of the number of substituted, inserted, and deleted letters divided by the number of letters in the correct transcription of the data.

6.2 Training morph-based statistical language models

The language models were trained by using exactly the same text corpus which was previously used for extracting the original word list that each competitor had processed as the competition entry. This was not a coincidence, of course, because we wanted to have segmentations for all the different word forms to be able to use the whole corpus to train the optimal sub-word language models. Naturally, we could also have tried to split any words outside the given word list using the given morph lexicon and a Viterbi search for an optimal split, as explained in (Hirsimäki et al., 2006). However, this was not needed in this case.

Finnish. In the Finnish newspaper, book and newswire training corpus there were totally 40 M words and 1.6 M different word forms. After splitting the whole corpus into subwords and adding the word break symbols to assist the language model, n-gram language models were trained as if the units were word sequences. The language model used resembled the traditional n-gram model as used in (Hirsimäki et al., 2006), but instead of a fixed maximum value for n, the n was allowed to be optimized for each sequence context using the growing n-gram algorithm (Siivola and Pellom, 2005). The idea in this approach is to start from unigrams and gradually add those n-grams that maximize the training set likelihood with respect to the increase of the model size. In addition to controlling the memory consumption for training and recognition, restricting the model complexity is important also to avoid over-learning, because natural language corpora are always very sparse, even if morph units are utilized.

Turkish. In Turkish training corpus, there are totally 16.6 M words and 583 K different word forms. For language modeling and perplexity experiments, we used the SRI Language Modeling toolkit (Stolcke, 2002). We used interpolated modified Kneser-Ney smoothing to assign probabilities to zero probability strings. 4-gram language models are generated for each model. Entropy based pruning (Stolcke, 1998) with a pruning constant of 10^{-8} is applied to each model to reduce the model size.

Model size limitation. Despite the originally given upper limit for the lexicon size 300,000, we decided to accept submitted morph lexicons that exceeded the limit. In fact, to achieve comparable models, we only controlled the final size of the language models. For practical reasons in training and recognition, the size was set to approximately 10 million n-grams in Finnish and 50-70 Mbytes in Turkish.

6.3 Using cross-entropy to measure modeling accuracy

One way to directly evaluate the accuracy of a language model is to compute the average probability of an independent test text. To obtain a useful comparison measure, this probability is normalized by the number of words in the text. Typical comparison measures derived from this normalized probability are *perplexity* and *cross-entropy*. For this competition we chose cross-entropy, which is the logarithmic version (log2) of perplexity.

Given the held-out text data T consisting of W_T

words and a language model M, the *cross-entropy* $H_M(T)$ was computed as:

$$H_M(T) = -\frac{1}{W_T} \log_2 P(T|M) \qquad (4)$$

Here it is important that it is normalized by the number of *words*, not morphs, because a different morph lexicon was used for each model and, thus, the number of morphs in the test text varied.

Table 5: The obtained LM performance for the submitted segmentations in Finnish (Competition 2). CE is the average cross-entropy in the test text. Note that the cross-entropy is the logarithm of perplexity. As low a value as possible is desirable. OOV is the average out-of-vocabulary rate in the test text. The additional references at the bottom are explained in section 6.6.

Finnish	CE	OOV	Lexicon size
A1	13.65	0.36	297 981
A2a	13.54	0.03	73 178
A2b	13.63	0.04	65 557
A4a	13.55	2.70	609 458
A4b	*12.93*	0.99	1 559 199
A5	13.50	1.24	650 154
A7	13.81	0.85	530 543
A9	13.78	0.95	615 809
A11	13.59	0.58	690 601
A12a	13.66	0.40	317 870
M1	13.59	0.02	121 862
M2	13.53	0.08	155 065
M3	13.53	0.16	164 311
A12b	13.45	0.47	355 145
A12c	13.58	0.14	171 663

Some additional references			
Finnish	CE	OOV	Lexicon size
M1 26k	13.62	0.00	26 935
G1	13.62	0.03	69 929
G2	13.31	0.61	368 412
W1	13.95	0.00	394 266
W2	*12.04*	5.47	410 001

Table 5 shows the obtained cross-entropies on a test text of 50,000 Finnish sentences that was randomly selected from our text corpus and held-out from the training. Although the unsupervised morph lexicons were designed to process all words, there was a small OOV (out-of-vocabulary rate) in the test text. The OOV is shown in the table, because the higher it is, the more it affects the perplexity and cross-entropy by making it look smaller than it actually would be, if the OOV was zero.

Table 6: The obtained LM performance for the submitted segmentations in Turkish (Competition 2). CE is the average cross-entropy in the test text. The OOV rate was zero, because all OOVs were split into letters. # subwords is the ratio of the number of subwords in test text to the number of words.

Turkish	CE	# subwords	Lexicon size
A1	15.49	2.92	121 942
A2a	14.22	2.42	48 619
A2b	15.28	2.87	37 253
A4a	14.92	2.66	204 555
A4b	14.23	2.23	561 905
A5	15.29	3.03	195 487
A7	14.60	2.61	189 239
A9	16.05	2.89	218 320
A11	*13.83*	2.04	264 502
A12a	15.19	2.77	148 650
M1	13.99	2.30	51 542
M2	14.96	2.79	96 182
M3	14.73	2.70	88 429

Table 6 shows the performance on a Turkish test text consisting of 553 newspaper sentences (6989 words). If the segmentation of a test word was available in the segmentation list, we split that word into the corresponding subwords. Otherwise, the test word was left as a whole. In all of the submissions, the lexicon contained the individual letters of the Turkish alphabet as morphs. Therefore, the OOV rates were zero.

6.4 Large-vocabulary continuous speech recognition tests

The objective of Competition 2 was to evaluate the word splits in an application that would be as realistic as possible. When we originally planned this competition, we hesitated to choose speech recognition, because we thought it would take too much effort to build a set of state-of-art large-vocabulary continuous speech recognizers just for this evaluation. However, this was in line with our other research objectives and we have recently built several corresponding morph-based evaluation systems for Finnish, Estonian and Turkish (Hirsimäki et al., 2006; Siivola and Pellom, 2005; Kurimo et al., 2006).

Finnish. The speech recognizer consists of four main components: Acoustic phoneme models, language models, a lexicon and a decoder. For the acoustic models we chose the same speaker and context-dependent cross-word triphones with explicit phone duration models as for the Finnish models in (Kurimo et al., 2006) and also the same decoder (Pylkkönen, 2005). The real time factors were measured on 2.2 GHz CPU.

The lexicon and language models were created from the word splits of each competition participant and differed a little from the earlier morph models. The Finnish speech data utilized for recognizer training and evaluation was exactly the same book reading corpus as in (Hirsimäki et al., 2006; Kurimo et al., 2006). The speaker-dependent reading recognition is not the most difficult large-vocabulary recognition task as can be seen from the rather low error rates obtained, but it suits well to the scope of the Finnish language model training data and has several interesting previous benchmark results.

In a complete speech recognizer there is an almost endless amount of parameter "tweaking" in order to tune the performance, speed, memory consumption, hypothesis pruning etc., not to mention the various parameters tuned for training the models. To save effort we adopted as much as possible the same parameters as in the previous works (Hirsimäki et al., 2006; Siivola and Pellom, 2005; Kurimo et al., 2006) even if they were perhaps not exactly optimal for the new models. The only parameter that we optimized individually for each competitor was the weighting factor between the acoustic and language model to achieve the best performance on a held-out development set.

Turkish. The Turkish language models were evaluated by our Turkish large-vocabulary continuous speech recognizer. The main difference to the Finnish system were the speaker-independent acoustic models, the HTK frontend (Young et al., 2002) and that no explicit phone duration models were applied. The acoustic training data contained 40 hours of speech from 550 different speakers. The Turkish evaluation was performed using another decoder (Mohri and Riley, 2002) on a 2.4GHz CPU. The recognition task consisted of approximately one hour of newspaper sentences read by one female speaker.

Table 7: The obtained speech recognition performance the submitted segmentations in Finnish (Competition 2). The main measure here is the letter error rate LER. The additional references at the bottom are explained in section 6.6.

Finnish	LER %	WER %	RTF
A1	1.42	10.58	17.67
A2a	1.39	9.53	12.88
A2b	*1.32*	*9.47*	15.92
A4a	*1.32*	9.81	15.59
A4b	1.64	13.54	10.89
A5	1.88	13.10	13.55
A7	1.55	11.33	13.97
A9	1.59	11.71	16.31
A11	1.45	11.17	*10.10*
A12a	1.40	10.72	15.65
Winner	A2b, A4a	A2b: 9.47	A11: 10.10
M1	1.31	9.84	12.34
M2	1.32	10.18	14.38
M3	*1.30*	10.05	15.64
A12b	1.31	10.12	12.01
A12c	*1.25*	9.80	13.60

Some additional references			
Finnish	LER %	WER %	RTF
M1 26k	1.55	10.67	9.51
G1	1.33	9.60	10.58
G2	1.34	9.88	11.74
W1	1.37	10.83	11.84
W2	2.07	17.86	*7.42*

6.5 Results of Competition 2

The results of the speech recognition evaluation are shown in Table 7 (Finnish) and Table 8 (Turkish). The main performance measure is the letter error rate (LER). The word error rate (WER) was computed, too, because it is a more common measure although not so useful for the very variable-length words in Finnish. Another interesting figure is the recognition speed measured by the real-time factor (RTF).

In the Finnish task, the winners of Competition 2 were the models obtained from algorithm A2b from TIMC-IMAG in France and A4a from the University of Leipzig. The next competitors were not far behind: A2a from TIMC-IMAG, A12a and A1 from University of Leeds. The Morfessors M1, M2 and M3 were all very close to the winner. Among the top 5 models and the refer-

Table 8: The obtained speech recognition performance for the submitted segmentations in Turkish (Competition 2). The main measure here is the letter error rate LER.

Turkish	LER %	WER %	RTF
A1	15.0	43.0	2.68
A2a	13.6	38.9	2.15
A2b	*13.4*	*37.5*	2.19
A4a	15.7	46.3	2.43
A4b	16.7	50.2	*1.75*
A5	13.5	38.9	2.46
A7	13.8	40.3	2.33
A9	16.9	47.7	3.03
A11	14.6	41.4	1.85
A12a	14.5	41.9	2.56
Winner	A2b: 13.4	A2b: 37.5	A4b: 1.75
M1	13.7	39.4	1.98
M2	14.3	41.2	2.10
M3	*13.2*	*37.2*	1.89

ences, A2a and M1 differ from the others by being somewhat faster to run. However, the sixth best model A11 from the University of Dalhousie in Canada is clearly faster to run than the top five.

A2b from TIMC-IMAG won also Competition 2 for Turkish, but A5 from Leeds and A2a from TIMC-IMAG were very close. The Morfessor M3 produced the lowest error rates.

Since the best speech recognition error rates were not far apart, we performed the Wilcoxon's Signed-Rank test as in (Hirsimäki et al., 2006) pairwise between every algorithm pairs to see which differences are also statistically significant. For the Finnish data the best Morfessor M3 was significantly better than M1, A9, A5 and A4b. The winners of the competition A2b and A4a were both significantly better than A12a, A11, A9, A7, A5, A4b and A1.

6.6 Comparisons to previous references

It is also interesting to compare the current results to our earlier benchmarks. In (Hirsimäki et al., 2006) we compared pruned Morfessor baseline M1 morphs (26k and 66k lexicon) to grammatical (gold-standard) morphs (79k) and a large word-based lexicon (410k). The letter error rates in the same evaluation data were then: 4.21, 4.35, 4.57 and 6.14. However, those experiments were run in 2004 and since then we have improved the whole recognition system in many ways.

Table 9: Some additional references. In "letters" all OOVs are split to letters and in "skip" they are just left out.

Name	Info	OOV
M1 26k	A small lexicon Morfessor	letters
G1	Gold-standard morphs	letters
G2	Gold-standard morphs	skip
W1	Large word lexicon	letters
W2	Large word lexicon	skip

In (Kurimo et al., 2006) the results of the pruned Morfessor baseline M1 morphs (26k) and the large word-based lexicon (400k) in almost the same setup as in Table 7 were LER: 0.95 and 1.20; and WER: 7.0 and 8.5. The main difference was that the language models were trained such that any OOVs were modeled letter-by-letter, the training data was significantly extended (150 M words instead 40 M) and the language models were much larger (50 M n-grams instead of 10 M).

Inspired by the comparison to earlier results, we computed five additional language models for the current setup: Two from grammatical (gold-standard) morphs (79k lexicon), one pruned Morfessor baseline M1 (26k), and two large word-based lexicon (400k), see Table 9. These were all squeezed into the standard size (about 10 M n-grams) and trained with the same older (40 M) training text corpus. The results are in Table 5 and Table 7. The gold-standard morphs (G1) and the word lexicon (W1) seem to be very close in performance to the M1, but the pruned M1 (26k) has a slightly higher error rate. However, if the OOVs (the words that cannot be segmented by the lexicon) are skipped as we did for other algorithms in the Finnish part of the Competition 2, the error rates grow and cross-entropies shrink, especially for the word lexicon (W2) because of the much higher OOV rate than for any other model.

7 Conclusions

The objective of the Challenge was to design a statistical machine learning algorithm that segments words into the smallest meaning-bearing units of language, morphemes. Ideally, these are basic vocabulary units suitable for different tasks, such as speech and text understanding, machine translation, information retrieval, and statistical language modeling.

The scientific goals of this Challenge were:

- To learn of the phenomena underlying word construction in natural languages
- To discover approaches suitable for a wide range of languages
- To advance machine learning methodology

14 different segmentation algorithms from 12 research groups were submitted and evaluated. The evaluations included 3 different languages: Finnish, Turkish and English. The algorithms and results were presented in Challenge Workshop, arranged in connection with other PASCAL Challenges on machine learning, April 10-12, 2006.

8 Acknowledgments

We thank all the participants for their submissions and enthusiasm. We owe great thanks as well to the organizers of the PASCAL Challenge Program who helped us organize this challenge and the challenge workshop. Our work was supported by the Academy of Finland in the projects *New information processing principles*, *Adaptive Informatics* and *New adaptive and learning methods in speech recognition*. Funding was also provided by the Graduate School of Language Technology in Finland. We thank the Finnish Federation of the Visually Impaired, Departments of Speech Science and for providing the speech data. We are also grateful to the Finnish news agency (STT) and the Finnish IT center for science (CSC) for the text data. The authors would like thank Sabanci and ODTU universities for the Turkish acoustic and text data and AT&T Labs – Research for the software. This work was supported in part by the IST Programme of the European Community, under the PASCAL Network of Excellence, IST-2002-506778. This publication only reflects the authors' views. We acknowledge that access rights to data and other materials are restricted due to other commitments.

References

M. Creutz and K. Lagus. 2002. Unsupervised discovery of morphemes. In *Proceedings of the Workshop on Morphological and Phonological Learning of ACL-02*, pages 21–30.

M. Creutz and K. Lagus. 2004. Induction of a simple morphology for highly-inflecting languages. In *Proceedings of 7th Meeting of the ACL Special Interest Group in Computational Phonology (SIGPHON)*, pages 43–51, Barcelona, Spain.

M. Creutz and K. Lagus. 2005a. Inducing the morphological lexicon of a natural language from unannotated text. In *Proceedings of the International and Interdisciplinary Conference on Adaptive Knowledge Representation and Reasoning (AKRR'05)*, pages 106–113, Espoo, Finland.

M. Creutz and K. Lagus. 2005b. Unsupervised morpheme segmentation and morphology induction from text corpora using Morfessor. Technical Report A81, Publications in Computer and Information Science, Helsinki University of Technology. URL: http://www.cis.hut.fi/projects/morpho/.

M. Creutz and K. Linden. 2004. Morpheme segmentation gold standards for Finnish and English. Technical Report A77, Publications in Computer and Information Science, Helsinki University of Technology. URL: http://www.cis.hut.fi/projects/morpho/.

Ozlem Cetinoglu 2000. Prolog based natural language processing infrastructure for Turkish. M.Sc. thesis Bogazici University, Istanbul, Turkey.

Helin Dutagaci 2002. Statistical Language Models for Large Vocabulary Continuous Speech Recognition of Turkish. M.Sc. thesis Bogazici University, Istanbul, Turkey.

P. Geutner. 1995. Using morphology towards better large-vocabulary speech recognition systems. In *Proceedings of the IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, pages 445–448, Detroit, Michigan.

T. Hirsimäki, M. Creutz, V. Siivola, M. Kurimo, S. Virpioja, and J. Pylkkönen. 2006. Unlimited vocabulary speech recognition with morph language models applied to Finnish. *Computer Speech and Language*. (In press).

M. Kurimo, A. Puurula, E. Arisoy, V. Siivola, T. Hirsimäki, J. Pylkkönen, T. Alumäe, and M. Saraclar. 2006. Unlimited vocabulary speech recognition for agglutinative languages. In *Proceedings of the Human Language Technology, Conference of the North American Chapter of the Association for Computational Linguistics (HLT-NAACL)*, New York, USA.

Y.-S. Lee. 2004. Morphological analysis for statistical machine translation. In *Proceedings of the Human Language Technology, Conference of the North American Chapter of the Association for Computational Linguistics (HLT-NAACL)*, Boston, MA, USA.

M. Mohri and M. D. Riley. 2002. DCD library, speech recognition decoder library, AT&T Labs research. http://www.research.att.com/sw/tools/dcd/.

J. Pylkkönen. 2005. New pruning criteria for efficient decoding. In *Proceedings of the 9th European Conference on Speech Communication and Technology (Eurospeech)*, pages 581–584, Lisboa, Portugal.

R. Quirk, S. Greenbaum, G. Leech, and J. Svartvik. 1985. *A Comprehensive Grammar of the English Language*. Longman, Essex.

V. Siivola and B. Pellom. 2005. Growing an n-gram language model. In *Proceedings of 9th European Conference on Speech Communication and Technology*.

A. Stolcke. 1998. Entropy-based pruning of backoff language models. In *Proc. DARPA Broadcast News Transcription and Understanding Workshop*, pages 270–274, Lansdowne, VA.

A. Stolcke. 2002. Srilm – an extensible language modeling toolkit. In *Proc. Intl. Conf. on Spoken Language Processing*, pages 901–904, Denver.

S. Young, D. Ollason, V. Valtchev, and P. Woodland. 2002. The HTK book (for HTK version 3.2.).

Y.L. Zieman and H.L. Bleich. 1997. Conceptual mapping of user's queries to medical subject headings. In *Proceedings of the 1997 American Medical Informatics Association (AMIA) Annual Fall Symposium*.

Morfessor in the Morpho Challenge

Mathias Creutz and **Krista Lagus**
Helsinki University of Technology, Adaptive Informatics Research Centre
P. O. Box 5400, FIN-02105 HUT, Finland
{mathias.creutz, krista.lagus}@hut.fi

Abstract

In this work, Morfessor, a morpheme segmentation model and algorithm developed by the organizers of the Morpho Challenge, is outlined and references are made to earlier work. Although Morfessor does not take part in the official Challenge competition, we report experimental results for the morpheme segmentation of English, Finnish and Turkish words. The obtained results are very good. Morfessor outperforms the other algorithms in the Finnish and Turkish tasks and comes second in the English task. In the Finnish speech recognition task, Morfessor achieves the lowest letter error rate.

1 Introduction

This paper briefly describes three consecutive steps in the development of a morpheme segmentation and simple morphology induction algorithm, called *Morfessor*. Morfessor has been developed by the organizers of the Morpho Challenge and was therefore excluded from the official competition. However, we believe that the performance of Morfessor in the Morpho Challenge task will be of interest to a broader audience than the current authors, especially since the obtained results are generally very good.

The readers should keep in mind that a comparison of Morfessor to its competitors is not entirely fair, since portions of the Finnish and English data sets used in the competition have been utilized during the development of the Morfessor model. It is thus probable that the model implementation to some degree reflects properties of these very data sets. Nevertheless, the data set of the third language, Turkish, is as new to the organizers as to the participants. No modifications to the tested versions of the Morfessor model have been made after the acquisition of the Turkish data.

In the following sections, some characteristics of the Morfessor model will be outlined and experimental results obtained in the morpheme segmentation as well as Finnish speech recognition task will be reported and discussed.

2 Characterization of the Morfessor model

Morfessor is an unsupervised method for the segmentation of words into morpheme-like units. The general idea behind the Morfessor model is to discover as compact a description of the data as possible. Substrings occurring frequently enough in several different word forms are proposed as *morphs* and the words are then represented as a concatenation of morphs, e.g., “hand, hand+s, left+hand+ed, hand+ful”.

An optimal balance is sought between compactness of the *morph lexicon* versus the compactness of the representation of the *corpus*. The morph lexicon is a list of all distinct morphs (e.g., “hand, s, left, ed, ful”) together with some stored properties of these morphs. The representation of the corpus can be seen as a sequence of pointers to entries in the morph lexicon; e.g. the word “left-handed” is represented as three pointers to morphs in the lexicon.

A very compact lexicon could consist of the individual letters of the language. However, this would result in a very expensive representation of the corpus, since every word would be broken down into as many morphs as the number of letters it contains. The opposite situation consists of having a short representation of the corpus (e.g., no words would be split into parts), but then the lexicon would necessarily be very large, since it would have to contain all distinct words that occur in the corpus. Thus, the optimal solution is usually a compromise between these two extremes.

Among others, de Marcken (1996), Brent (1999), Goldsmith (2001), and Creutz and Lagus (2002; 2003; 2004; 2005a; 2006) have shown that the above type of model produces segmentations that resemble linguistic morpheme segmentations, when formulated mathematically in a probabilistic framework or equivalently using the Minimum Description Length (MDL) principle (Rissanen, 1989).

An alternative popular approach to the segmentation of words and phrases is based on the works by Zellig S. Harris (1955; 1967). For instance, Schone and Jurafsky (2000; 2001) make use of a Harrisian approach to suggest word stems and suffixes. In this approach, word or morpheme boundaries are proposed at locations where the predictability of the next letter in a letter sequence is low. Such a model does not use compactness of representation as an explicit optimization criterion. Other related work is described more thoroughly in our previous publications.

Next, the three tested versions of the Morfessor model will be described briefly. These versions are called *Morfessor Baseline*, *Morfessor Categories-ML*, and *Morfessor Categories-MAP*. The versions correspond to chronological development steps, starting with the simplest model and ending with the most complex one. For a discussion on how the early versions can be seen as special cases of the latest model, the reader is encouraged to consult (Creutz and Lagus, 2006). Note that the current paper merely presents the underlying ideas and characteristics of the Morfessor model; in order to find an exact mathematical formulation it is necessary to read our previous works.

2.1 Morfessor Baseline

The Morfessor Baseline algorithm was originally introduced in (Creutz and Lagus, 2002), where it was called the “Recursive MDL” method. Additionally, the Baseline algorithm is described in (Creutz and Lagus, 2005b; Hirsimäki et al., 2006). The implementing computer program is publicly available for download at `http://www.cis.hut.fi/projects/morpho/`.

The Baseline method is a *context-independent* splitting algorithm. It is used as a baseline, or initialization, for the later *context-dependent* model versions (Categories-ML and Categories-MAP). In slightly simplified form, the optimization criterion utilized in Morfessor Baseline corresponds to the maximization of the following posterior probability:

$$
\begin{aligned}
&P(\text{lexicon} \mid \text{corpus}) \propto \\
&P(\text{lexicon})P(\text{corpus} \mid \text{lexicon}) = \\
&\qquad \prod_{\text{letters } \alpha} P(\alpha) \cdot \prod_{\text{morphs } \mu} P(\mu). \qquad (1)
\end{aligned}
$$

The lexicon consists of all distinct morphs spelled out; this forms a long string of letters α. The probability of the lexicon is the product of the probability of each letter in this string. Analogously, the corpus is represented as a sequence of morphs, which corresponds to a particular segmentation of the words in the corpus. The probability of this segmentation equals the product of the probability of each morph token μ. Letter and morph probabilities are maximum likelihood estimates.

When segmentations produced by the Baseline method are compared to linguistic morpheme segmentations, the algorithm suffers from three types of fairly common errors: *undersegmentation* of frequent strings, *oversegmentation* of rare strings, and *morphotactic violations*. This follows from the fact that the most concise representation is obtained when any frequent string is stored as a whole in the lexicon (e.g., English “having, soldiers, states, seemed”), whereas infrequent strings are better coded in parts (e.g., “or+p+han, s+ed+it+ious, vol+can+o”). Morphotactic violations are a consequence of the context-independent nature of the model: For instance, the morphs “-s” and “-ed” are frequently occurring *suffixes* in the English language, but the algorithm occasionally suggests them in word-initial position as *prefixes* (“s+wing, ed+ward, s+urge+on”).

2.2 Morfessor Categories-ML

Morfessor Categories-ML (Creutz and Lagus, 2004) introduces morph categories. The segmentation of the corpus is modeled using a Hidden Markov Model (HMM) with transition probabilities between categories and emission probabilities of morphs from categories (see Fig. 1). Three categories are used: *prefix*, *stem*, and *suffix* and an additional *non-morpheme* (or *noise*) category. Some distributional properties of the morphs in a proposed segmentation of the corpus are used for determining category-to-morph emission probabilities. A morph that is observed to precede a large number of different morphs is a likely prefix (e.g., English “re-, un-, mis-”); this is measured by *right perplexity* (Fig. 2a). Correspondingly, a morph that is observed to follow a large set of

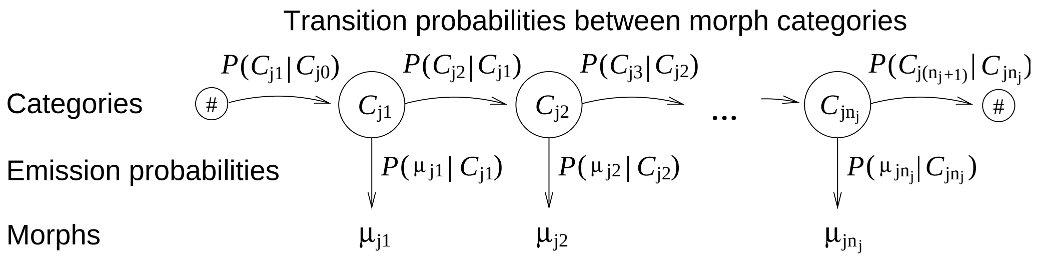


Figure 1: Hidden Markov model used in Categories-ML and Categories-MAP to compute $P(\text{corpus}\,|\,\text{lexicon})$. The picture shows the HMM representing one word in the corpus (the j^{th} word, which has been split into n_j morphs). The word consists of a sequence of morphs $\mu_{j\cdot}$ which are emitted from latent categories $C_{j\cdot}$. Note that the transition probabilities comprise transitions from and to a special word boundary category (#).

morphs is likely to be a suffix (e.g., "-s, -ed, -ing"); this is measured by *left-perplexity* (Fig. 2b). A morph that is not very short is likely to be a stem (e.g., "friend, hannibal, poison"); see Fig. 2c. A morph that is not an obvious prefix, stem, or suffix in the position it occurs may be an indication of an erroneous segmentation. Such morphs are tagged as noise (e.g., all morphs in the segmentation "vol+can+o").

The identification of "noise' and likely erroneous segmentations makes it possible to apply some heuristics in order to partly remedy the shortcomings of Morfessor Baseline. Undersegmentation is reduced by forcing splits of redundant morphs in the lexicon. These morphs consist of other morphs that are also present in the lexicon (e.g., "seemed = seem+ed"). Some restrictions apply, such that splitting into noise morphs is prohibited. The opposite problem, oversegmentation, is alleviated by joining morphs tagged as noise with their neighbors (e.g, "vol+can+o" becomes "volcano"). Morphotactic violations are less likely to occur due to the context-sensitivity of the HMM model.

2.3 Morfessor Categories-MAP

The Categories-MAP model version (Creutz and Lagus, 2005a) emerged in an attempt to reformulate Categories-ML in a more elegant fashion. In Categories-ML, the optimal segmentation of the corpus is sought through Maximum Likelihood (ML) re-estimation, whereas the complexity of the lexicon is controlled heuristically. In a Maximum a Posteriori (MAP) model, an explicit probability is calculated for both the lexicon and the representation of the corpus conditioned on the lexicon. Categories-MAP and the Baseline method are MAP models.

The most important new feature of the Categories-MAP model is that the lexicon may contain hierarchical entries. That is, a morph can either consist of a string of letters (as in the previous models) or of two submorphs, which can recursively consist of submorphs.

As was the case in the Baseline model, frequent strings typically end up as entries of their own in the lexicon (e.g, the English word "straightforwardness"). However, unlike in the Baseline model, these frequent strings now have a hierarchical representation; see Figure 3. In a morpheme segmentation task, the existence of this inner structure makes it possible to "expand" morphs into their submorphs, thereby avoiding undersegmentation. Since every morph at every level is tagged with its most likely category, it is possible to avoid *over*segmentation as well, since one can refrain from expanding nodes in the tree if the next level contains *non-morphemes*, i.e. "noise morphs". For instance, in Figure 3, the word "straightforwardness" is expanded into "straight+forward+ness". The morph "forward" is not expanded into its constituents "for+ward" (although this may have been appropriate), because "for" is tagged as a non-morpheme in the current context.

3 Morpheme Segmentation Experiments

In the following, some differences between the tested versions of Morfessor as well as the three tested languages are illustrated in the light of experimental results. The experiments were run on the datasets provided in the Challenge. The

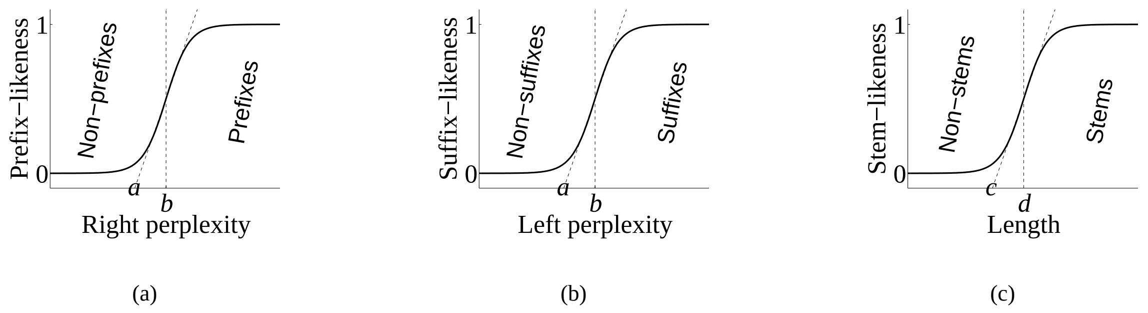


Figure 2: Sketch of sigmoid functions (used in the Categories models), which express how the right and left perplexity as well as the length of a morph affect its tendency to function as a prefix, suffix, or stem. The parameters a, b, c, d determine the shape of the sigmoids. A probability distribution is obtained by first computing the probability that a morph μ belongs to *none* of the three categories. The probability of this so-called non-morpheme, or noise, category given the morph μ equals: $(1 - \textit{prefix-like}(\mu)) \cdot (1 - \textit{suffix-like}(\mu)) \cdot (1 - \textit{stem-like}(\mu))$. Then the remaining probability mass is distributed between prefix, stem and suffix proportionally to the prefix-, stem- and suffix-likeness values.

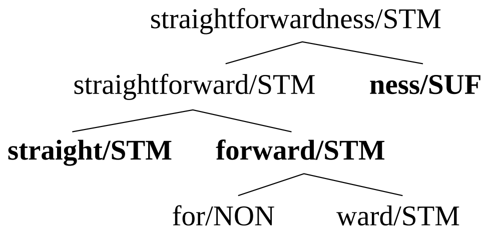


Figure 3: Hierarchical representation of the English word "straightforwardness" in the lexicon induced by Morfessor Categories-MAP. Each morph has been tagged with a category: stem (STM), suffix (SUF), or non-morpheme (NON). (No morph was tagged as a prefix in this example.) The finest resolution that does not contain non-morphemes is rendered using a bold-face font. This corresponds to the proposed morpheme segmentation.

Morfessor Baseline algorithm is entirely unsupervised and does not require that any parameters be set. The Categories algorithms have one parameter (the perplexity threshold b in Fig. 2) that needs to be set to an appropriate value for optimal performance. This parameter value was optimized separately for each language on the small development sets (model segmentations) provided.[1]

[1] A fixed (dataset-independent) scheme works fine for the other parameters in Fig. 2: $a = 10/b, c = 2, d = 3.5$. This is good, since the amount of necessary supervision should be kept to a minimum.

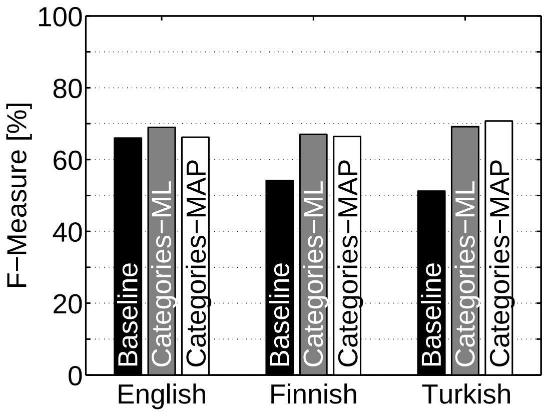


Figure 4: F-measures computed for the placement of morpheme boundaries in relation to linguistic morpheme segmentations, obtained by the three different versions of Morfessor on the three test languages.

3.1 Results

The morpheme segmentation task of the competition is won by the participant achieving the highest *F-measure* of correctly placed morpheme boundaries. Figure 4 shows the F-measures of the three Morfessor methods on the three tested languages. The F-measure is the harmonic mean of *precision* and *recall*. The precisions and recalls obtained by Morfessor are displayed in Figures 5 and 6, respectively.

The results show that there are different tendencies for the English data, on the one hand, and the

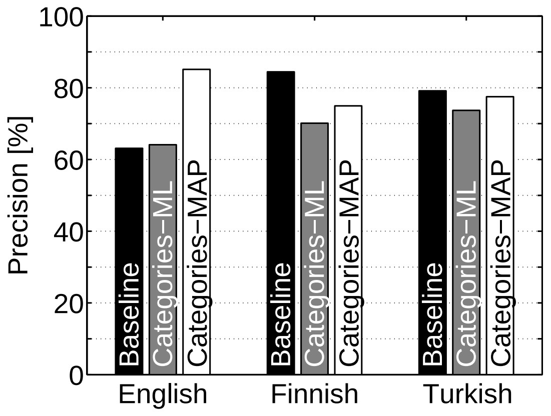


Figure 5: Precision of the three Morfessor methods on the three languages tested.

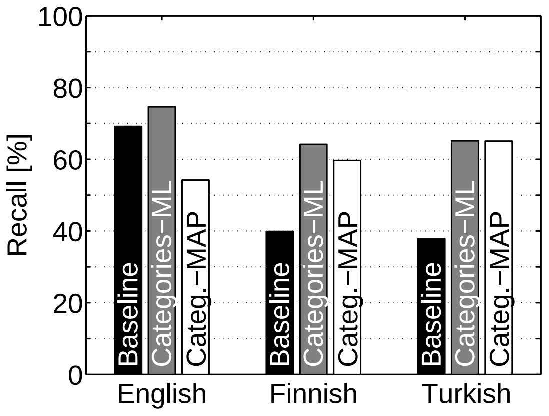


Figure 6: Recall of the three Morfessor methods on the three languages tested.

Finnish and Turkish data, on the other hand. For Finnish and Turkish, the context-dependent Categories models produce clear improvements over the context-independent Baseline splitting algorithm (with F-measures 10 – 20 points higher; Fig 4). For English, the improvement is minor, but on the other hand the Baseline here attains a considerably higher level than for Finnish and Turkish. The best F-measure obtained by Morfessor for all three languages is at the same level, around 70 %.

The precision and recall plots in Figures 5 and 6 provide more detailed information. For English, even though the F-measures of all three algorithms are approximately equal, the produced segmentations are very different. Categories-MAP has a significantly higher precision than the other model versions (and correspondingly a lower recall). For Finnish and Turkish, the Categories models display a great improvement of recall in relation to the Baseline method. This comes at the expense of lower precision, which is observed for Finnish and to a lesser degree on the Turkish data.

In order to better understand the differences observed in the results for the different languages, the output at various stages of the segmentation process has been studied for each of the Morfessor model variants. No obvious explanation has been found other than the difference in the morphological structures of the languages. Finnish and Turkish are predominantly agglutinative languages, in which words are formed through the concatenation of morphemes. The type/token ratio is high, i.e., the number of different word forms encountered in a piece of running text is relatively high. By contrast, word forming in English involves fewer morphemes. The type/token ratio is lower, and the proportion of frequently occurring word forms is higher.

In the Finnish and Turkish segmentation task, Morfessor outperforms all algorithms proposed by the participants of the Morpho Challenge; compare the following F-measures for Finnish: 67.0 % (Morfessor Categories-ML) vs. 64.7 % (best participant), and for Turkish: 70.7 % (Morfessor Categories-MAP) vs. 65.3 % (best participant). In the English segmentation task, Morfessor comes second: 69.0 % (Morfessor Categories-ML) vs. 76.8 % (best participant).

4 Finnish Speech Recognition Experiments

N-gram language models have been estimated from the segmentations produced by the three Morfessor models on the Finnish data. The language models have been used in speech recognition experiments, and results are shown in Table 1. The evaluation of the language models alone (cross-entropy on a held-out data set) suggests that the Categories models are better than Morfessor Baseline, since their cross-entropy is lower. The cross-entropies do not, however, correlate with the actual speech recognition results. Categories-MAP obtains the lowest letter error rate (LER) – 1.30 % of the recognized letters are incorrect in comparison with the reference transcript – which is also lower than the letter error rate achieved by any participant of the Challenge (best result:

Table 1: Results from the Finnish speech recognition experiments: cross-entropy (log-perplexity) of the language models (H), letter error rate (LER) and word error rate (WER).

Method	H [bits]	LER [%]	WER [%]
Baseline	13.59	1.31	9.84
Categ.-ML	13.53	1.32	10.18
Categ.-MAP	13.53	1.30	10.05

1.32 %). Nevertheless, the word error rate (WER) of Categories-MAP is higher than that of Morfessor Baseline and the WER:s of three participants. This suggests that the letter errors made by Categories-MAP are spread over a larger number of words, which increases WER, whereas the other methods have a concentration of errors on a smaller set of words.

5 Conclusions

In the morpheme segmentation task, the current versions of Morfessor attain an F-measure value of about 70 % for all three tested languages. For English, a language with "poorer" morphology and less morpheme boundaries to discover, the simple Baseline method seems to almost reach to this level. The characteristically agglutinative languages Finnish and Turkish, which have "richer" morphology and a larger number of morpheme boundaries to be detected, require more complex models (the context-sensitive Categories model) to perform on the same level. It is particularly encouraging to see that Morfessor performs so well in the Turkish segmentation task, since Turkish data was never used in the development of the model.

References

M. R. Brent. 1999. An efficient, probabilistically sound algorithm for segmentation and word discovery. *Machine Learning*, 34:71–105.

Mathias Creutz and Krista Lagus. 2002. Unsupervised discovery of morphemes. In *Proc. Workshop on Morphological and Phonological Learning of ACL'02*, pages 21–30, Philadelphia, Pennsylvania, USA.

Mathias Creutz and Krista Lagus. 2004. Induction of a simple morphology for highly-inflecting languages. In *Proc. 7th Meeting of the ACL Special Interest Group in Computational Phonology (SIGPHON)*, pages 43–51, Barcelona, July.

Mathias Creutz and Krista Lagus. 2005a. Inducing the morphological lexicon of a natural language from unannotated text. In *Proceedings of the International and Interdisciplinary Conference on Adaptive Knowledge Representation and Reasoning (AKRR'05)*.

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

Mathias Creutz and Krista Lagus. 2006. Unsupervised models for morpheme segmentation and morphology learning. *ACM Transactions on Speech and Language Processing*. (Accepted for publication).

Mathias Creutz. 2003. Unsupervised segmentation of words using prior distributions of morph length and frequency. In *Proc. ACL'03*, pages 280–287, Sapporo, Japan.

C. G. de Marcken. 1996. *Unsupervised Language Acquisition*. Ph.D. thesis, MIT.

John Goldsmith. 2001. Unsupervised learning of the morphology of a natural language. *Computational Linguistics*, 27(2):153–198.

Zellig S. Harris. 1955. From phoneme to morpheme. *Language*, 31(2):190–222. Reprinted 1970 in *Papers in Structural and Transformational Linguistics*, Reidel Publishing Company, Dordrecht, Holland.

Zellig S. Harris. 1967. Morpheme boundaries within words: Report on a computer test. *Transformations and Discourse Analysis Papers*, 73. Reprinted 1970 in *Papers in Structural and Transformational Linguistics*, Reidel Publishing Company, Dordrecht, Holland.

Teemu Hirsimäki, Mathias Creutz, Vesa Siivola, Mikko Kurimo, Sami Virpioja, and Janne Pylkkönen. 2006. Unlimited vocabulary speech recognition with morph language models applied to finnish. *Computer Speech and Language*. (In press).

Jorma Rissanen. 1989. *Stochastic Complexity in Statistical Inquiry*, volume 15. World Scientific Series in Computer Science, Singapore.

P. Schone and D. Jurafsky. 2000. Knowledge-free induction of morphology using Latent Semantic Analysis. In *Proc. CoNLL-2000 & LLL-2000*, pages 67–72.

P. Schone and D. Jurafsky. 2001. Knowledge-free induction of inflectional morphologies. In *Proc. NAACL-2001*.

Unsupervised Morphological Segmentation Based on Segment Predictability and Word Segments Alignment

Delphine Bernhard
TIMC-IMAG
Institut de l'Ingénierie et de l'Information de Santé
Faculté de Médecine
F-38706 LA TRONCHE cedex
Delphine.Bernhard@imag.fr

Abstract

Word segments are relevant cues for the automatic acquisition of semantic relationships from morphologically related words. Indeed, morphemes are the smallest meaning-bearing units. We present an unsupervised method for the segmentation of words into sub-units devised for this objective. The system relies on segment predictability to discover a set of prefixes and suffixes and performs word segments alignment to detect morpheme boundaries.

1 Introduction

Morphemes are defined as the minimal meaning bearing units. Knowledge of morphologically related terms is thus worthy for many applications. This is especially true for morphologically complex languages like German or Finnish or scientific and technical vocabulary like biomedical language. Some research has for instance been devoted to the use of morphological decomposition for text indexing and retrieval in the biomedical domain (Schulz et al., 2002) or the acquisition of semantic relationships from morphologically related words (Zweigenbaum and Grabar, 2000; Namer and Zweigenbaum, 2004; Claveau and L'Homme, 2005). Work on the system presented in this paper has been undertaken with the objective of retrieving semantic relationships from morphologically related words (i.e. words sharing the same stem) in technical and scientific domains. Contrary to Schulz et al. (2002) or Namer and Zweigenbaum (2004) we have not built a morphological analyser relying on a dictionary of affixes and stems. Rather, morphological structure is discovered from a raw list of words and the method is not dependent on a given language, nor on a given domain. Related work on morphology induction is discussed in section 2. Our method is detailed in section 3. Finally in section 4 we present the results obtained.

2 Related work

2.1 Methods relying on segment predictability

Segment predictability is one possible cue for word segmentation. Harris (1955) proposes to use the number of different phonemes following a given phoneme sequence: morpheme boundaries are identified when this number reaches a peak. This method has been extended to written texts by Hafer and Weiss (1974) and Déjean (1998). Similarly, Saffran et al. (1996) suggest that learners use drops in the transitional probabilities between syllables to identify word boundaries. Like Déjean (1998) we use segment predictability to identify prefixes and suffixes. However, rather than determining segment boundaries by counting the number of letters following a given substring, as suggested in (Harris, 1955), we have developed a variant of this method based on transitional probabilities, following the proposition made by Saffran et al. (1996) (see Section 3.1).

2.2 Strategies based on word comparison

Other methods for the identification of morphologically related words are based on word comparison to identify similar and dissimilar parts in words. Neuvel and Fulop (2002) perform alignments starting either on the left or right edge of words to discover similarities and differences between the words compared. These similarities and differences correspond to word-formation strategies which can be used to generate new words without resorting to the notion of morpheme. Similarly, Schone and Jurafsky (2001) insert words in a trie either in good or reverse order to easily discover places where words differ from one another. Substrings which repeatedly differentiate words are considered as potential affixes. These methods based on the identification of initial or final common substrings are fine for prefix of suffix discovery but insufficient for words formed by compounding. In order to overcome these shortcomings our system performs word comparisons which are not anchored on word boundaries but rather on a shared stem which can be found at any position in the word (see Section 3.3).

2.3 Methods based on optimisation

Paradigmatic series of morphemes are extracted by Goldsmith (2001) in the form of "signatures" which are sets of suffixes which appear with the same stem. The method makes use of minimum description length (MDL) analysis to measure how effectively the morphology encodes the corpus. MDL is used by Creutz and Lagus (2002) as well to split words for highly-inflecting and morphologically complex languages. Our method is not directly related to MDL-based methods though it heavily relies on word segment length and frequency. Zipf (1968, page 173) had already noticed that, as well as words, "the length of a morpheme tends to bear an inverse ratio to its relative frequency of occurrence". If we draw a parallel between words and morphemes, stems, which bear more meaning than affixes, are long and not so frequent while affixes are frequent and short[1]. Length is also used in the probabilistic framework proposed by Creutz and Lagus (2004) where the stem-likeness of a segment is function of its length. We use these general properties in a differential framework, drawing upon Saussure's theory that syntagmatically related elements (like morphemes contained in a word) are defined by the differences amongst them. So rather than focusing on absolute values, we rely on differences in length and frequency (1) to distinguish between stems and affixes: a stem is identified as the longest and less frequent segment and (2) to impose constraints on the segments identified within a word: affixes have to be shorter and more frequent than stems.

3 Description of the method

The aim of the method described is to segment words into labelled segments. We only consider concatenative morphology and assign the following categories to morphological segments: stem, prefix, suffix and linking element. The latter category is not used by methods described in the previous section, but we think it is linguistically motivated in the sense that classical syntagmatic definitions of prefixes and suffixes fail to encompass linking elements. Indeed, prefixes are found before stems, at the beginning of words and suffixes are found after stems, at the end of words; linking elements can never be found at word boundaries and are always preceded and followed either by a stem or by another affix. For instance, "-**o**-" in "hormon**o**therapy" is a linking element.

Moreover, similarly to Creutz and Lagus (2004) we use the syntagmatic definition of morphological categories to impose constraints on possible sequences of word segments. In the next sections, we detail the procedure used to learn word segments.

[1]See also (Vergne, 2005) for a method to distinguish function and content words based on differences in length and frequency.

3.1 Extraction of prefixes and suffixes

The input of the system is a plain wordlist *L*. The method does not make use of word frequency. The first step of the segmentation procedure is the extraction of a preliminary set of prefixes *P* and suffixes *S*. These are acquired using a method based on transitional probabilities between substrings. Moreover, only the longest words are segmented, following the intuition that these are the words most likely to be affixed. Words are sorted in reverse length order and are segmented using the variations of the transition probability between all the substrings coalescing at any given position *k* in the word.

Let *w* be a word whose boundaries are explicitly marked by the # symbol; n is the length of *w* (boundary markers included); $s_{i,j}$ is a substring of *w* starting at position *i* and ending at position *j*. For each position in the word *k* with k in [1, ..., n-1] we compute the following function, which corresponds to the mean of the maximum transition probabilities for all substrings ending and beginning at position *k*:

$$f(k) = \frac{\sum_{i=0}^{k-1} \sum_{j=k+1}^{n} \max[p(s_{i,k}|s_{k,j}), p(s_{k,j}|s_{i,k})]}{k \times (n-k)}$$

Where the transitional probabilities $p(s_{i,k}|s_{k,j})$ and $p(s_{k,j}|s_{i,k})$ are estimated by:

$$p(s_{i,k}|s_{k,j}) = \frac{f(s_{i,j})}{f(s_{k,j})} \quad \text{and} \quad p(s_{k,j}|s_{i,k}) = \frac{f(s_{i,j})}{f(s_{i,k})}$$

The frequency of a substring is equal to the number of times it occurs in *L*.

This yields a profile of the variations of the transition probabilities for *w*. Local minima indicate potential segment boundaries. A local minimum is validated if its difference both with the preceding and following maximum is at least equal to a standard deviation of the values. Figure 1 depicts this profile for the word "ultracentrifugation". Valid boundaries are indicated by a bold vertical line, which corresponds to the following word segmentation: *ultra* + *centrifug* + *ation*.

Once a word has been segmented, the longest and less frequent segment is identified as stem if it also appears at least twice in the lexicon and once at the beginning of a word. The substrings which directly precede and follow this stem in the wordlist are added to the lists *P* and *S* if they are shorter and more frequent than the stem. Moreover, we discard prefixes of length 1 since we have noticed that these lead to erroneous segmentations in further stages of the process.

It is not necessary to apply this process of affix acquisition to all words. Indeed, the number of new affixes acquired decreases as the number of segmented words augments. This procedure ends when for *N* running words less than half of the affixes learned do not already belong to the lists *P* and *S*. Table 1 lists the

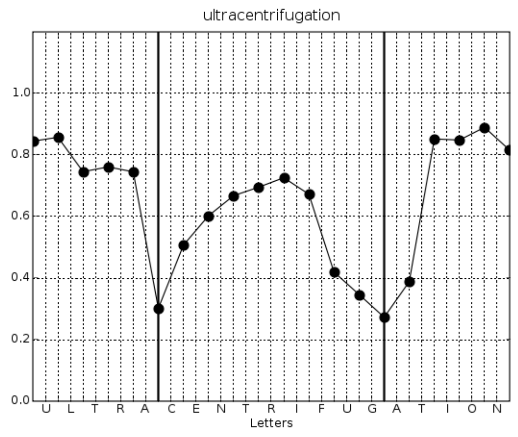


Figure 1: Profile for the variation of transitional probabilities for the word "ultracentrifugation"

most frequent prefixes and suffixes extracted from the MorphoChallenge English wordlist for N=5.

Prefixes		Suffixes	
in-	pre-	-s	-ly
un-	natur-	-e	-ble
inter-	counter-	-ed	-tion
dis-	over-	-al	-es
mis-	psycho-	-ally	-ately
re-	ultra-	-ing	-ity
ex-	hyper-	-ation	-l
pseudo-	con-	-ness	-ism

Table 1: Most frequent prefixes and suffixes extracted from the MorphoChallenge English wordlist for N=5.

3.2 Acquisition of stems

Stems are obtained by stripping off from each word in the list *L* all the possible combinations of the affixes previously acquired and the empty string. Of course this list is rather noisy. The following constraints are therefore applied on each extracted stem *s*:

1. it must have a minimum length of 3.
2. it can be followed by at least 2 different letters (including the word boundary marker); otherwise, this would mean that the stem is included in another stem.
3. it cannot contain a hyphen, since hyphens are boundary markers.
4. at least one word must begin with *s*.

3.3 Segmentation of words

Word segmentation is performed by comparing words containing the same stem *b* in order to find limits between shared and different segments (see Figure 2).

Segments thus obtained are assigned one of the three affix types (prefix, suffix, linking element) according

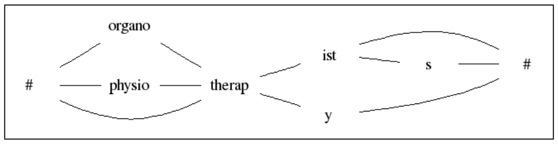


Figure 2: Example word segments alignment for the stem "therap".

to their position within the word, relatively to the stem. For instance, segments 'ist', 's' and 'y' in Figure 2 are labelled as suffixes. In order to deal with compounding, we make use of a temporary category for segments which contain another stem. These segments are labelled as 'potential stems'. This is the case for the segments 'organo' and 'physio' in Figure 2.

As a result of the alignment new affixes which do not belong to the lists *P* and *S* may be discovered and these have to be validated. The validation procedure is somewhat similar to the validation of new morphemes in (Déjean, 1998) and is performed as follows: amongst aligned words which share the same stem we form subgroups of words beginning with the same segment. Table 2 lists word-final segments for the sub-group of the words containing the stem "hous" and starting with the empty string prefix.

Words	Suffixes from list *S*	Potential stems	New suffixes
housekeeping		-ekeeping	
housing	-ing		
household		-ehold	
house's			-e's
house	-e		
housed	-ed		

Table 2: Word final segments for words containing the stem "hous" and starting with the empty string prefix.

Let $|A_1|$ be the number of suffixes from list *S*, $|A_2|$ the number of potential stem segments and $|A_3|$ the number of new suffixes. For the examples in Table 2 $|A_1|$=3, $|A_2|$= 2 and $|A_3|$=1. New suffixes and potential stem segments are validated only if the following conditions are met:

$$\frac{|A_1| + |A_2|}{|A_1| + |A_2| + |A_3|} \geq a \text{ and } \frac{|A_1|}{|A_1| + |A_2|} \geq b$$

The same procedure is applied for the validation of word-initial segments.

Valid segmentations for each word are stored: we thus keep trace of all the segments proposed for a word, since a word may contain more than one stem and may therefore be aligned and segmented more than once. When all stems have been analysed for segmentation, we examine the segments stored for each word and remove potential stem segments. Potential stem segments are either replaced by other segments (as a whole or only partially) or assigned a final category (prefix, suffix or linking element) if no replacement is possible. Finally, we compute a frequency of occurrence for each segment. Frequency of occurrence is equal to the number of different words whose analysis includes the segment considered.

3.4 Selection of the best segments

For each word, we have stored the labelled segments resulting from its successive segmentations (one segmentation per stem). In order to choose the best possible segments, we perform a best-first search privileging the most frequent segment, given a choice. The final segmentation must also obey constraints related to word structure (at least one stem amongst the segments, a prefix cannot be directly followed by a suffix, only one running linking element between two prefixes or suffixes) and to the frequency of the segments relatively to one another (stems must be less frequent than the other types of segments). At the end of this stage, each word in the list *L* is segmented. Another output of this stage is the list of selected segments associated with their category (prefix, stem, suffix, linking element) and the number of times they have been selected (this corresponds to segment frequency). This list of segments can be used to segment any word in the same language, as explained in the next section.

3.5 Using the list of learned segments

Given the list of the best segments selected in the previous stage of the method, it is possible to segment any list of words. This stage is therefore optional and is proposed as a solution for the segmentation of words which do not belong to the list of words from which segments have been learned. The A* algorithm is used to find the best segmentation for each word. The global cost for a segmentation is the sum of the costs associated with each segment s_i. We have used two different segment cost functions for MorphoChallenge resulting in two different submissions:

$$cost_1(s_i) = -log\frac{f(s_i)}{\sum_i f(s_i)}$$

$$cost_2(s_i) = -log\frac{f(s_i)}{\max_i[f(s_i)]}$$

Moreover, the same constraints on possible successions of word segments as those described in section 3.4 are used.

4 Results

There are two main ways of directly assessing the quality of the results, either by evaluating the conflation sets built out of morphologically related words sharing an identical stem or by evaluating the position of the boundaries within a word. The latter is used by MorphoChallenge 2005.

4.1 Conflation-based evaluation

We have performed an evaluation of the results of the method on a list of words extracted from an English corpus on breast cancer. This corpus has been automatically built from the Internet and contains about 86,000 different word forms. We have manually built morphological word families for the top 5,000 keywords in the corpus. Keywords have been identified by comparison with a corpus on volcanology, using the method described in (Rayson and Garside, 2000). For instance, one of the manually built morphological families contains the words “brachytherapy”, “chemoradiotherapy”, “chemotherapeutic”, “therapies”, “therapist”, etc. We have used conflation-based evaluation, since we wish to assess the ability of the method to retrieve words linked both by form and by meaning, which is closer to our objective of retrieving semantic relationships between words thanks to morphology. Evaluation consists in counting the number of correct, incorrect and missing pairs of morphologically related words. Words are considered as morphologically related if they contain the same stem according to the method. For instance, the words “chemoradiotherapy” and “therapist” form a correct pair of words. Precision is defined as the number of correct word pairs divided by the number of suggested word pairs. Recall is defined as the number of correct word pairs divided by the number of word pairs in the list of manually built morphological families. For this evaluation, we used the segmentations provided directly after selection of the best segments (see Section 3.4) with N=5, a=0.8 and b=0.1. Results are given in Table 3. Precision suffers from the fact that most words ending with -logy, -logic or -logical share the same stem “log” according to the system. Results also evidence that recall should be improved. For instance, “artery” is segmented as arter + y while “arterial” is segmented as arteri + al. Both words are therefore not conflated in the same set.

	Number	Example
Correct word pairs	3,936	lymphedematous lymphoedema
Incorrect word pairs	2,359	additive addresses
Missing word pairs	5,210	therapeutics therapy

Precision	Recall	F-measure
62.5	43.0	51.0

Table 3: Results of conflation-based evaluation.

4.2 MorphoChallenge 2005 results

The MorphoChallenge 2005 datasets were considerably bigger than the dataset used for the previous assessment. Word segments learning has been performed on the whole English dataset. However, for Finnish and

				F-measure			
				Sample evaluation		Final evaluation	
Language	N	a	b	method 1	method 2	method 1	method 2
English	5	0.85	0.1	64.29	61.05	66.6	62.4
Finnish	5	0.8	0.1	63.18	64.44	63.3	64.7
Turkish	5	0.7	0.1	55.93	66.06	55.3	65.3

Table 4: Parameter values used and results obtained for the submissions to MorphoChallenge 2005.

Turkish learning has been performed only on a subset of the datasets (the 300,000 most frequent words), due primarily to heavy memory consumption. Three different parameter values have to be set: N (see Section 3.1), a and b (see Section 3.3). Parameter values used for each language were roughly the same, only we took those values which yielded the best results on the evaluation datasets. Yet keeping default values N=5, a=0.8 and b=0.1 does not bring a change of more than about 1 or 2% in F-measures. Table 4 details parameter values used and the results obtained. Method 1 corresponds to results obtained by using $cost_1$ and method 2 to results obtained by using $cost_2$ (see Section 3.5).

Results for method 2, using $cost_2$, indicate better recall but lower precision on all datasets. This is especially noticeable on the Turkish dataset. Recall for the Turkish dataset was indeed an issue which led to the use of $cost_2$. This might be due to the fact that segments in the Turkish gold standard sample are shorter on the average than Finnish and English gold standard sample segments.

5 Conclusions

Thanks to MorphoChallenge 2005 the method has been tested on new languages (Finnish and Turkish), bigger wordlists and for different objectives (speech recognition). Results show that the method performs well even on Finnish and Turkish. Planned improvements include better implementation to deal with large datasets and incorporation of equivalence matching between stems to capture orthographic variants like "canc**e**r" and "canc**é**r". In work in progress, we are investigating the usefulness of morphological segmentation for the automatic acquisition of semantic relationships.

References

Vincent Claveau and Marie-Claude L'Homme. 2005. Structuring Terminology by Analogy-Based Machine Learning. In *Proceedings of the 7th International Conference on Terminology and Knowledge Engineering, TKE'05*.

Mathias Creutz and Krista Lagus. 2002. Unsupervised Discovery of Morphemes. In *Proceedings of the Workshop on Morphological and Phonological Learning of ACL-02*, pages 21–30.

Mathias Creutz and Krista Lagus. 2004. Induction of a Simple Morphology for Highly-Inflecting Languages. In *Proceedings of the 7th Meeting of the ACL Special Interest Group in Computational Phonology (SIGPHON)*, pages 43–51, Barcelona.

Hervé Déjean. 1998. Morphemes as Necessary Concept for Structures Discovery from Untagged Corpora. In D. Powers, editor, *Proceedings of the CoNLL98 Workshop on Paradigms and Grounding in Language Learning*, pages 295–298.

John Goldsmith. 2001. Unsupervised Learning of the Morphology of a Natural Language. *Computational Linguistics*, 27(2):153–198.

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

Zellig Harris. 1955. From phoneme to morpheme. *Language*, 31(2):190–222.

Fiammetta Namer and Pierre Zweigenbaum. 2004. Acquiring meaning for French medical terminology: contribution of morphosemantics. In *Proceedings of Medinfo. 2004*, volume 11, pages 535–539, San Francisco CA.

Sylvain Neuvel and Sean A. Fulop. 2002. Unsupervised Learning of Morphology Without Morphemes. In *Proceedings of the ACL Workshop on Morphological and Phonological Learning 2002*, pages 31–40.

Paul Rayson and Roger Garside. 2000. Comparing corpora using frequency profiling. In *Proceedings of the workshop on Comparing Corpora, held in conjunction with the 38th annual meeting of the Association for Computational Linguistics (ACL 2000)*, pages 1–6, Hong Kong, 1-8 October 2000.

Jenny R. Saffran, Elissa L. Newport, and Richard N. Aslin. 1996. Word Segmentation: The Role of Distributional Cues. *Journal of Memory and Language*, 35(4):606–621.

Patrick Schone and Daniel Jurafsky. 2001. Knowledge-Free Induction of Inflectional Morphologies. In *Proceedings of the Second meeting of the North American Chapter of the Association for Computational Linguistics*, pages 1–9.

Stefan Schulz, Martin Honeck, and Udo Hahn. 2002. Biomedical Text Retrieval in Languages with a Complex Morphology. In *Proceedings of the ACL Workshop on Natural Language Processing in the Biomedical Domain*, pages 61–68, Philadelphia, July.

Jacques Vergne. 2005. Une méthode indépendante des langues pour indexer les documents de l'internet par extraction de termes de structure contrôlée. In *Actes de la Conférence Internationale sur le Document Électronique (CIDE 8)*, Beyrouth, Liban.

George Kingsley Zipf. 1968. *The Psycho-biology of Language. An Introduction to Dynamic Philology.* The M.I.T. Press, Cambridge, second paperback printing (first edition: 1935) edition.

Pierre Zweigenbaum and Natalia Grabar. 2000. Liens morphologiques et structuration de terminologie. In *Actes de IC 2000 : Ingénierie des Connaissances*, pages 325–334.

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 information 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 algorithm 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.

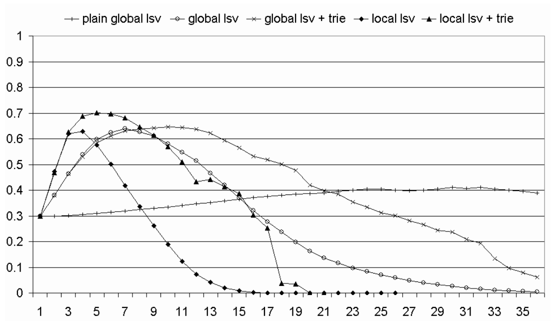


Figure 1: Comparison of global LSV vs. local LSV and after application of the trie-based classificator for a variety of thresholds. Baseline is plain global LSV without normalisations.

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.

A Simpler, Intuitive Approach to Morpheme Induction

Samarth Keshava
Yale University
New Haven, CT 06520
samarth.keshava@yale.edu

Emily Pitler
Yale University
New Haven, CT 06520
emily.pitler@yale.edu

Abstract

We present a simple, psychologically plausible algorithm to perform unsupervised learning of morphemes. The algorithm is most suited to Indo-European languages with a concatenative morphology, and in particular English. We will describe the two approaches that work together to detect morphemes: 1) finding words that appear as substrings of other words, and 2) detecting changes in transitional probabilities. This algorithm yields particularly good results given its simplicity and conciseness: evaluated on a set of 532 human-segmented English words, the 252-line program achieved an F-score of 80.92% (Precision: 82.84% Recall: 79.10%).

1 Introduction

This paper addresses the problem of segmenting natural language words into morphemes, the smallest units of language that still contain meaning. While one cannot extract meanings from lists of words and their frequencies, we can nevertheless use statistical information to make useful predictions about likely morphemes.

There is a large body of literature on morpheme induction, and while it is impossible to give a complete survey, see (Goldsmith, 2001) for a good summary of previous approaches. Goldsmith divides these past attempts into four categories: identification of morpheme boundaries using transitional probabilities; identification of morpheme-internal bigrams or trigrams; discovery of relationships between pairs of words; and an information-theoretic approach to minimize the number of letters in the morphemes of the language. Our work combines ideas from several of these approaches and does not fit neatly into any one of the categories.

The key idea in this paper is to use words that appear as substrings of other words and transitional probabilities together to detect morpheme boundaries. The first approach derives from the observation that the stem left over after removing prefixes and suffixes is often a legitimate word. Though, due to spelling changes, this is not always the case and therefore this method should not be used to actually segment a word. Given a large enough corpus, however, the most common morphemes can be found in this way. The other idea, using transitional probabilities, was initially presented by (Harris, 1955). Given an utterance, Harris proposed finding how many other utterances in the corpus shared each starting fragment of that utterance. He hypothesized that peaks in these counts correspond to morpheme boundaries.

(Hafer and Weiss, 1974) further developed the ideas presented in Harris's paper. Using Harris's transitional probability technique as a starting point, Hafer and Weiss created 15 different algorithms that achieved various levels of precision and recall. One issue with their approach is its heavy reliance on empirically determined parameters. For example, their best algorithm (with a precision of 91.0% and a recall of 61.0%) posited a morpheme boundary if the suffix is a word and the predecessor count is at least 5, or if the predecessor count is at least 17 and the successor count is at least 2.

Our goal was to design a simple algorithm based on our intuition that simpler algorithms are more likely to approximate human processes. We consciously limited both the number of language-specific assumptions that our program makes and "magic numbers"—parameters arbitrarily tuned to make the program work. We did not limit the length of morphemes, the number of morphemes per word, or the total number of morphemes.

2 Methodology

Our algorithm has four basic steps. We

1. build trees with probabilities based on the corpus,
2. score word fragments using these trees to obtain a large list of morphemes,
3. prune this list of morphemes, and
4. segment the test words using the morpheme list and the lexicographic trees.

Each of these steps is described in further detail below.

2.1 Building the Lexicographic Trees

At the beginning of the algorithm, we create two trees of letters and their associated counts: the "forward tree" and "backward tree". We explain here the construction of the "forward tree" (the other construction is symmetric). Suppose the alphabet of the language has b letters, and the longest word in the corpus consists of d letters. Then conceptually, we construct a complete b-way tree with depth d. At each node, each of the b branches represents one of the letters in the language. Thus, any path from the root to some node spells out the starting fragment of some word(s), and the node itself contains the frequency of that string. (Note that in practice, actually creating such a tree would be prohibitive as well as wasteful since most letter combinations never occur; thus we actually only store nodes with non-zero counts.)

The forward and backward trees allow us to calculate conditional probabilities in O(1) time given a starting or ending substring of a word. For example, we would use the forward tree to calculate $Pr_f(\text{s}|\text{report})$ (by dividing the frequency of words starting with "reports" by the frequency of words starting with "report"). In the opposite direction, we would use the backward tree to calculate $Pr_b(\text{e}|\text{ports})$ (by dividing the frequency of words ending in "eports" by the frequency of words ending in "ports").

2.2 Scoring Potential Morphemes

Once we have finished constructing the trees as described above, we begin finding morphemes. We maintain two lists of morphemes: a prefix list and a suffix list.[1] To populate the suffix list, for each word, we scan from the end of the word and consider every possible suffix in order of increasing length. Suppose we are considering the suffix $B\beta$ in the word $\alpha AB\beta$. We hypothesize the proposed suffix is correct if

1. αA is also a word in the corpus,
2. $Pr_f(A|\alpha) \approx 1$, and
3. $Pr_f(B|\alpha A) < 1$.

Similarly, the criteria for determining if αA is a prefix in the word $\alpha AB\beta$ is as follows

1. $B\beta$ is also a word in the corpus,
2. $Pr_b(B|\beta) \approx 1$, and
3. $Pr_b(A|B\beta) < 1$.

The first criterion corresponds to the observation that prefixes and suffixes are often added on to root words. For example, after removing the suffix "ed" from "corresponded", the resulting fragment "correspond" is still a word. The second and third criteria are checked using the forward and backward trees. They check that the stem has multiple children (thus implying other prefixes or suffixes can be joined to the stem) and that the stem's parent has only one child (thus identifying it as a true stem). Using the same example as before, the algorithm would check that $Pr_f(\text{d}|\text{correspon}) \approx 1$, and that $Pr_f(\text{e}|\text{correspond}) < 1$. If a given morpheme passes all three tests, we increase its score by 19 points; otherwise, we decrease its score by 1. After we have iterated through the entire corpus, we consider all strings with positive scores morphemes.

The rule of rewarding word fragments by 19 and punishing by 1 may seem arbitrary, but the constants were chosen so that a string has a positive final score only if it passes our tests at least five percent ($= \frac{1}{1+19}$) of the times it appears. Moreover, the numbers 19 and 1 are not special; any positive numbers x and y such that $\frac{y}{x+y} = .05$ would

[1] We use the terms prefix and suffix loosely, to denote any morpheme generally found at the beginning or end of words. For example, "man" is not technically a suffix, but it is a morpheme that often appears at the end of a word.

produce identical results.[2] The rewarding and punishing scheme is more effective than checking the percentage of tests passed because given two morphemes with the same percentage, the more common morpheme will have a higher score. Thus, the punishing/rewarding scheme takes into account both the reliability and the frequency of the string appearing as a morpheme. Single letters such as ‘t’, which sometimes deceivingly appear to be prefixes, are punished far more often than they are rewarded. Strings such as ‘psycho’, which do not appear often but are almost always true morphemes when they do appear, are rewarded more often than they are punished. Suffixes like ‘s’ are punished occasionally but rewarded very frequently, and are ranked at the top of the list.

2.3 Pruning

Clearly, this method is not perfect. In particular, one problem that often arises is that the final list of morphemes includes strings that are the concatenation of two other morphemes. For example, the list might include all of ‘er’, ‘s’, and ‘ers’. This is undesirable since the final step of segmenting words may process the word “throwers” as throw+ers instead of as throw+er+s. Fortunately, though, this problem has a relatively simple solution which we refer to as “pruning”. We scan each list of morphemes, and if any morpheme is composed of two others with better scores, then it is thrown out.

2.4 Segmenting Words

Finally, we come to the actual segmenting of words. Given the list of morphemes, one possible approach is to simply peel morphemes off the ends of words as they are found. But words such as “politeness” pose a problem: should it be segmented as politenes+s or as polite+ness? Neither the scores nor the lengths of morphemes can be reliably used to answer this question. In this case, ‘s’ would have a higher score, while ‘ness’ is a longer morpheme. They key observation is that the same probability criteria that was used earlier to detect morphemes can be applied here to measure the appropriateness of segmenting at a particular position. In this example, we expect $Pr_f(\text{n}|\text{polite})$ to be lower than $Pr_f(\text{s}|\text{politenes})$ which leads to the correct segmentation.

Table 1: Evaluation results of *RePortS*

Language	Precision	Recall	F-Score
English	82.84 %	79.10 %	**80.92 %**

Thus, our method for segmenting is as follows. First, we scan each word from the end, and find all morphemes $B\beta$ from the suffix list such that our word can be written as $\alpha B\beta$ (for some α). The morpheme with the lowest value of $Pr_f(B|\alpha)$ that is also smaller than 1 is chosen. If such a morpheme is found, it is removed and the processed is repeated until no more morphemes can be removed. We then repeat the same process, attempting to peel off morphemes in the prefix list from the beginning of the word (using Pr_b instead of Pr_f).

3 Results

The algorithm described above was implemented as a Perl program called *RePortS*[3]. The English frequency-word list provided by the Neural Networks Research Centre at the Helsinki University of Technology was combined with a year’s worth of articles from the *Wall Street Journal* and a Linux dictionary file to obtain a corpus containing 185,696 words for training. To determine the performance of the algorithm, we ran our program on a “gold standard” of 532 words (again, provided by the Neural Networks Research Centre) and evaluated our proposed segmentation against the human-determined standard (see Table 1).

Our program identified a total of 1795 morphemes (808 in the prefix list and 987 in the suffix list). Table 2 contains the ten highest-scoring morphemes from each list.

The program was tested on a dual 2.8 GHz processor with 2 GB of memory. We monitored the total running time, i.e. training and segmentation time, and the maximum memory usage of *RePortS*. They are reported in Table 3 for test data of different sizes (note that the same training corpus was used in both cases).

While our algorithm was designed with English and other Indo-European languages in mind, we

[2] Suppose that we rewarded and punished by $x > 0$ and $y > 0$ respectively, satisfying $y/(x + y) = 0.05$. Then $y = 0.05\,(x + y) \Rightarrow 0.05x = 0.95y \Rightarrow x = 19y$. Thus, if a string is rewarded r times and punished p times, it would have a score of $xr - yp = 19yr - yp = y(19r - p)$, which is exactly y times our score. In particular, a string has a positive score if and only if it had a positive score in our algorithm.

[3] The earliest versions of the algorithm determined that the most common prefix, stem and suffix are ‘re’, ‘port’ and ‘s’, respectively; hence, the name *RePortS*.

Table 2: Top English morphemes

Morpheme	Score
un	15858
re	5312
dis	3783
non	2998
over	2717
mis	1812
in	1689
sub	1632
pre	1418
inter	1189

Morpheme	Score
s	24351
ly	18847
ness	10430
ing	8740
ed	5669
al	2655
ism	2169
less	1940
ist	1669
able	1613

Table 3: Resource usage for different test data

# Words	Time	Space
532	0m 27sec	$\sim$ 139 MB
167,377	34m 37sec	$\sim$ 139 MB

decided to test our program with other languages as well. We used test data for Turkish and Finnish provided by the Neural Networks Research Centre. Our results for these two languages are given in Table 4.

4 Discussion

As mentioned earlier, our algorithm performs well given its conciseness and simplicity: the Perl implementation was a total of 252 lines including comments and the algorithm itself can be fully described in four basic steps. Examples of words that our program segments correctly include “repayments” and “passionflowers”. The former contains several affixes and *RePortS* correctly suggests re+pay+ment+s as the segmentation. The latter, on the other hand, is an uncommon compound word segmented as passion+flower+s.

However, the algorithm is obviously not flawless. Consider a word such as “widen” (with a correct segmentation of wid+en). The letters ‘en’ often appear at the end of a word but not as a suffix (e.g. even, ten, hen, etc.) Thus, the potential morpheme ‘en’ is punished far more frequently than it is rewarded, and does not appear in the final list of morphemes. This omission causes us to incorrectly segment words such as “widen” (which our program leaves untouched). However, on the whole, our program performs better with the exclusion of ‘en’.

Table 4: Evaluation results of *RePortS*

Language	Precision	Recall	F-Score
Turkish	72.68 %	43.01 %	**54.04 %**
Finnish	83.76 %	32.30 %	**46.62 %**

Furthermore, there is some evidence that our algorithm is psychologically plausible. As shown in (Saffran et al., 1996a) and (Saffran et al., 1996b), adults as well as infants are able to identify words from continuous speech where the only available cues are transitional probabilities between phonemes. These results show that it is possible for humans to keep track of transitional probabilities and use it in segmentation tasks. However, as (Yang, 2004) points out, transitional probabilities by themselves are not sufficient for larger corpora, and indeed, our algorithm depends on other information as well.

5 Future Work

One notable feature of *RePortS* is that it uses only a list of words and their frequencies. Clearly, contextual information is lost when English text is collapsed into such a list. We feel that the performance could only be improved by extending our algorithm to take advantage of such information. Along the same lines, the inclusion of phonological information may also improve the performane of our algorithm. (Saffran et al., 1996b) showed that humans learned better when presented with both transitional probabilities and prosidic cues than with transitional probabilities alone. Thus, this too is an avenue for improvement.

On a slightly different note, another possibility for further research would be to modify our program to even more closely mirror human learning. More specifically, humans generally do not perform “batch learning”. Therefore, instead of feeding hundreds of thousands of words to the program at once, we could supply the words in smaller chunks, and in the order in which infants would likely hear them. It would be interesting to compare our current results to those from this process.

6 Conclusion

We described an efficient algorithm that uses statistical relationships within and between words to predict morpheme boundaries. Humans are also sensitive to such patterns in natural language.

Moreover, our heuristics make intuitive sense. While we do not claim that humans use our algorithm to segment words, we believe that further research along this line has potential to reveal insight into human language processing.

The program *RePortS* performs quite well against a human-segmented gold standard for English; its precision and recall were both approximately 80%, with an F-Score of 80.92%. Moreover, even though the algorithm was designed for Indo-European languages with a concatenative morphology, it achieved surprisingly decent results for Finnish and Turkish. We experimented with other variants that achieved higher F-Scores, but the algorithm presented here achieved the best balance between performance and elegance.

7 Acknowledgements

We thank Charles Yang for his support throughout the project. It was only through his encouragement and support that we expanded our class project into this work. We would also like to thank Bob Berwick for pointing out the relevance of (Hafer and Weiss, 1974) to our work.

References

John Goldsmith. 2001. Unsupervised Learning of the Morphology of a Natural Language. *Computational Linguistics*, 27(2):153–198.

Margaret A. Hafer and Stephen F. Weiss. 1974. Word Segmentation by Letter Successor Varities. *Information Storage and Retrieval*, 10:371–385.

Z. Harris. 1955. From Phoneme to Morpheme. *Language*, 31(2):190–222.

Jenny R. Saffran, Richard N. Aslin, and Elissa L. Newport. 1996a. Stastical Learning by 8-Month-Old Infants. *Science*, 274(5294):1926–1928.

Jenny R. Saffran, Elissa L. Newport, and Richard N. Aslin. 1996b. Word Segmentation: The Role of Distributional Cues. *Journal of Memory and Language*, 35(4):606–621.

Charles D. Yang. 2004. Universal Grammar, statistics or both? *TRENDS in Cognitive Sciences*, 8(10):451–456.

Morphological learning as principled argument

Lars G Johnsen
Dept. Linguistics and literature
University of Bergen
lars.johnsen@lili.uib.no

Abstract

We develop a morphological learner that evaluates evidence supporting specific claims that a string of letters is a distributional meaningful unit. The distributional evidence is evaluated by selectional properties of morphs, while evidence towards meaning is modelled by looking at the relationship between stems and words. To assess a proposed affix, it gets a probability measure of meaning by comparing all the possible stems the affix occur with to the particular subset that also occur as words. Since for a stem to be a word counts as evidence towards its meaning, the ratio formed by taking stems that are words to the whole set of possible stems for an affix gives a predictive probability measure for the affix that measures the chance that it has combined with a meaningful stem. This measure, taken in conjunction with the selectional statistics of stems and affixes, provides a basis for deciding on the best morphological structure for a given word. The results for English show a combined precision and recall of 45.

1 Introduction

A lexicon for a language will contain, among a lot of facts about the language, a list of words, the morphemes and rules for how to combine different morphemes like stems and affixes into words. It is assumed that the rules are those of concatenative morphology. Given a lexicon with the above properties, the following statement[1] captures the conditions on a word *w* that consists of a stem *x* plus suffix *y*.

$$\begin{array}{c} morph(w, x, y) \\ \Updownarrow \\ stem(x) \wedge suff(y) \wedge sel(x, y) \end{array} \tag{1.1}$$

The binary predicate *sel* encodes the selectional restriction between *x* and *y*. Joining two morphs together may not result in a well formed word, and *sel* encodes the pairs that can be combined together.

Depending on the language, there will be a couple of rules like those in equation (1.1): one for prefix plus stem, stem plus suffix, stem plus infix, and stem plus stem. For the purpose of the morphochallenge task, here restricted to English, we do not consider infixes, nor reduplicative morphology or suprasegmental morphology.

The definition in equation (1.1) splits a word only one time. In order to get a list of morphs from this definition it has to be applied recursively. A predicate *morphs* serves this purpose, and can be defined as follows[2], relating a word form *w* to the list of morphs constituting *w*, in this case a '+' separated list:

$$\begin{array}{c} morphs(w, morphlist) \\ \Updownarrow \\ morph(w, stm, suff) \\ \wedge \\ morphs(stm, stemlist) \\ \wedge \\ morphlist = stemlist + suff \end{array} \tag{1.2}$$

Our point of departure is that the characterization of morphological analysis is the same

[1] Standard notation from predicate logic is used.

[2] The defining expression translates into the Prolog programming language.

whether the lexicon is given or not. The difference between a learner of a lexicon and a knowledgeable performer is viewed as a difference in the level of confidence.

The problem of learning a lexicon from a word list is, according to this view, taken to be the problem of estimating the truth of the terms in equation (1.1). The truth is assessed via probability measures over a set of hypotheses. A simplifying assumption is that each word has a unique split into stem and affix.

Following (Goldsmith, 2001) the number of hypothesized suffixes considered for an English word *w* is limited to six including null morphs. The set of suffix hypotheses for a word like *drinking* is then

$$\begin{aligned} &H = \{ \\ &morph(drinking, drinking, \emptyset), \\ &morph(drinking, drinkin, g) \\ &morph(drinking, drinki, ng) \\ &morph(drinking, drink, ing) \\ &morph(drinking, drin, king) \\ &morph(drinking, dri, nking) \\ &\} \end{aligned} \quad (1.3)$$

The present approach explores ways for calculating the morphological structure using only the distributional properties of stems and suffixes considered as atoms. Their internal letter structure is not taken into account, but see e.g. (Goldsmith, 2001; Creutz & Lagus, 2005) for how one may go about using that kind of information. The information contained in the inherent substring ordering of the morphs is not utilized either.

2 The probability formulation

Equation (1.1) contains the logical statement of the relationship between a stem and an affix conditioned on the facts in the lexicon. This is converted into a probability equation conditioned on the wordlist *W* considered as a set of propositions of what counts as a word.

$$\begin{aligned} &p(morph(w, x, y) \mid W) = \\ &p(stem(x), suff(y), sel(x, y) \mid W) \end{aligned} \quad (1.4)$$

The right hand side can be expanded to a product of the terms in (1.5) and (1.6) below.

$$p(stem(x), suff(y) \mid sel(x, y), W) \quad (1.5)$$

and

$$p(sel(x, y) \mid W) \quad (1.6)$$

When replacing the lexicon with the wordlist *W* as the conditioning facts in these equations, a couple of assumptions have to be revised. The use of *sel* as a conditioning term in equation (1.5) is assumed to be superfluous. The predicate *sel* is for a learner reinterpreted as continuous measure of selectional information, and as such really is a ternary predicate, relating two possible morphs to their selectional information. By doing this, *sel* contributes to the overall value solely through equation (1.6).

This independence assumption turns (1.5) into (1.7) below

$$p(stem(x), suff(y) \mid W) \quad (1.7)$$

The probability formulation then leaves us to compute the equations (1.6) and (1.7).

We will make one change to the objects in the equations. Instead of working with the morph tokens themselves, they are replaced with their respective distributions, indicated using a * on the morph variable.

A stem *x* corresponds then to the class of possible suffixes it combines with. In the following equations a dot "." is used to indicate concatenation.

$$x \mapsto x^* = \{ z \mid x.z \in W \} \quad (1.8)$$

A suffix *y* corresponds to the class of stems it combines with

$$y \mapsto y^* = \{ z \mid z.y \in W \} \quad (1.9)$$

2.1 Selection

The selectional properties are computed by comparing *W* with the possible combinations from *w=x.y* of stems from *y** and suffixes from *x**, denoted y*.x*. This object is closely related to the paradigm in (Snover & Brent, 2002) and the signatures in (Goldsmith, 2001).

The conditional probability of the two sets *y*.x** and *W* is interpreted in a standard way as being the proportion of successes of their intersection, which is computed as the ratio of good words from *y*.x** to all words in *y*.x**.

$$p(sel(x, y) \mid W) = \frac{| y^*.x^* \cap W |}{| y^*.x^* |} \quad (1.10)$$

The implementation used in the morphochallenge uses a *beta*(*a,b*) density for calculating this equation. The first argument, *a*, of this distribution is filled with the positive cases, the numerator of (1.10), and the second argument, *b*, consists of the number of negative cases, the difference between denominator and numerator. The probability assigned to the selectional property is calculated from this density by taking its mean and subtracting one standard deviation. Subtracting one standard deviation gives a more conservative predictive probability than the taken from (1.10) directly, and will penalize those combinations that contain few examples.

For some stems and affixes, the total number of possible words got rather large, and so for the challenge, there was some experimentation with reducing the parameters for the beta density. Best results for both precision and recall was achieved by shrinking the parameters *a* and *b* by the 6th root.

Future improvements for the computation of *sel,* rest on a Bayesian inversion of the formula (1.10), which can be used in updating a distribution *d* over stems and suffixes via maximum likelihood. A particular *d* is a distribution over the hypotheses for a word as shown in (1.3).

The following equation lets *d* play a role in the computation of *sel* as well, and allows us to take into account various confidence levels as expressed by *d* in particular analyses.

$$p(d \mid sel(x,y),W) = \\ = \frac{p(sel(x,y) \mid W,d) \cdot p(d \mid W)}{p(sel(x,y) \mid W)} \quad (1.11)$$

The denominator and normalizing constant of the right hand side of this equation corresponds to the left side of (1.10) and can by using (1.11) be computed by summing over the relevant distributions *d*

$$p(sel(x,y) \mid W) = \\ = \sum_d p(sel(x,y) \mid W,d)\, p(d \mid W)$$

2.2 Stem and affix

There is no source of meaning for stems and affixes from the word list *W* beyond the assumption that any word itself has a meaning. We exploit this fact in the evaluation of the term (1.7) which we will rewrite slightly. Instead of expanding (1.7) into one term for computing the stem and one for the affix, we make the assumption that any evidence that the possible stem is a stem, also counts as evidence that the putative affix is an affix, and vice versa. Accordingly, the two propositions are combined into one, so that (1.7) becomes

$$p(stemsuff(x,y) \mid W) \quad (1.12)$$

This term gets its value solely from the assessment of meaning as follows. The stem *x* is in the word context *xy* so we ask what the probability is that *x* has any meaning given this contextual information. This is turned into an issue of predictive probability: what is the chance of finding a meaningful string in front of *y*? For example, in English, what is the probability for a token to have meaning in the context

x.ing=[open.ing, str.ing, s.ing, r.ing, laugh.ing, talk.ing]?

Three of the *x's* have an independent distribution on their own, namely [open, laugh, talk], so out of these six, the chance is 50% for anything picked out in front of *ing* is a standalone word and carrying meaning using this measure. Note that the stem itself in stem affix combination is not evaluated directly. The independent distribution of stems is used in classifying the affix which in turn classifies the stem.

A predictive probability measure formulated on the basis of the foregoing discussion is then the ratio of actual words in *y** to *y**.

$$p(stemsuff(x,y) \mid W) = \\ = \frac{\mid y^* \cap W \mid}{\mid y^* \mid} \quad (1.13)$$

As for the case of selection, a beta density is used to localize this ratio. The actual probability assigned takes the standard deviation of this density into account in the same way as for the selection. Affixes with low frequency is penalized by this way of calculating the probability.

A crucial assumption for this approach to work is that the empty suffix is a witness for meaning through the word list. A mild supervision can be built into the learner by supplying other witness morphs that can be used as context for a possible stem. Using the word list as a witness set for meaning presupposes that a good portion of stems are actual words, enough so that different affixes can be distinguished on the basis of it.

With access to a corpus a better model of meaning can be formulated, as shown in (Schone & Jurafsky, 2000).

2.3 Combining the results

The probabilities from each of these estimates are combined for each hypothesis resulting in a total score, and ranking of all the hypotheses. The decision scheme adopted is to select the best hypothesis, i.e. the one with highest probability. An alternative method could be iterative: remove the worst and recalculate the probabilities, and repeat that process until only one hypothesis remains.

Selecting the highest ranked hypothesis results in an F-score of 45%, recall at 54%, and precision at 39% for the English word list, using the tools available for the competition.

3 Conclusion and further work

We have shown how one can use the concept of meaning in evaluating the different candidates for morphological analysis. The method should lend itself to all languages that permit a certain proportion of its stems to occur as words.

The work reported here is in a state of flux and particularly equation (1.11) is explored.

Acknowledgements

I would like to thank Christer Johansson and Kolbjørn Slethei for discussions on the matters presented here, and three anonymous reviewers for their comments on an earlier draft. Part of this work was supported by a grant from the Norwegian research council over the KUNSTI program, project BREDT.

References

Creutz, M. & Lagus, K. (2005). Inducing the Morphological Lexicon of a Natural Language from Unannotated Text. In *Proceedings of the International and Interdisciplinary Conference on Adaptive Knowledge Representation and Reasoning* (pp. 106-113). Espoo.

Goldsmith, J. (2001). Unsupervised learning of the morphology of a natural language. *Computational Linguistics, 27,* 153-198.

Schone, P. & Jurafsky, D. (2000). Knowledge-free induction of morphology using latent semantic analysis. In *CoNLL-2000 and LLL-2000* Lisbon, Portugal.

Snover, M. & Brent, M. (2002). A Probabilistic Model for Learning Concatenative Morphology. In *Proceedings of NIPS 2002*.

Combinatory Hybrid Elementary Analysis of Text

Eric Atwell
School of Computing
University of Leeds
Leeds LS2 9JT, England
eric@comp.leeds.ac.uk

Andrew Roberts
Pearson Longman
Edinburgh Gate
Harlow CM20 2JE, England
andrew.roberts@pearson.com

Abstract

We propose the CHEAT approach to the MorphoChallenge contest: Combinatory Hybrid Elementary Analysis of Text. The idea is: acquire results from a number of other candidate systems; CHEAT will read in the output files of each of the other systems, and then line-by-line select the "majority vote" analysis - the analysis which most systems have gone for. If there is a tie, take the result produced by the system with the highest F-measure; if the other systems' output files are ordered best-first, then this is achieved by simply taking the first of he tied results. To justify our approach, we need to show that this really is unsupervised learning, as defined on the MorphoChallenge website; arguably the CHEAT approach involves super-sized unsupervised learning, as it combines three different layers of unsupervised learning.

1 Our guiding principle: get others to do the work

The reuse of existing components is an established principle in Software Engineering; it is quicker, easier, and overall better to engineer a system using components built by others, than to develop a complex system ourselves. This principle is behind our CHEAT approach to the MorphoChallenge task: to avoid doing work ourselves, we got others to do most of the work, and then copied their results. However, straightforward copying of another entrant's results might be considered unacceptable (perhaps even cheating), so we had to do something a bit smarter. Students generally know that blatant copying of another's work is condemned as plagiarism, and can be detected by text analysis software, eg (Atwell et al 2003); but some students may try to get away with less blatant "smart" copying (Medori et al 2002). We procured results from several candidate systems, and then developed a program to allow "voting" on the analysis of each word: for each word, examine the set of candidate analyses; where all systems were in agreement, the common analysis is copied; but where contributing systems disagree on the analysis, take the "majority vote", the analysis given by most systems. If there is a tie, take the result produced by the system with the highest F-measure; if the other systems' output files are ordered best-first, then this is achieved by simply taking the first of he tied results.

Procuring results from several candidate systems was a challenge by itself, given that entrants were to submit results direct to the MorphoChallenge organizers. These results would not be "on show" until the Workshop, well after the deadline for us to submit our own entry. Our ideal solution was to develop a set of intelligent agents, each of which would learn to develop and submit an entry for the MorphoChallenge contest; we could then use the results of these intelligent agents. However, we did not have sufficient time or AI expertise to build software agents capable of this advanced learning. Fortunately, Eric Atwell had to teach an MSc course in the School of Computing at Leeds University, on Computational Modeling. For assessment, students had to undertake a computational modeling exercise; as the course and the MorphoChallenge contest ran concurrently, this presented the opportunity to set the MorphoChallenge as a student coursework exercise, and require the students to submit their entries to their lecturer (for internal assessment) at the same time as submitting to the organizers.

2 But is this really "unsupervised learning"?

According to the MorphoChallenge website FAQ, "unsupervised learning" means that "*…the program cannot be explicitly given a training file containing "example answers", and nor can example answers be hard-coded into the program.*" We must admit that we originally formulated this definition (to suit our approach) and proposed this to the organizers, who accepted and published it. The presence or absence of "example answers" distinguishes supervised from unsupervised learning: in supervised learning, the system is shown the correct analysis or answer for at least some input words (but not all, otherwise this would not be Machine Learning but dictionary-lookup!) Our CHEAT program is *not* shown definitely correct answers for any word, as it is given not one but several files: although each results file constitutes a set of *candidate/possible* answers, they may not be correct answers, and there is no way of knowing which is correct – the voting system is designed for disagreements between candidates, who cannot all be correct. So, strictly speaking, our CHEAT system **is** an unsupervised learning system.
In fact, there are three cascading layers of unsupervised learning in the overall process, so we call this "Super-sized unsupervised learning":

2.1 Unsupervised learning by autonomous agents: students

Of course, Leeds University MSc students are far more intelligent than any software agent; but they still needed to learn how to tackle the MorphoChallenge task. The Computational Modeling class included students on Cognitive Systems, BioInformatics, GeoInformatics, and Health Informatics programmes, so the students had little or no previous knowledge of morphological analysis or machine learning systems development. Their approach to learning was unsupervised, or at least semi-supervised: Eric Atwell presented lectures on machine learning and linguistic principles underlying morphological analysis, and formulated a coursework specification www.comp.leeds.ac.uk/cmd/assessment.htm and marking scheme involving entry to the contest; but then the students were left to learn for themselves how to develop algorithms and systems. They were **not** explicitly given "example answers" – in this case, example algorithm or code to perform unsupervised learning of morphological analysis. And "example answers" were defnitely **not** hard-coded into the students – in this case, this would mean downloading algorithm or code direct into their brains, something even Leeds University teaching methods can't achieve. The students learning about morphological analysis and machine learning constituted the first phase in the CHEAT cascade: a set of autonomous unsupervised learning agents.

2.2 Unsupervised learning by student programs

The students worked in pairs, each pair designing and implementing a program to perform unsupervised learning of morphological analysis. So, these programs constitute the second phase of the CHEAT cascade: a set of independent unsupervised learning programs, each producing a candidate set of morphological analyses of the contest word-files. Detailed descriptions of the student programs are available in the reports submitted by the students alongside the results files. For our purposes, we treated each student program as a "black box" – all we needed were the results files.

2.3 Unsupervised learning by cheat.py

The third phase in the CHEAT cascade is a simple program to read in the candidate results file, choose the most popular analysis of each word, and output this as the CHEAT result. In the spirit of the CHEAT approach, to avoid doing work by getting others to do it, Eric Atwell tried to avoid having to write this program himself, by asking Andy Roberts to do it – hence our collaboration on this entry. Eric Atwell wrote a basic Python version which worked in theory but not in practice; Andy Roberts supplied a much improved version which coped with the unexpected problems.

3 cheat.py

Python has straightforward yet elegant features for reading, processing and writing text, and "mainstream" syntax similar to Java or C++, so seemed the obvious choice for implementation language. Eric Atwell's first python program is so simple that it should be self-explanatory. The version below reads in 7 candidate results files for the English dataset, ordered by their F-measure scores: hr.txt, cd.txt, b.txt, hz.txt, km.txt. aa.txt. mw.txt. Letters hr, cd etc are initials of the student surnames; b.txt shows one student worked alone.

```
# CHEAT: Combinatory Hybrid
# Elementary Analysis of Text
# Eric Atwell's first PYTHON
# program, 15/01/2006
# first open each result-file,
# open a.txt to write CHEAT result
aa=open('aa.txt','r')
b=open('b.txt','r')
cd=open('cd.txt','r')
hr=open('hr.txt','r')
hz=open('hz.txt','r')
km=open('km.txt','r')
mw=open('mw.txt','r')
a=open('a.txt','w')
# a.txt will be the result file
n=6
# n+1: the no of files to combine
# loop: read each result-file-line
# in array Results[0..n]
# ordered by F-measure score: hr
# was the best, mw was the worst
for Results in
zip(hr,cd,b,hz,km,aa,mw):
# setup array Votes[0..n]
# all values initially 1
  Votes=[1 for x in range(n+1)]
# Votes=[1,1,1,1,1,1] might be
# simpler, but less showoffy...
  for r in range(1,n):
   for t in range(r):
     if Results[r]==Results[t]:
       Votes[t]= Votes[t] + 1
# set Votes[N] to number of copies

# next find the top scoring result
  topscore=1
  topresult=1
  for r in range(n):
   if Votes[r] > topscore:
     topresult=r

# Finally output Results[topresult]
  a.write(Results[topresult])

# after end of loop, close all
# files to terminate cleanly
aa.close()
b.close()
cd.close()
hr.close()
hz.close()
km.close()
mw.close()
a.close()
```

This appeared to work with test samples. However, it assumes the input files are all valid, correctly formatted and containing the analyzed words in the same sequence as the given input. Unexpectedly, this turned out not to be the case with all the student results files. Some of the student programs tried to read in the entire word-file, process and segment words in a program buffer, and then print out the buffer contents in alphabetically sorted order. Unfortunately, the details of sort-ordering are different in some packages or programming languages; in particular, Capital and lower-case letters can be sorted together or separately, and non-alphabetic characters (common in Turkish and Finnish datasets, and found in some loanwords even in the English dataset) may also vary in rank-order. The result was that several student results files did not match the ordering of the input dataset; so the simple cheat.py above was not comparing segmentations of the same words.

4 cheat2.py

Andrew Roberts came to the rescue with a much improved comparison algorithm, which read all the input files into memory, ensured comparisons of “like with like”, then wrote out the majority-vote analysis. Unfortunately the program is too long to include in this paper, but we can assure the reader that it is much more robust, elegant and exception-proof than the first version of cheat.py.

5 Results

We evaluated the final cheat2.py results files using the evaluation.perl program provided by the MorphoChallenge organizers, which compared the results files against small Gold Standard samples of words which we were assured had “correct” segmentation. We then compared the evaluation.perl scores for CHEAT output against the scores for the contributing systems' outputs: 7 systems for English, but only 4 systems managed to cope with the much larger Turkish and Finnish datasets.

```
Evaluation of segmentation
in English results file
against gold standard
segmentation in file
"goldstdsample.eng":
Number of words in gold
standard: 532 (type count)
Number of words in data set:
167377 (type count)
Morpheme boundary detections
statistics:
```

System	F-measure %	Precision %	Recall %
CHEAT	59.19	60.71	57.74
hr	54.89	53.87	55.94
cd	51.83	48.06	56.23
B	49.10	46.90	51.52
hz	38.62	37.55	39.75
km	36.96	33.04	41.95
aa	30.55	23.17	44.83
mw	28.48	22.01	40.35

```
Evaluation of segmentation
in Turkish results file
against gold standard
segmentation in file
"goldstdsample.tur":
Number of words in gold
standard: 774 (type count)
Number of words in data set:
582935 (type count)
Morpheme boundary detections
statistics:
```

System	F-measure %	Precision %	Recall %
CHEAT	56.63	62.05	52.08
cd	55.94	59.39	52.87
hr	44.38	59.46	35.39
B	42.05	54.51	34.23
mw	40.44	37.40	44.02

```
Evaluation of segmentation
in Finnish results file
against gold standard
segmentation in file
"goldstdsample.fin":
Number of words in gold
standard: 660 (type count)
Number of words in data set:
1636336 (type count)
Morpheme boundary detections
statistics:
```

System	F-measure %	Precision %	Recall %
CHEAT	60.26	66.10	55.37
cd	60.18	64.97	56.04
hr	43.46	67.18	32.12
B	38.69	56.95	32.90
mw	28.30	24.18	34.12

We also downloaded the Morfessor system developed by the MorphoChallenge organizers, as advertised on the website (!), and used it to analyse the English, Turkish and Finnish datasets. We then repeated the previous experiments, this time including the Morfessor output as an additional candidate file. We were very surprised to find that the resulting F-measure, Precision and Recall for CHEAT remained unchanged from the values in the tables above – the Morfessor output seemed to have no influence whatsoever on the votes! We then realized that the version of Morfessor freely available via the contest website had apparently been modified so that none of the words from the three Gold Standard samples are included in the evaluation. Thus Morfessor appeared to yield Precision and Recall scores of 0/0 or 100%, but this presumably did not mean other words in the output were all correct.

```
Evaluation of segmentation
in file "m.txt" against
gold standard segmentation
in file "goldstdsample.fin":
Number of words in gold
standard: 660 (type count)
Number of words in data set:
1636336 (type count)
Number of words evaluated: 0
(0.00% of all words in data
set)
Morpheme boundary detections
statistics:
F-measure:  100.00%
Precision:  100.00%
Recall:     100.00%
```

6 Conclusions

For all three languages (English, Turkish, Finnish), our CHEAT system scored a higher F-measure than any of the contributing systems. It also achieved better Precision and Recall scores, with a couple of exceptions: *cd* had a slightly higher Recall for Turkish and Finnish (but not English, and *cd* had a lower Precision and F-measure for all three languages), and *hr* had a higher Precision for Finnish (but lower Precision and F-measure). Combinatory Hybrid Elementary Analysis of Text is a valid approach to Unsupervised Learning of morphological analysis.

We thought we had dreamt up the CHEAT approach as a clever scam to avoid work, get students to do the hard work while letting us come up with a winning system. However, an anonymous reviewer of our draft paper pointed out that the CHEAT approached seemed similar to, or even a copy of, an approach already known in the Machine Learning literature: a committee of unsupervised learners. It transpires that we have inadvertently adopted an unsupervised learning approach to machine learning research: we de veloped the CHEAT algorithm without use of training material such as the background literature, eg (Banko and Brill 2001), adding a fourth layer to the super-sized unsupervised learning model.

Yet another thing we learnt from searching in http://scholar.google.com for research papers on "committee of unsupervised learners" is that "unsupervised learning" is a recognized term in Education research, referring to student learning with minimal explicit direction from teachers, eg (Pursula 2004). It turns out that super-sized unsupervised learning is not only a valid (and hopefully interesting) approach to Machine Learning for the MorphoChallenge task, but also a valid approach to Student Learning. Student feedback suggests that the MSc students relished the challenge of participating in an international research contest, and this inspired many of them to produce outstanding coursework … which made the CHEAT results even better!

Reference

Eric Atwell, Paul Gent, Julia Medori, Clive Souter. 2003. Detecting student copying in a corpus of science laboratory reports. In: Archer, D, Rayson, P, Wilson, A & McEnery, T (editors) *Proceedings of CL2003: International Conference on Corpus Linguistics*, pp. 48-53 Lancaster University.

Michele Banko, Eric Brill. 2001. Scaling to very very large corpora for natural language disambiguation. *Proceedings of the 39th Annual Meeting on Association for Computational Linguistics ACL '01* 26-33.

Julia Medori, Eric Atwell, Paul Gent, Ckive Souter. 2002. Customising a copying-identifier for biomedical science student reports: comparing simple and smart analyses. In: O'Neill, M, Sutcliffe, R, Ryan, C, Eaton, M, & Griffith, N (editors) *Artificial Intelligence and Cognitive Science, Proceedings of AICS02*, pp. 228-233 Springer-Verlag.

Matti Purusla. 2004. An integrated model for lifelong learning. In *Proceedings of the 9th World Conference on Continuing Engineering Education*, Tokyo.

Swordfish: Using Ngrams in an Unsupervised Approach to Morphological Analysis

Chris Jordan[*]
Dalhousie University
6050 University Ave.
Hfx., NS, Canada B3H 1W5
cjordan@cs.dal.ca

John Healy
Dalhousie University
6050 University Ave.
Hfx., NS, Canada B3H 1W5
healy@cs.dal.ca

Vlado Keselj[†]
Dalhousie University
6050 University Ave.
Hfx., NS, Canada B3H 1W5
vlado@cs.dal.ca

Abstract

Morphological analysis refers to the art of separating a word into its base units of meaning, or morphemes. Many popular approaches to this, including Porter's algorithm, have been rule-based. These rule-based algorithms however, generally only perform stemming, the identification of root morphemes, which is only a part of morphological analysis. Such algorithms can only reasonably be applied to languages with a limited number of possible affixes for a given term. Rule based algorithms require a great deal more complexity in order to handle languages with many affixes reliably. We propose Swordfish, an ngram-based unsupervised approach to morphological analysis, as an alternative. An ngram is simply a substring of length n which occurs within a corpus. We take those ngrams with the highest probabilities of occurring within our corpus to be our candidate morphemes. We apply a recursive algorithm, which repeatedly splits a term using a probabilistic-based criterion. The evaluation on the PASCAL dataset shows somewhat better performance on English and worse on Finnish and Turkish word lists than the state-of-the-art system Morfessor, with a significantly lower cost in running time.

1 Introduction

Words in a language are typically a combination of smaller base units of meaning, referred to as morphemes. The act of separating a word into its morphemes is called morphological analysis. For example, the code block below illustrates the morphemes in the word "pretested":

```
pre + test + ed
```

Morphological analysis is an important task, as it allows for the identification of the base units of meaning in a word. Morphemes can be used to identify terms which are semantically similar. This is a common process in document retrieval and has been shown to improve performance of these systems (Kantrowitz et al., 2000). Morphemes are also useful in speech recognition (Siivola et al., 2003) since in many languages the spelling and pronunciation of a word are directly related.

Finding morphemes, unfortunately, is not always simple to do particularly in compounding languages such as Finnish, German, Swedish or Greek; and in highly inflective languages such as Finnish and Hungarian (Hirsimäki et al., 2005). Rule-based stemmers such as Porter's Algorithm (Baeza-Yates and Ribeiro-Neto, 1999) have had some success in identifying and removing affixes in the English language. However, English is not a particularly complex language, at least in terms of affixes. There are typically only one or two prefixes or suffixes possible for most words. A rule-based approach for compounding or highly inflective languages is not particularly effective, due to the sheer number of possible word forms. Similarly, to develop training data for a supervised approach would require an inordinate effort to gain an appropriate level of coverage.

An unsupervised approach to morphological analysis is attractive for highly inflective languages and languages with extensive use of com-

[*] http://www.chrisjordan.ca
[†] http://www.cs.dal.ca/∼ vlado/

pounding. Additionally, an unsupervised approach by definition requires no or minimal linguistic knowledge, which makes it convenient for less common languages and languages with sparse linguistic resources. An ideal unsupervised approach, by definition, should require no training data and very little user input in order to learn the morphemes for a given language. Furthermore, an ideal unsupervised approach should be language-independent and, as such, should be able to extract morphemes from any language, given enough exposure to it. Swordfish is such an approach. The Swordfish algorithm processes a corpus that lists terms and their respective frequencies. A language model is built, using ngram frequencies, and candidate morphemes are identified within words.

2 Previous Work

Morphological analysis and stemming are two similar text processing tasks. Morphological analysis identifies all the morphemes in a word, while stemming finds the root morpheme or stem for a term. Since stems are morphemes themselves, stemming is a subset of morphological analysis. Porter's algorithm (Baeza-Yates and Ribeiro-Neto, 1999) is a popular rule-based approach to stemming words in the English language. There are other rule-based approaches (Kantrowitz et al., 2000; Frakes and Fox, 2003) each differing in the degree to which they will stem a term. The stronger the stemmer, the more a word is altered and the smaller the document index. Retrieval recall also tends to increase with the strength of the stemmer, while precision decreases. While rule-based approaches do encounter some success in the English language, they are less successful when applied to compounding or highly inflective languages (Hirsimäki et al., 2005).

There has been a limited amount of work done in the area of unsupervised morphological analysis. One proposed approach is Morfessor (Hirsimäki et al., 2005; Creutz and Lagus, 2005), which recursively builds a morpheme lexicon. It begins by initializing the set of candidate morphemes, the morpheme lexicon, as the entire vocabulary. Using the frequencies which these morphemes occur in the corpus, Morfessor passes through the entire vocabulary, splitting words into the most likely morphemes. Each word is recursively split into the two most likely substrings until such a split is less probable than the substring being split. The morpheme lexicon is updated with the resulting splits. Passes through the vocabulary are made until the lexicon can no longer be improved. Essentially, the Morfessor algorithm looks for the most likely morphemes by splitting words in the most likely manner. The candidate morpheme lexicon which is produced tends to have good precision though it suffers from low recall. Recall and precision in morphological analysis are standard measures expressed in terms of correctly and incorrectly identified "breakpoints" within a word. These measures will be explained in more details in later sections.

A significant component to the Swordfish algorithm is a suffix array which is used to extract the ngrams. The Yamamoto–Church algorithm (Yamamoto and Church, 2001), modified slightly to handle a list of term frequencies instead of regular text, was employed here due to its ease of implementation and $O(N \log(N))$ run time. In the proposed Swordfish algorithm presented here, ngrams used during the morphological analysis must also be a longest common prefix (LCP) with a length greater than or equal to 1. Hence only those ngrams that occur in multiple terms will be considered as candidate morphemes.

3 Swordfish

The Swordfish algorithm consists of two main phases. The first phase computes the ngram frequencies for our corpus via a modified Yamamoto-Church algorithm (Yamamoto and Church, 2001) that deals with word lists instead of regular text. We include all n-grams, i.e., word substrings, of lengths ranging from 1 to the maximal word length. This phase of our algorithm takes up the majority of run time and memory usage. The resulting set of ngrams are treated as a lexicon of possible morphemes. With this lexicon, it is possible to calculate a probability model for the ngrams using maximum likelihood estimates (MLE) as shown in Equation 1.

$$P(\mathit{ngram}_i) = \frac{\mathit{freq}(\mathit{ngram}_i)}{\sum \mathit{freq}(\mathit{ngram}_n)} \qquad (1)$$

where $P(\mathit{ngram}_i)$ is the probability of ngram_i in the corpus probability model. $\mathit{freq}(\mathit{ngram}_i)$ is the frequency which ngram_i occurs in the corpus and $\sum \mathit{freq}(\mathit{ngram}_n)$ is the total number of ngram occurrences in the corpus.

The second phase uses our ngram frequencies to determine probable splits for dividing words into their base morphemes. There are several plausible methods for dividing a word into its base morphemes. We make the assumption that, since words are built up from morphemes, we expect to see the ngrams representing these morphemes more frequently than we would a random string of characters of equal length.

The steps Swordfish takes to split terms are as follows:

Step 1: Calculate the probability of the current term occurring in the ngram lexicon. This probability is calculated using the MLE. If the term does not occur in the lexicon then it has a probability of 0 or only appears as a substring of a unique larger ngram.

Step 2: Find the two ngrams with the highest probability of forming the term. The probability of two ngrams forming a particular term is considered to be the product of their probabilities in the lexicon. In other words, given a term t we identify the two subterms x' and y' as:

$$(x', y') = \underset{xy=t}{\arg\max}\, P(x)P(y)$$

Step 3: If the probability of the current term is less than the product of the two ngrams in Step 2 then the term is split into its two constituent ngrams. These ngrams are then considered to be terms themselves and Steps 1–3 are repeated.

Step 4: The final set of ngrams that result from Step 3 are considered to be the morphemes for the original term.

The Swordfish algorithm essentially considers all ngrams to be possible morphemes. For a given term, it recursively splits it into two substrings based on the most likely combination of ngrams. The set of ngrams resulting from this splitting process are the suggested morphemes for the original term. The Swordfish algorithm has two strong advantages. First, it is parameter-free and thus requires no tuning on the part of a user; second, it is a purely unsupervised approach, requiring no training data to accomplish its task.

4 Evaluation

The development of Swordfish was prompted by PASCAL's 2005 challenge to facilitate the unsupervised segmentation of words in to morphemes (Morpho Challenge, 2005). For this reason, we use both their corpus and their methodologies to evaluate our algorithm. They use three corpora: one in English; one in Finish; and one in Turkish. The corpora were presented in term frequency lists derived from real world corpora. Currently, Swordfish requires this format in order to run, so any text document would have to be preprocessed into a term frequency list beforehand.

The algorithms for this challenge are evaluated by sampling a gold standard data set that contains a subset of the terms split into their appropriate morphemes. These splits are referred to as surface-level segmentations, or segmentations that contain exactly the same characters as the initial term. Thus where a more traditional morphological analysis might separate 'unsupervised' into 'un+supervise+ed' our ideal separation will be 'un+supervis+ed'.

This evaluation is not weighted by frequency of terms. For example, an error on the term 'the' would be equivalent to an error on the term 'agglutinative'. The metrics used to evaluate performance are precision, recall, and F-measure. In order to compute these values one runs an algorithm across a given term frequency list and divides the terms into morphemes. A random subset of these terms have been tagged and provided to participants as an evaluation set. Of these, a random sample has been selected to evaluate the performance of the various algorithms.

5 Results

Table 1 compares the results for precision, recall and F-measure scores from a baseline run, which shows the results of placing a split between every character, Morfessor, and the Swordfish algorithm. From initial inspection based on the F-measure, we can see that Swordfish outperforms Morfessor on English while Morfessor produces more accurate results on Finnish. The baseline outperforms both approaches on Turkish.

At its heart, this problem can be thought of as a classification problem, with the division between each character in every term being classified as either a morpheme boundary or not a morpheme boundary. This reduces our precision/recall problem to the traditional problem of false positives vs false negatives. A false positive refers to a morpheme boundary being predicted at a location where no morpheme boundary exists while a false negative refers to a morpheme boundary unidentified as such. From this point of view, it is evident that Morfessor has accepted a large number

Language	Algorithm	Precision	Recall	F-measure
English	Baseline	14.56	100.00	25.42
	Morfessor	74.19	26.64	39.20
	Swordfish	55.11	37.45	44.60
Finnish	Baseline	19.71	100.00	32.92
	Morfessor	83.66	29.51	43.63
	Swordfish	70.39	23.58	35.33
Turkish	Baseline	25.87	100.00	41.11
	Morfessor	76.33	24.17	36.71
	Swordfish	58.90	16.79	26.13

Table 1: Comparison of algorithms: precision, recall, F-measure

of false negatives, thus its low recall, in order to minimize its false positives, and thus keep a high level of precision.

Swordfish has similar performance to Morfessor. Both algorithms have a precision that is much higher than its recall, over all languages. Comparing Swordfish with Morfessor in the analysis of English, Swordfish has higher recall and lower precision. In the analysis of Finnish and Turkish, Morfessor has a higher precision and recall. From these preliminary results, it appears that morphemes in English can be more easily extracted using ngrams, while the recursive approach to building a morpheme lexicon used in Morfessor is more effective for Finnish. Neither approach appear to be effective for Turkish though the high precision scores do indicate promise for further research on them.

Two factors not taken into consideration in this evaluation are running times and memory usage of the algorithms. In our initial experiments Morfessor seems to be an order of magnitude slower than Swordfish. Data sets that took a few hours with Swordfish took closer to a day to run with Morfessor.

On the other hand, Swordfish seems to use approximately an order of magnitude more memory than Morfessor. In our experiments we've seen Morfessor use 300MB of memory to process a 20MB file while Swordfish took 3GB to process the same file. Swordfish is currently implemented in Perl and uses some very inefficient hashes to store its suffix arrays and LCP tables.

It should also be noted that these numbers are just observations and more formal benchmarking will be required before any concrete comparisons of memory usage and running time can be made between the two algorithms.

6 Conclusions

In this work, we present Swordfish, a recursive approach to morphological analysis using ngrams. It is purely unsupervised and requires no parameter tuning or supervised training. It constructs an ngram lexicon from which all candidate morphemes are drawn. Morphemes are extracted from terms based on the most probable combination of ngrams.

A major component of the Swordfish algorithm is a Perl module implementing the Yamamoto–Church algorithm used to calculate our ngram frequencies. Currently this algorithm is particularly memory intensive due to heavy use of Perl hashes. There is current development to make this module more memory efficient, which should result in a dramatic decrease in the memory necessary to run the Swordfish algorithm.

Regardless, the Swordfish algorithm is a time-efficient algorithm and results in moderately high precision. Unfortunately, it suffers from low recall. We believe that using a ngram lexicon as the foundation for performing morphological analysis shows a lot potential. Further research should go into how the probabilities of ngrams are compared to the probabilities of terms. The method used here, where the probability of the term is compared to the product of the ngram probabilities is rather simplistic. A better method for comparison may lead to improved recall.

As well, further investigation should be conducted into how the ngram lexicon is constructed. Obviously, not all ngrams are morphemes for a language. In our current approach though, all ngrams are considered possible morphemes. The lexicon might be improved by filtering out ngrams that are determined to not be morphemes. This process can be considered to be "removing noise" from the lexicon and may lead to greater precision.

Both algorithms here were evaluated against randomly selected terms. Unfortunately, there was no evidence to suggest conclusively that our results were not simply an aberration. In order to properly evaluate our algorithm we would like to repeatedly bootstrap our gold standard in order to generate test sets. These test sets could be used to create confidence intervals for our precision, recall and F-measure.

In the results reported here, Swordfish outperforms Morfessor on English but Morfessor is better on Finnish and Turkish. The use of an ngram lexicon, as implemented in Swordfish, has potential as an unsupervised approach to morphological analysis. There is still a great deal of work that could be done to improve the algorithm, especially with regards to the Turkish language. It remains to be seen whether a language-independent approach to morpheme extraction will succeed, or whether the problem will require differing approaches for different families of languages.

7 Acknowledgments

This research was funded by the Natural Sciences and Engineering Research Council of Canada, NSERC and Sun Microsystems.

References

Mark Kantrowitz, Behrang Mohit, and Vibhu Mittal. 2000. Stemming and its effects on TFIDF ranking (poster session). *SIGIR '00: Proceedings of the 23rd annual international ACM SIGIR conference on Research and development in information retrieval*, 357–359.

Vesa Siivola, Teemu Hirsimäki, Mathias Creutz, and Mikko Kurimo. 2003. Unlimited Vocabulary Speech Recognition Based on Morphs Discovered in an Unsupervised Manner. *Proc. Eurospeech'03*, 2293-2296.

Teemu Hirsimäki, Mathias Creutz, Vesa Siivola, Mikko Kurimo, Sami Virpioja, and Janne Pylkkönen. 2005. Unlimited vocabulary speech recognition with morph language models applied to Finnish. *Computer, Speech and Language*.

R. Baeza-Yates and B. Ribeiro-Neto. 1999. *Modern Information Retrieval*. Addison-Wesley.

William B. Frakes and Christopher J. Fox. 2003. Strength and similarity of affix removal stemming algorithms. *SIGIR Forum*, 37(1): 26–30.

Mathias Creutz and Krista Lagus. 2005. Unsupervised Morpheme Segmentation and Morphology Induction from Text Corpora Using Morfessor 1.0. *Technical Report: A81*. Helsinki University of Technology.

Mikio Yamamoto and Kenneth W. Church. 2001. Using suffix arrays to compute term frequency and document frequency for all substrings in a corpus. *Comput. Linguist.*, 27(1): 1–30.

PASCAL Unsupervised Segmentation of Words into Morphemes Challenge 2005. 2006. WWW: http://www.cis.hut.fi/morphochallenge2005/.

Simple Unsupervised Morphology Analysis Algorithm (SUMAA)

Minh Thang Dang
School of Computing
University of Leeds
Leeds LS2 9JT, UK
thangdang@hotmail.co.uk

Saad Choudri
School of Computing
University of Leeds
Leeds LS2 9JT, UK
saad.choudri@gmail.com

Abstract

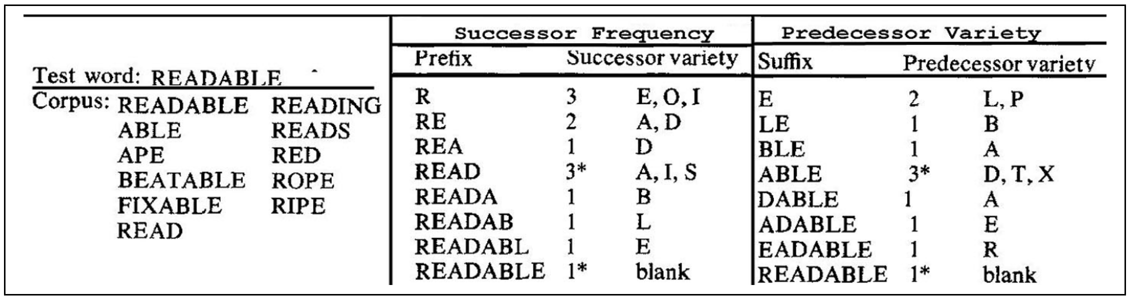
SUMAA is a hybrid algorithm based on letter successor varieties for an entirely unsupervised morphological analysis. Using language pattern and structural recognition it works well on both isolated and agglutinative languages. This paper gives a detailed analysis of how we developed SUMAA. F-Measures (MorphoChallenge, 2005) achieved by SUMAA for the English, Finnish and Turkish datasets were 51.83%, 60.18% and 55.94% respectively.

1 Introduction

Unsupervised automated word segmentation is required for morphological analysis to replace human intervention, with the primary goal to determine the location of breaks between morphemes within words. In this paper we describe an algorithm that segments words into morphemes on an unsupervised basis, i.e. with no prior knowledge (e.g. a dictionary) of the corpus under consideration. The algorithm is applied to both agglutinative languages, where words are composed of fused morphemes denoting their syntactic meanings, and isolating languages where a majority of morphemes are considered to be full fledged words (Wikipedia, 2005). In the provided corpora, Turkish and Finnish represent the former and English represents the latter (Wikipedia, 2005). A generic characteristic between the three is that of varying forms of metaphonics, vowel harmony, unlike e.g. Cantonese, which we think avoids "overfitting" (Mitchell, 1997). The corpora and evaluation script are available on the MorphoChallenge 2005 official web page (http://www.cis.hut.fi/morphochallenge2005/) which compare our results to a gold standard, or "desired" result. Evaluation measures used are Precision, Recall and F-Measure. Precision is a calculation of the number of correct cuts made against the total cuts made, Recall is the total number of correct cuts made against the total possible boundaries and the F-Measure is a harmonic mean of the two (MorphoChallenge, 2005). The F-Measure's we obtained for testing our algorithm on English, Finnish and Turkish were 51.83%, 60.18%and 55.94% respectively. Section 2 explains the "letter successor varieties" approach (Hafer and Weiss, 1974) on which our algorithm is based. Section 3 describes our most relevant experiments leading up to our algorithm. The morpheme boundary statistics are visualised for all our experiments on English in figure 3 and our preferred experiments on all three languages in figure 5. Section 3.4 and 3.5 describe our final algorithm and data structure respectively. The pseudo code is in Table 1 located at the end of section 3.

2 Letter Successor Varieties

Hafer and Weiss (1974) suggested a method called "letter successor varieties" for segmenting lexical text into stems and affixes based on Z.S Harris's solution to the problem of morpheme discovery for phonetic



Test word: READABLE

Corpus: READABLE, ABLE, APE, BEATABLE, FIXABLE, READ, READING, READS, RED, ROPE, RIPE

Successor Frequency			Predecessor Variety		
Prefix	Successor variety		Suffix	Predecessor variety	
R	3	E, O, I	E	2	L, P
RE	2	A, D	LE	1	B
REA	1	D	BLE	1	A
READ	3*	A, I, S	ABLE	3*	D, T, X
READA	1	B	DABLE	1	A
READAB	1	L	ADABLE	1	E
READABL	1	E	EADABLE	1	R
READABLE	1*	blank	READABLE	1*	blank

Figure 1. Letter successor varieties recreated from Hafer and Weiss (1974).

text. The method uses statistical properties, successor and predecessor variety counts, of a corpus to indicate where words should be divided (Hafer and Weiss, 1974).

2.1 Successor varieties and predecessor varieties

The successor frequency (we use the term frequency and variety interchangeably), defined as W [1...n], of the nth letter of a word is the total number of distinct letters occurring at the n+1st position in the words of a corpus that match this set of letters from the selected word. Figure 1 illustrates this, and has been adapted from Hafer and Weiss's paper. In the example, "READABLE" is the "test" word in the corpus consisting of 11 words. If n=1 then the prefix is "R" and comparing "R" to the rest of the words gives a total of 3 distinct letters "E", "O" and "I" occurring at position n+1. Hence, the successor variety is 3. The same is repeated for W [1...2] until n reaches the end of the word. The results are shown. The predecessor variety is a similar concept but with the reverse of the test word, e.g. "ELBADEAR", and the reverse of the corpus (Hafer and Weiss, 1974). This is also shown in figure 1 under the heading of "Predecessor Variety".

2.2 Experimental design

Hafer and Weiss (1974) proposed four basic segmentation strategies that use the statistical method mentioned, viz. cut-off, peak and plateau, complete word and entropy. Our algorithm is based on the peak and plateau design. Take S_n to be the successor count S of the position n in a word W. A cut is made in W after a prefix denoted by the successor count S_n forms a local peak or a plateau of the count vector. The same is applied with predecessor count. In the example discussed in section 2.1, the predecessor count and the successor count both suggest that "READ" and "ABLE" are affixes. Figure 1's 2nd and 3rd columns show the "*" at "3" marking the "peak" of the counts found for both varieties.

3 Our experiments leading up to our proposed algorithm

This section describes only a few of our experiments that have lead to our final solution which is described in section 3.4. Their results are visualised in figure 3.

3.1 Successor frequency at peak/plateau (SF)

By applying only the successor variety counts the results obtained for English were F-Measure 37.95%, Precision 43.00% and Recall 33.97%. This method appeared to split words at the beginning as "peaks" tend to occur there and occasionally ended up with splits such as "cre dit ing" for "crediting" and "cre wmen" for "crewmen".

3.2 Predecessor frequency at peak/plateau (PF)

We then tried the "predecessor frequency at peak/plateau" a reverse of the above method. The results improved to an F-Measure of 41.43%, Precision of 41.67% and Recall of 41.20% for English. This is because the words in the corpus are more heavily suffixed than prefixed. In this case "crediting" was split as "credit ing".

3.3 Successor and predecessor frequency at peak/plateau

We realised that the words not segmented by method 3.1 were segmented correctly by method 3.2. Thus we tried a combination of the two. First, we segmented the words using SF and then for the words that had not been segmented, we segmented them using PF. The improved results for English were now 44.36%, 42.81% and 46.04% for F-Measure, Precision and Recall respectively. We applied this to Turkish and got an F-Measure of 53.24%, Precision of 60.45% and Recall of 47.57%. With Finnish we got an F-Measure of 58.03%, Precision of 63.88% and Recall of 53.16%. Trying a reverse of the combination, i.e. PF first and SF second, did not give good results. For English we got 42.48%, 40.27% and 44.94% for F-Measure, Precision and Recall respectively.

3.4 Our proposed algorithm, SUMAA

So far the first combination of SF and PF (SFPF) has shown the best results. However in the result files, splits like “abandonedly” as “abandon edly” and “acceptances” as “accept ances” were noticed. Analysing the corpora we saw that if one word was a substring of a word below it, it was often a morpheme of that word. We applied this concept to our algorithm. This procedure resembles the bubble sort algorithm except after the comparison the words are not sorted and remain in their original positions. This is illustrated in figure 2. Consider an extraction of the English corpus in the following order: aaa, abandon, abandoned and abandonedly. The steps are as below:

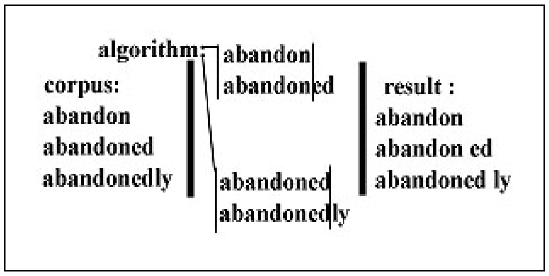


Figure 2. Bubble sort string boundary finder example.

1. Read “aaa”. As it has no preceding word, apply SFPF & segment it .Print to file.
2. Read “abandon”. Check if it has the preceding string “aaa”. It doesn't, so segment it with SFPF and print to file.
3. Read “abandoned”. It contains its preceding word, so first segment it into “abandon” and “ed” and then apply SFPF to the left side split (“abandon”). Print to file.
4. Read “abandonedly”. It contains its preceding word, so segment it into “abandoned” and “ly” and then apply SFPF to the left side split (“abandoned”). In this case, “abandoned” segmented by SFPF resulted in “abandon” and “ed”. Thus the word is finally segmented into “abandon”, “ed” and “ly”. Print to file.

The results were F-Measure 51.83%, Precision 48.06% and Recall 56.23% for English, F-Measure 55.94%, Precision 59.39% and Recall 52.87% for Turkish; and for Finnish they were F-Measure 60.18%, Precision

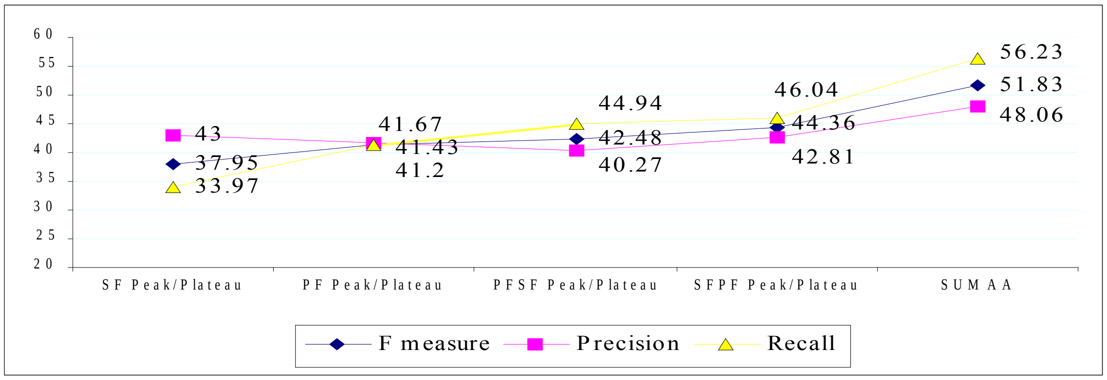


Figure 3. Results, section 3.1, 3.2, 3.3 both combos and 3.4 for tests on English.

64.97% and Recall 56.04%. As the MorphoChallenge requires a high F-Measure, this experiment is very fruitful (shown in figure 5). This new algorithm performed surprisingly well on Turkish and Finnish. It performed better for Finnish than it did for Turkish and not to mention for English! Pseudo code for SUMAA is in table 1. SUMAA seems to have worked particularly well on Finnish with an F-Measure of 60.18% and Precision of 64.97%.

3.5 Data Structure

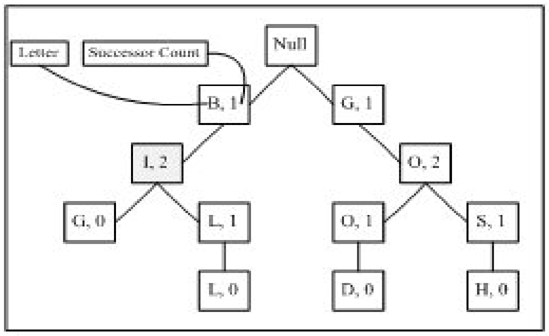


Figure 4. The trie constructed from the words: BIG, BILL, GOOD, GOSH

For letter successor varieties, words of a corpus have been represented as a data structure called a trie (please refer to (Huynh et al.) for more details) as shown in figure 4. Each node represents a letter and contains a successor count. The root represents a null string and each branch represents a word. Consider the following words: “BIG”,” “BILL”, “GOOD”, “GOSH”.

Let’s take “BIG” as the test word. To calculate the successor count of the prefixes of “BIG”, we traverse from the root and then retrieve the successor counts of “B”, “BI” and “BIG” which are 1, 2, and 0 respectively. By organising all words and their reverse words in a corpus in this form, we can efficiently retrieve the SFPF counts of any prefixes or suffixes of a word. In our algorithm we implemented 2 tries, one for retrieving successor counts and one for retrieving predecessor counts. The trie for retrieving predecessor counts was built from the reverse words in the corpora. In doing this, it took less than 3 minutes to segment the whole corpus of Finnish (MorphoChallenge, 2005) which includes 1,636,336 word types on a computer using an AMD 2800+ CPU with 512MB RAM.

4 Conclusion

This paper is a summary of our detailed research which includes experiments with a version of MDL using a codebook, and other versions of the letter successor varieties such as “cut-off”. We abandoned the idea of using the former as it took too much time due to recursive string comparisons and on a standard current day PC, it would have taken 4 days to compute the Finnish corpus as opposed to this algorithm that takes under 3 minutes. The “cut-off” experiment was not used, although it was efficient, as there was a fear that it would cause overfitting problems for Finnish as it used predefined values.

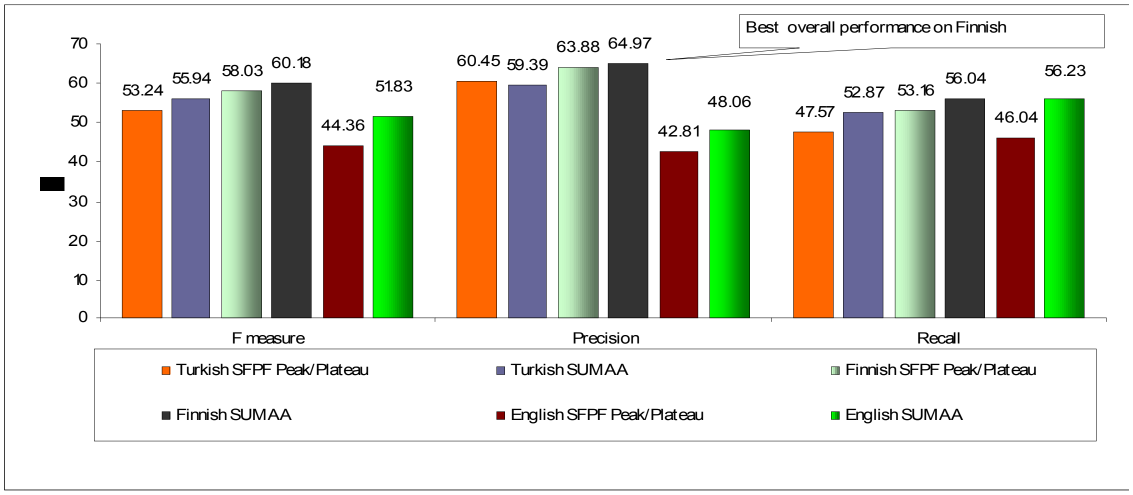


Figure 5. Results for Turkish, Finnish and English with SFPFPeak/Plateau & SUMAA.

```
Main()
{
   BuildTries(); // Build 2 tries,
                 // one from the words as they are and
                 // one from them reversed.
   For each word in the corpus
   {
      If the word contains its preceding word
      {
         Segment the word into 2 parts using the
         boundary of the shorter word;
         SegmentUsingSuccessor(left part);
         Print the right part to file at the end of the
          current line;
      } Else
      {
         SegmentUsingSuccessor(word);
      }
   }
}
SegmentUsingSuccessor(word)
{
    For each substring S of the word
    {
       Calculate the successor count $S_n$;
       If found a local peak/plateau
          Save this position to an array of split
           points;
    }
    If the array of split points empty  // No split point
                                        // found
       //Try to segment the word

       //using Predecessor frequency

       SegmentUsingPredecessor(word);
    Else,
       Use the array of split points to segment the
       word and print to file;
}
SegmentUsingPredecessor(word)
{
    Reverse the word;
    For each substring S of the reverse word
    {
       Calculate the predecessor count $P_n$;
       If found a local peak/plateau
          Save this position to an array of split
           points;
    }
    If the array of split points empty  // No split point
                                        // found
       Print the whole word to file;
    Else,

      Use the array of split points to segment the word
      and print to file;
}
```

Table 1. SUMAA pseudo-code.

It performed well over Turkish but not as well as SUMAA.

SUMAA detected a majority of both true and incorrect boundaries for English with respect to the other languages. This may be because the English corpus contained multi-language words such as “abbotabad” (where abad means population in Urdu), a city in Pakistan. Although it did segment “abbot” and “abad” correctly there were problems with other such noise in the data, which could have lead to the F-Measure and Precision drop. Modelled on English, which is considered to be an isolated language, SUMAA performs better on agglutinative languages and hence our algorithm is robust against overfitting. Figure 5 shows a comparison of SFPF and SUMAA (section 3.3 and 3.4) on all three languages as those obtained the highest results. We propose SUMAA as a very useful and efficient morphological analysis system.

References

David Huynh, David Breton and Evelyne Robidoux. Tries and suffix trees. *Winter 1997 Class Notes.* School of Computer Science, McGill University.

John Elliott and Eric Atwell. 2000. Is anybody out there? The detection of Intelligent and Generic Language-Like Features. *JBIS,* 53:13-22.

John Goldsmith. 2001. Unsupervised learning of the morphology of a natural language. *Computer Linguistics,* 27:153-198.

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

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

MorphoChallenge. 2005. EU Network of Excellence PASCAL Challenge Program. http://www.cis.hut.fi/morphochallenge2005/.

Tom M. Mitchell. 1997. *Machine learning.* New York; London, McGraw-Hill.

Wikipedia. 2005. The Free Encyclopedia. http://www.wikipedia.org/.

Unsupervised Morphemes Segmentation

Khalid ur Rehman
School of Computing
University of Leeds
Leeds, LS2 9JT
scs5kur@leeds.ac.uk

Iftikhar Hussain
School of Computing
University of Leeds
Leeds, LS2 9JT
scs5lh@leeds.ac.uk

Abstract

In this work, we describe the algorithm adopted to split the words into smallest possible meaningful units or morphemes. The algorithm is unsupervised and not dependent on any language. The model is developed using English language. However, the linguistic rules specific to English language are not implemented. The algorithm focuses on the identification of smallest units of words based on their frequency of occurrence in a given text corpus. The model works in two stages. In first stage the model learns from a given text corpus and makes a list of possible morphemes. In the second stage the model divides the words into possible meaningful segments. There is no predefined list of morphemes attached or hardcoded in the model.

1 Introduction

Generally, words are considered as the most basic unit of any language. However, this assumption is not true. In fact, words are the means to communicate in a language and their use vary with time and location. For example, there are words in English language that are spoken in UK but not in USA and vice versa. Similarly, the words in old English poetry are no more in use. Another interesting fact is that the numbers of words in any language are not fixed. According to Dr. Goodword's Separation hypothesis, "*there are no such things as words*" [1]. He claims that "*what we take as words are in fact two distinct phenomena lexemes and morphemes. Lexemes are noun, verb, and adjective ……… Morphemes are everything else, including suffixes like -y, -ness, -er, -ing, -ly and prefixes like re-, un-, anti*"[1]. Lexemes refer to things in real world whereas the morphemes only refer to the grammatical categories.

A morpheme is the smallest meaningful unit in the grammar of a language [1]. There are two basic types of morphemes: roots and affixes [2]. Roots make the main part of the word and repeat only once in a word. On the other hand, affixes are the subordinate parts that may or may not exist in a word. Affixes either precede or follow their root.

2 Assumptions

Following assumptions are made in order to reduce the complexity and improve the efficiency of algorithm.
The special characters (like -, /, ‘ etc.) are treated as the word separators.
The maximum length of a suffix is limited to three characters and maximum length of a root morpheme is limited to 13 characters.
After separating affixes (prefix and suffix) remaining word will be considered as root morpheme and its length must be no less than five characters for further division. As shown in example below.

EXAMPLE:
Actual Word: uneducated
Divided word: un……educate……d
Prefix – un
Root morpheme – educate
Suffix – d

3 Model

The learning model learns from the corpus and prepares a list of possible segments based on their frequency. After the learning is done the words from the corpus are picked up one by one and segmented into possible morphemes.

3.1 Learning Model

This is implemented in Microsoft Access. It takes the most frequent words from the given corpus for identification of possible morphemes. At first, the model extracts the list of words from the corpus ignoring the words having frequency less than seven. The model was also tested by adding lower frequency words. However, it was not affecting the segmentation results but was making the segmentation process slower. Therefore to reduce the complexity the lower limit of frequency was set to seven.

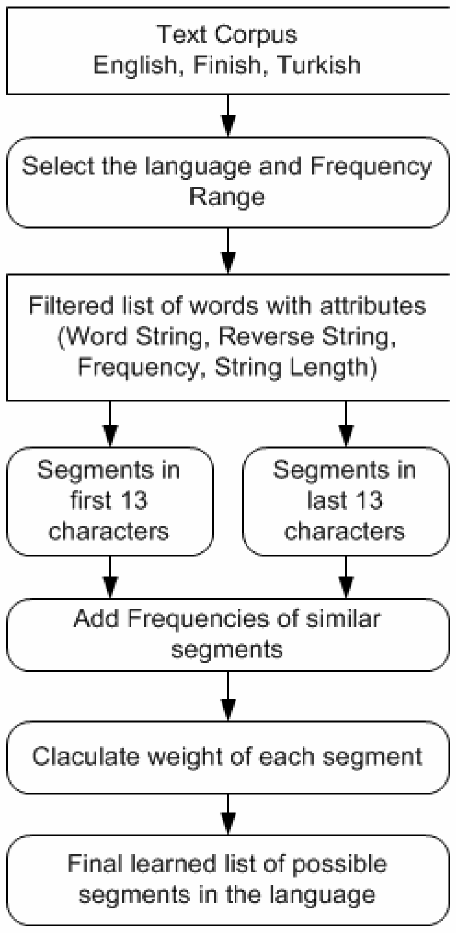

Text Corpus
English, Finish, Turkish
Select the language and Frequency Range
Filtered list of words with attributes (Word String, Reverse String, Frequency, String Length)
Segments in first 13 characters
Segments in last 13 characters
Add Frequencies of similar segments
Claculate weight of each segment
Final learned list of possible segments in the language

Figure 1. Learning Model

Segments (affixes, root morphemes) are searched from within first and last thirteen characters of a word. The maximum length of affixes in English is mostly shorter than four characters however in our model maximum limit is set to thirteen characters. This helps in separating affixes and root morphemes having a maximum length of thirteen characters. The process of finding possible segments in a word works on two sets of thirteen characters (leading 13 characters and trailing 13 characters). See Fig.1.

Before executing the algorithm, the list is sorted using dictionary sorting. Now to understand the algorithm execution let's take an example of six words to be segmented.

- ab
- abacus
- about
- abreast
- again
- bargain

From the above list, model will pick the first character from first word (i.e. 'a') and will check its frequency in the remaining words. The frequency of 'a' is five, now if 'a' also exists as a single word then it will be qualified as a valid segment. However, in the list there is no complete word as 'a' therefore it will be ignored. In the next step model will pick up 'ab' and then check its frequency in the list. The frequency of 'ab' is four. Now as 'ab' exists in the list as a word therefore it will be qualified as a valid segment and will be added to the learned list. This process will continue till maximum thirteen characters of a word (if the length of word is thirteen characters or more).

In order to find segment from trailing side of word, similar procedure will be followed starting from last character.

If a similar segment is found in both processes mentioned above then their frequency will be added and the segment will be included in the learned list only once. See Fig.1.

The last step calculates the weights of different segments. In any corpus single characters have maximum frequencies. Like in our learned list "t, c, g, r and n" occur more than 5000 times however these characters may not be valid segments. Therefore, in order to ignore these characters during segmentation process their weights are calculated by subtracting the standard deviation from their respective frequencies. For example, "t" occurs 5408 times and the standard deviation of frequencies of all single character segments is 3614. This makes the weight of "t" 1794. Similarly frequency of "h" is 2908 but its calculated weight is -705 (2908-3614=-705). As the weight is negative, therefore the segmentation model will not consider "h" as a valid segment.

3.2 Segmentation Model

The segmentation portion of the model is developed in Visual Basic 6.0. The segmentation process pursues the following steps:

- Separation of prefix from the word
- Separation of suffix from the word
- Segmentation of root morpheme (if the word length is more than five character)

This model takes a word from corpus and compares it with the learned list of segments prepared during the execution of learning model.

The segmentation model creates a model list of all words that have been segmented. During the process of segmentation, this list is continuously updated.

As the segmentation model receives a word for segmentation it is broken into parts depending upon the existence of assumed, word separation characters (like -, / , ‘ etc). Both the character strings before and after separation character are treated as independent words. However in this case the word before the separation character is not evaluated for the suffix and the word after the separation character is not evaluated for prefix.

The segmentation is done in two phases. First phase checks each segment of characters starting from the first character till the last character. If any segment of character/s is found in the list and the remaining segment is also found in the list then the separated segment is treated as a possible prefix. At this stage if the segment is of two character length or less its weight is assessed. If the weight is negative, the segment will be disregarded. This process continues until a valid prefix found.

The remaining string (after removing prefix) is passed to the suffix separation module. This module starts from the first trailing character and goes till maximum segmentation of three trailing characters. It could result in more than one suffix. The one with high weight will be considered as valid suffix.

At this stage the remaining root segment is passed to the prefix separation module to separate any possible root morphemes. For root morpheme segmentation a remaining word must have at least five characters. As per the assumption the words of five characters and less are treated as single root morpheme. This assumption is made because till this stage valid prefix and suffix are already separated.

If the prefix separation module fails to segment a word then the word will not be passed to the suffix separation module. It will be passed to the second phase of the model.

The second phase separates first trailing character of the word and then passes the remaining segment to the prefix separation module. Now the prefix separation module repeats the process with one trailing character trimmed. The phase two continues to trim the trailing characters and keep on passing the remaining segment to the prefix separation module till the time the prefix separation module can find any valid segment. The purpose of this module is to pick those words, which cannot be separated by prefix separation module during the first phase. The integration of second phase helps in segmenting the words where two valid segments do not exist in the learned list. For example if we take the word “controlled” the prefix separation module will start from the first character ‘c’ and the first valid morpheme boundary will be after ‘l’, which will make a valid segment “control”. However, there is a possibility that the learned list does not have the remaining segment “led”. Therefore, the programme will not split the word “control led”. Similarly as the word ‘controll’ does not exist in the learned list therefore the other possible segmentation “controll ed” will also not take place. Combination of two phases of model ensures that maximum possible morpheme boundaries are detected.

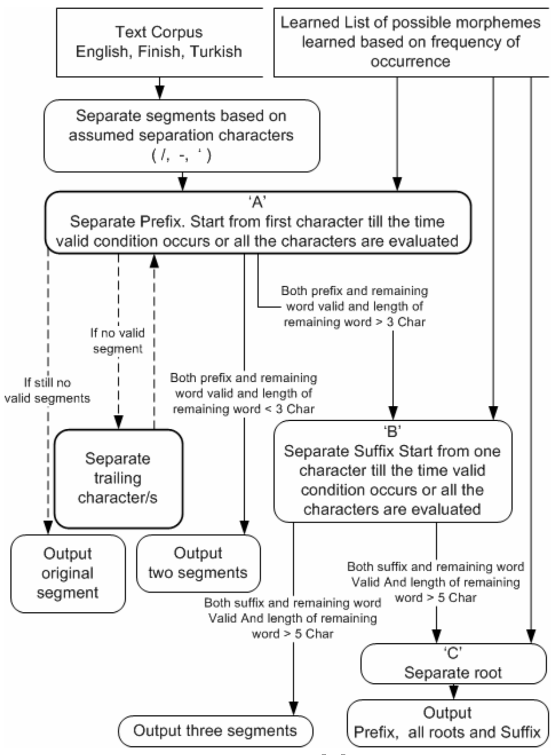

Text Corpus
English, Finish, Turkish
Learned List of possible morphemes learned based on frequency of occurrence
Separate segments based on assumed separation characters (/, -, ‘)
‘A’
Separate Prefix. Start from first character till the time valid condition occurs or all the characters are evaluated
Both prefix and remaining word valid and length of remaining word > 3 Char
If no valid segment
If still no valid segments
Both prefix and remaining word valid and length of remaining word < 3 Char
‘B’
Separate Suffix Start from one character till the time valid condition occurs or all the characters are evaluated
Separate trailing character/s
Output original segment
Output two segments
Both suffix and remaining word Valid And length of remaining word > 5 Char
Both suffix and remaining word Valid And length of remaining word > 5 Char
‘C’
Separate root
Output three segments
Output
Prefix, all roots and Suffix

Figure 2. Segmentation Model

4 Evaluation

The evaluation is done by using the Perl script given on the Morpho Challenge website [3]. The script compares the segmented lists with the gold standard lists given for each language. The evaluation is based on three possibilities mentioned below.

- **Hit.** A valid cut that means word is cut at the right place.
- **Insertion** A wrong cut that means word is cut at the wrong place.
- **Deletion** A missed cut that means a valid cut is ignored.
- Following three parameters are calculated based on these possible cuts.
- **Precision** It is the number of hits divided by the sum of the number of hits and insertions.
- **Recall** It is the number of hits divided by the sum of the number of hits and deletions.
- **F-measure** It is the harmonic mean of precision and recall. As per the gold standard the results having higher value of F-Measure are considered as better segmentation results.

The model was run using English, Turkish and Finnish word lists. The results achieved are as follows:-

Morpheme Boundary Detections Statistics

	English	Turkish	Finnish
F-measure	56.68%	44.38%	43.46%
Precision	53.36%	59.46%	67.18%
Recall	60.46%	35.39%	32.12%

Table 1: Morpheme Boundary Detections Statistics

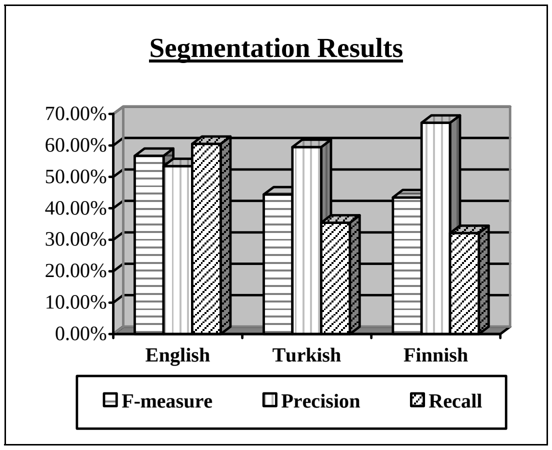


Figure 3. Segmentation Results

- F-measure is maximum in English language. However, it is almost same in Turkish and Finnish.
- Precision is least in English and it increases in Turkish and Finnish.
- Recall is maximum in English and reduces considerably in Turkish and Finnish.

If we plot the line graph of precision and recall then it shows reciprocal behaviors.

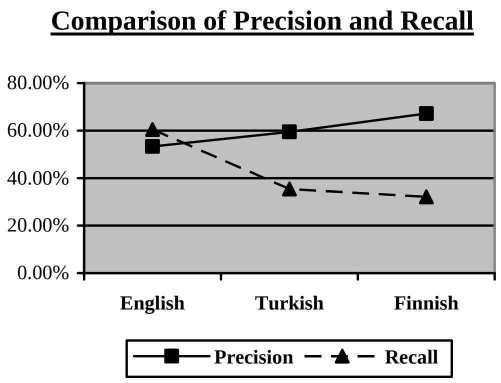


Figure 4. Comparison of Precision and Recall

The proposed model detects the morpheme boundaries based on the frequency of various segments in a given corpus. The results show that possibility of ignoring a valid cut is more than putting a wrong cut in Turkish and Finnish language; however it is opposite in English. As the model always compares both segments for validity, therefore at some occasions a valid morpheme boundary may be ignored. For example if we consider the word ‘stopped’ for segmentation it may not identify any valid segments. Like if ‘stop’ is identified as a valid segment then the remaining ‘ped’ may not be a valid segment. Similarly while the segmentation is done in reverse order ‘ed’ may be recognized as a valid segment, however ‘stopp’ may not be found in the learned list. Under such circumstances the second phase of the model helps to cut the trailing character/s till valid segments are found in the initial set of characters. In this case “stopped” will be segmented as “stop p ed” (here weight for p is 654). This approach helped in avoiding wrong cuts because of which the precision is high in Turkish and Finnish corpus.

The high recall and low precision in English shows that there are less ignored cuts as compared to wrong cuts. The wrong cuts are because our model finds more segments in longer words. Like if we take the example of word “unconstrainedly”. Our learner model has calculated the weight for ‘s’ and ‘t’ as ‘13999’ and ‘1794’ respectively. Therefore the segmentation of the word would be ‘un con s t rained ly’. This has resulted in more wrong cuts in longer words, especially in English language.

The assumptions made at the beginning make the model a bit specific to English. The separation characters helped the model in identifying the morpheme boundaries. The limit on the length of word to be assessed for segmentation of root morpheme, which is set to five, is also based on English language knowledge. As the model is developed by learning from English language corpus, therefore it has resulted in better identification of morpheme boundaries in English language.

The lower **recall** in Turkish and Finnish language has adversely affected the value of F-Measure in these languages. However, the value of **F-measure** for these languages is almost same. This shows that the effect of assumptions on both languages (Turkish and Finnish) is same.

Reference

http://www.alphadictionary.com/articles/drgw004.html, “How many words are in English? – alphaDictionary”, 14-01-06

http://www.ruf.rice.edu/~kemmer/Words/rootaffix.html, “Words in English: Roots and Affixes, 14-01-06

http://www.cis.hut.fi/morphochallenge2005/, “Unsupervised segmentation of Words into morphemes challenge 2005”, 14-01-06

Microsoft Corporation-Online MSDN, http://msdn.microsoft.com/vbasic/, visual Basic Developer Centre, 14-01-06

Mathias Creutz, “Unsupervised Segmentation of Words Using Prior Distribution of Morph Length and Frequency”, Neural Network Research Centre, Helsinki University of Technology, Finland.