Comparing word segmentation algorithms

Judit Ács Géza Velkey
Department of Automation and Applied Informatics
Budapest University of Technology and Economics
judit@aut.bme.hu, evelkey@gmail.com

Abstract We present a comparison of four morphological segmentation algorithms, one of which is Morfessor, an open source unsupervised morphological segmenter, the others are our work. One is based on the simple compression method, byte-pair encoding, one is based on the observation that ngrams at morpheme boundaries have a different distribution from global ngram distributions. The last method uses neural networks for segmentation. All methods are trained and tested on Hungarian web corpora.

1 Introduction

Morphology is the study of word structure, how words are formed and how they are related to each other. Traditionally linguists define *morpheme* as the smallest unit that has semantic meaning on its own. Morphological analysis is the process of segmenting words into smaller units, called *morphemes* and analyzing their syntactic and semantic roles.

Morphemes are either *bound* or *free*. Bound morphemes may only appear as parts of a larger word, while free or unbound morphemes stand on their own. Bound morphemes are further categorized into *derivational* and *inflectional* morphemes. The former category consist of morphemes that change the grammatical category or the meaning of the word, while the latter category only affects plurality, tense, case and so on.

Morphological analysis plays a central role in many natural language processing tasks such as information retrieval, machine translation or spell checking. However, the large variation of morphological paradigms across different languages makes it especially hard to create general solutions for morphological analysis. For example Chinese or Vietnamese are both *isolating languages* i.e. they almost exclusively use free morphemes, therefore analysis below word level is neither hard nor necessary. In contrast, Uralic languages such as Hungarian exhibit rich morphology, and the number of grammatical unique word forms is very large. English – by far the most well resourced language – lies somewhere in the middle, therefore most general solutions do not focus as much on morphology as for example Hungarian NLP would necessitate.

Rule-based morphological analyzers apply an often very large and complicated set of rules that describe the morphology of a natural language and use finite state transducers as inference engines. The creation of these rules may

require years of work from trained linguists. On the other hand, statistical analyzers train in a supervised fashion thus they require labeled corpora, which are very limited in size or non-existent for most languages. Another problem is that morphological rules are very much language-dependent with little applicability outside a language family. [1] showed that the multilingual learning of morphology works better when the languages in question are related. The difficulty of creating morphological analyzers and the relative lack of interest in morphology from the English-speaking community resulted in a paucity of freely available analyzers.

Although labeled treebanks are scarce, unlabeled corpora is readily available in large quantities on the Internet, resulting in a steady interest in the unsupervised learning of morphology. Most work tackles the first task of morphological analysis, morpheme segmentation. In the case of inflectional morphology, the word is composed of morphemes that are concatenated after each other and morphological segmentation is the task of recovering morpheme boundaries inside words.

The paper is organized as follows: Section 2 gives a brief overview of related work, Section 3 describes the four algorithms we compared, data and preprocessing steps are detailed Section 4. Section 5 presents the results and Section 6 concludes the paper.

2 Related Work

Unsupervised morphological segmentation has been studied since the 1950s [2, 3] with [4] being the first to employ probabilistic notions, namely the conditional entropy of words at different positions in the word. [5] gives a thorough review of the field. More recently, *Morfessor* [6] has been the go-to solution for unsupervised segmentation, and it remains the most popular openly available unsupervised segmenter. Morpho Challenge [7] is an online competition where competitors have to solve several morphological problems, including unsupervised segmentation. Morpho Challenge was organized 5 times between 2005 and 2010 and hasn't been organized since. It included tasks in four languages: English, Finnish, German and Turkish. Morfessor-based algorithms outperformed all other solutions in all languages except German.

In 2016 another morphology-related shared task was organized, called the SIGMORPHON 2016 Shared Task, which among 15 others included Hungarian data [8]. The task was supervised morphological reinflection. An example of English reinflection is the conversion of *ran* to its present participle, *running*. The winning team outperformed all other teams in all languages and all subtasks with a relatively simple Bidirectional LSTM model [9].

3 Algorithms

We compare four algorithms: Morfessor, character pair, entropy-based and neural network word segmentation. Although strictly speaking, only the last group

can be classified as a machine learning method, and the others as statistical methods, we use the term supervision to describe the degree which they employ the gold standard segmentation. We also separate a training and a testing or segmentation phase for each method.

3.1 Morfessor

Morfessor is an unsupervised morphological segmenter introduced by [6], which was later extended to semi-supervised algorithms in [10]. Morfessor has several implementation, and we used the Morfessor 2.0 Python package [11]. The system trains on a list of unsegmented words and repeatedly segments the corpus into morpheme-like segments. The goodness of the segmentation is measured by Minimum Description Length. To our knowledge, Morfessor is the state-of-the-art unsupervised segmenter for the Finnish language.

3.2 Character pair segmentation

Character pair segmentation is based on byte pair encoding[12], which is a simple data compression algorithm that repeatedly replaces the most common pair of consecutive bytes and replaces them with a byte that does not occur within that data. A table of replacements is required to rebuild the original data. Our character pair segmentation method finds the most frequent N bigrams and replaces them with N characters that do not appear in the original data. This process is repeated I times, each iteration using the previous iteration's output as its input. This allows the method to replace longer sequences than two.

```
iteration 1: abcdefabc -> AcdefAc
iteration 2: AcdefAc -> BdefB
```

After this process, we decode the replacement table until all replacements contain a non-terminal on their left side and two or more terminals on their right side:

```
before decoding:
    A -> ab
    B -> Ac
after decoding:
    B -> abc
```

We treat the right hand side of these rules as morphs and sort them according to length and frequency. The segmentation phase tries to match each segment starting from the longest ones and if a segment is found in a word, it is split into three parts, then the same function is called recursively on each segment, until the string length 0 is reached.

```
segment(deabcdef) -> segment(de) + " " + segment(abc) +
                     " " + segment(def)
```

3.3 Entropy-based segmentation

This method exploits the linguistic observation that morphemes – as character strings or ngrams – tend to occur in more diverse context than ngrams in general. We quantify this notion by measuring the cross entropy between the neighboring ngram distribution of morphemes. Cross entropy is defined between discrete distributions p and q as:

$$H(p, q) = -\sum_x p(x) \log q(x). \tag{1}$$

Cross entropy cannot be defined if $q(x) = 0$ for any x, where $p(x) > 0$. The distributions in question are ngram frequencies, and thus contain many zero values. We solve this by adding 0.5 to every possible ngram combination's count in both distributions.

In 'training' phase of this method, we computed ngram statistics on the full Webcorpus after the replacement of digraphs. First, we compute word beginning, ending and morpheme beginning and ending unigram, bigram and trigram distribution. Then for every ngram we compute the frequency distributions of its preceding and succeeding ngrams, where n varies from 1 to 3. So for each ngram, we get 6 frequency distributions: its preceding unigrams, bigrams and trigrams and its succeeding unigrams, bigrams and trigrams. Then, we compute the cross entropy of each distribution with word and morpheme beginning and ending ngram distributions. The result is a table where each row corresponds to an ngram. For these experiments, we only used the succeeding ngram distributions. An example of the relevant fields is depicted in Table 1.

Table 1: An example of the relevant fields stored for the ngram *meg*

P	Q	**Cross entropy**
succeeding unigram	morpheme begin unigram	0.4994
succeeding bigram	morpheme begin bigram	1.7627
succeeding trigram	morpheme begin trigram	3.0214

Using a predefined threshold, we select all ngrams from the table if their succeeding ngram distribution's cross entropy with morpheme begin distribution is lower than the threshold, i.e. the two distributions are similar. The selected ngrams will be the segmentation boundaries, so in every word, we segment after an ngram if it appears in the table. Each segmentation experiment has three fixed parameters:

$N_{\textbf{left}}$ the size of ngrams to split after (in the example in Table 1 this would be 3). 1, 2 and 3 were tested.

$N_{\textbf{right}}$ the size of ngrams which the cross entropy was computed on,

threshold cross entropy threshold.

A character boundary is classified as a morpheme boundary if its preceding ngram's right ngram distribution is sufficiently similar to the morpheme begin ngram distribution.

3.4 Neural network segmentation

Our last method uses a neural network with a single hidden layer (more hidden layers actually decrease the performance). The network is trained to identify morpheme boundaries in unsegmented words, and morpheme boundaries are denoted with the plus symbol in target words. Each character of the word is mapped to a V dimensional one-hot vector, where V is the size of the alphabet, then the vectors are concatenated into a single NV dimensional vector, where N is the length limit of words. We set N to twenty, which includes more than 95% of unique word types. Words shorter than 20 letters are padded with spaces from their beginning. After preprocessing the size of the alphabet, V is 48, and thus a single word vector is 960 dimensional.

Segmentation is currently limited to words with exactly one morpheme boundary. We generate every possible segmentation as candidates, for example, the word `színtér` has the following candidates:

```
s+zíntér
sz+íntér
szí+ntér
szín+tér
színt+ér
színté+r
```

and encode them using the aforementioned one-hot coding. Then we propagate the unsegmented word through the network, and compare it to the candidate vectors using Euclidean distance. The closest candidate is selected as segmentation.

4 Data and preprocessing

The input word samples were obtained from MNSZ2[1] [13] and the Hungarian Webcorpus [14]. For supervised settings, we used the `e-magyar` system [15] to segment words into morphemes.

```
leváltása       le vált ás a
előadásokon     elő ad ás ok on
szárazföldön    szárazföld ön
hasonlatomat    hasonlat om at
színtér         szín tér
```

[1] Hungarian Gigaword Corpus [Magyar Nemzeti Szövegtár]

Since our input data is a web crawl and thus quite noisy, we discard words that contain characters outside the Hungarian alphabet, digits, dashes and full stops. We also discard words that do not contain at least one letter from the Hungarian alphabet and we lowercase all text. The corpora contain a few English words that we consider noise in this experiment, so we extract the 10,000 most frequent words from UMBC WebBase [16] and filter English words from the Hungarian frequency list.

Hungarian orthography employs digraphs – combinations of two letters representing one sound – for certain phonemes which can safely be converted into a single letter. We perform this by replacing digraphs with single letters that otherwise do not appear in the alphabet such as uppercase letters as everything had already been lowercased. This method introduces a small number of false positives on compound and morpheme boundaries such as *község*, where *zs* are in fact two separate phonemes and we replace them with a single *Z*. An example of the corpora after preprocessing is illustrated below.

hatással	hat ás sal
felSínre	felSín re
rohanva	rohan va
rajongójaként	rajongó ja ként
réSleteket	réSlet ek et
boSSantó	boSSant ó

Since our goal is morphological segmentation, we disregard word frequency in favor of having more unique word types for training, therefore more morphological phenomena to discover.

5 Results

Morphological segmentation assigns a binary label for each character boundary: is it a segment boundary or not. We use the standard evaluation metrics for binary classification, as well as the word accuracy of each method, which is the ratio of correctly segmented words. Each method is trained on as many word types as we could fit into memory and the individual numbers are listed at each experiment.

5.1 Morfessor

Morfessor was trained using its default parameters, we only varied the amount of training data it received. Table 2 presents the results for varying training sizes.

5.2 Character pair segmentation

Character pair segmentation has two parameters: N, the number of frequent bigrams to collect in each iteration and I, the number of iterations to run. The

Table 2: Morfessor results for different training sizes

train	precision	recall	F-score	word_accuracy
100	0.2207	0.9536	0.3584	0.0070
1000	0.3487	0.8683	0.4976	0.0973
10000	0.5157	0.7201	0.6010	0.2292
100000	0.6290	0.6349	0.6320	0.2552
200000	0.6372	0.6035	0.6199	0.2432
400000	0.6358	0.5754	0.6041	0.2314
500000	0.6410	0.5632	0.5996	0.2432

resulting segment list is at most NI long. We trained on 1,000,000 samples. We tested these parameters in a wide range (see Table 3), running 1000 experiments in total. Table 4 lists the the 5 best configurations, where goodness is measured using the F-score.

Table 3: Character pair segmentation parameter range

Parameter	**Range**
top N	10, 20, 30, 40, 50, 60, 70, 80, 90, 100
iterations	10, 25, 50, 75, 100, 150, 200, 250, 300, 500

Table 4: Character pair segmentation results. 5 best configurations are listed.

iter	topn	precision	recall	F-score	word_accuracy
75	10	0.3795	0.7646	0.5073	0.0822
200	10	0.4125	0.6585	0.5072	0.1151
150	10	0.4010	0.6864	0.5062	0.1041
75	40	0.4271	0.6187	0.5053	0.1115
25	30	0.3750	0.7681	0.5039	0.0599

5.3 Entropy-based segmentation

Entropy-based segmentation has three parameters: $N_{\text{left}}, N_{\text{right}}$ and the cross entropy threshold. Both N_{left} and N_{right} varied from 1 to 3, and the threshold was tested from 0 to 10 using 0.1 as the step size. Statistics were extracted from the full Webcorpus (600M words). Table 5 illustrates the highest scoring threshold values for each N_{left} and N_{right} combinations.

Table 5: Best threshold value and highest F-score for each N_{left} and N_{right} combination

nleft	nright	threshold	precision	recall	F-score	word_accuracy
1	1	1.1	0.3046	0.7404	0.4316	0.2728
1	2	3.8	0.3010	0.7903	0.4359	0.2609
1	3	6.6	0.3234	0.7173	0.4458	0.2903
2	1	1.6	0.3621	0.6852	0.4739	0.3361
2	2	4.2	0.4426	0.7295	0.5509	0.4067
2	3	8.1	0.3777	0.7578	0.5042	0.3454
3	1	2.5	0.2849	0.8032	0.4206	0.2188
3	2	4.2	0.5804	0.6071	0.5934	0.5049
3	3	7.6	0.4186	0.6220	0.5004	0.4220

The highest scoring combination is $N_{\text{left}} = 3$ and $N_{\text{right}} = 2$, but its performance depends heavily on the chosen threshold as Figure 1 attests.

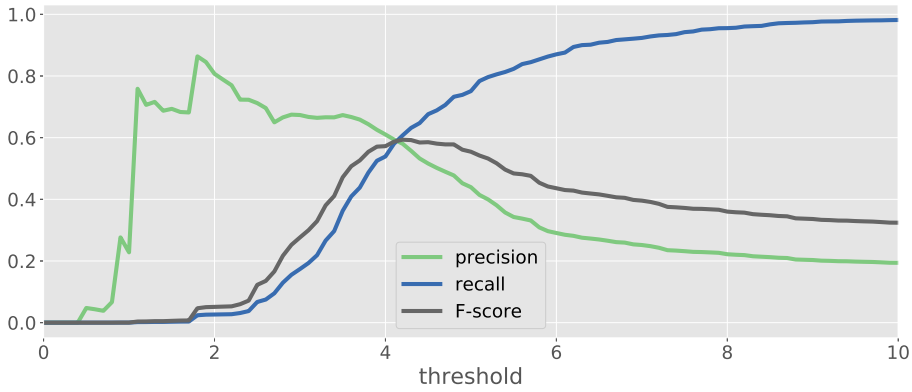

Figure 1: Performance of the $N_{\text{left}} = 3$ and $N_{\text{right}} = 2$ segmenter at different threshold values

5.4 Neural network segmentation

The neural network was trained on 400000 words. The only parameter in this experiment is the number of neurons in the hidden layer. Figure 2 show the F-score for different layer sizes. Since in this experiment, every word has exactly one segment boundary, precision always equals recall, and therefore F-score. The highest F-score, 0.7287, was observed at 550 dimensions, which comes close to what Morfessor achieves (0.7356) on the same dataset.

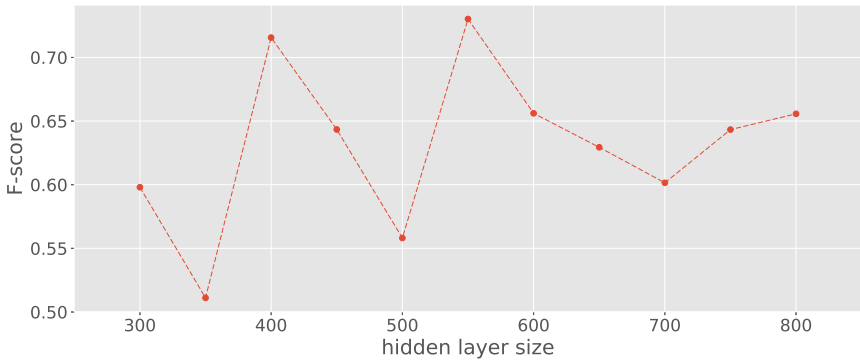

Figure 2: Neural network segmenter performance at different hidden layer sizes

5.5 Summary

All four methods easily achieve F-scores over 0.5, but stay below 0.75, suggesting there is room for improvement. Table 6 summarizes the best results with the amount of supervision used by each method. Although the last method is clearly the winner of the four, it is important to note that currently it is limited to words consisting of two morphemes. Among the general solutions, Morfessor scores the highest, but the entropy-based segmentation comes close too, while using a much simpler algorithm and minimal supervision.

Table 6: Summary of the results

Method	Supervision	F-score
Morfessor	unsupervised	0.6320
Character pair	unsupervised	0.5073
Entropy-based	uses morpheme start ngram distribution	0.5934
Neural	supervised	0.7287

6 Conclusion

We compared four algorithms for the morphological segmentation of natural language. The algorithms vary greatly in complexity and the amount of supervision they use. We tested all algorithms on Hungarian web corpora and found that simple entropy-based solutions potentially rival the state-of-the-art open source segmenter, Morfessor. The last group used neural networks and the results are very promising. We would like to extend this solution to words with more than two morphemes as well as try other network architectures such as sequence-to-sequence neural networks.

Hungarian inflectional morphology exhibits regular assimilation at the boundaries of morphemes which results in frequent allomorphs, i.e. different surface forms of the same underlying abstract morpheme. In the future we plan to cover assimilation effects and allomorphy.

Acknowledgement

Research supported by Hungarian Scientific Research Found (OTKA), contract number 120145.

References

[1] B. Snyder and R. Barzilay, "Unsupervised multilingual learning for morphological segmentation," in *Proceedings of ACL-08: HLT*, (Columbus, Ohio), pp. 737–745, 2008.

[2] Z. S. Harris, "From phoneme to morpheme," *Language*, vol. 31, no. 2, pp. 190–222, 1955.

[3] Z. S. Harris, "Morpheme boundaries within words: Report on a computer test," in *Papers in Structural and Transformational Linguistics*, pp. 68–77, Springer, 1970.

[4] M. A. Hafer and S. F. Weiss, "Word segmentation by letter successor varieties," *Information storage and retrieval*, vol. 10, no. 11-12, pp. 371–385, 1974.

[5] J. A. Goldsmith, "Unsupervised learning of the morphology of a natural language," *Computational Linguistics*, vol. 27, no. 2, pp. 153–198, 2001.

[6] M. Creutz and K. Lagus, "Unsupervised discovery of morphemes," in *Proc. 6th SIGPHON*, pp. 21—30, 2002.

[7] M. Kurimo, S. Virpioja, and V. T. Turunen, "Proceedings of the morpho challenge 2010 workshop," in *Proceedings of the Morpho Challenge 2010 Workshop*, (Espoo, Finland), Aalto University School of Science and Technology, 2010.

[8] R. Cotterell, C. Kirov, J. Sylak-Glassman, D. Yarowsky, J. Eisner, and M. Hulden, "The sigmorphon 2016 shared task—morphological reinflection," in *Proceedings of the 2016 Meeting of SIGMORPHON*, (Berlin, Germany), Association for Computational Linguistics, August 2016.

[9] K. Kann and H. Schütze, "Med: The lmu system for the sigmorphon 2016 shared task on morphological reinflection," *ACL 2016*, p. 62, 2016.

[10] O. Kohonen, S. Virpioja, and K. Lagus, "Semi-supervised learning of concatenative morphology," in *Proceedings of the 11th Meeting of the ACL Special Interest Group on Computational Morphology and Phonology*, pp. 78–86, Association for Computational Linguistics, 2010.

[11] S. Virpioja, P. Smit, S.-A. Grönroos, M. Kurimo, *et al.*, "Morfessor 2.0: Python implementation and extensions for morfessor baseline," 2013.

[12] P. Gage, "A new algorithm for data compression," *The C Users Journal*, vol. 12, no. 2, pp. 23–38, 1994.

[13] Cs. Oravecz, T. Váradi, and B. Sass, "The Hungarian Gigaword Corpus," in *Proceedings of LREC 2014*, 2014.

[14] P. Halácsy, A. Kornai, L. Németh, A. Rung, I. Szakadát, and V. Trón, "Creating open language resources for Hungarian," in *Proceedings of the Fourth International Conference on Language Resources and Evaluation (LREC 2004)*, pp. 203–210, ELRA, 2004.

[15] B. Sass, M. Miháltz, and P. Kundráth, "Az `e-magyar` rendszer gate környezetbe integrált magyar szövegfeldolgozó eszközlánca," in *XIII. Magyar Számítógépes Nyelvészeti Konferencia (MSZNY2017)*, (Szeged), p. (this volume), 2017.

[16] L. Han, A. L. Kashyap, T. Finin, J. Mayfield, and J. Weese, "Umbc_ebiquity-core: Semantic textual similarity systems," in *Second Joint Conference on Lexical and Computational Semantics (*SEM)*, (Atlanta, Georgia, USA), pp. 44–52, Association for Computational Linguistics, 2013.