Unsupervised Discovery of Morphemes

Mathias Creutz and Krista Lagus
Neural Networks Research Centre
Helsinki University of Technology
P.O.Box 9800, FIN-02015 HUT, Finland
{Mathias.Creutz, Krista.Lagus}@hut.fi

Abstract

We present two methods for unsupervised segmentation of words into morpheme-like units. The model utilized is especially suited for languages with a rich morphology, such as Finnish. The first method is based on the Minimum Description Length (MDL) principle and works online. In the second method, Maximum Likelihood (ML) optimization is used. The quality of the segmentations is measured using an evaluation method that compares the segmentations produced to an existing morphological analysis. Experiments on both Finnish and English corpora show that the presented methods perform well compared to a current state-of-the-art system.

1 Introduction

According to linguistic theory, morphemes are considered to be the smallest meaning-bearing elements of language, and they can be defined in a language-independent manner. However, no adequate language-independent definition of the word as a unit has been agreed upon (Karlsson, 1998, p. 83). If effective methods can be devised for the unsupervised discovery of morphemes, they could aid the formulation of a linguistic theory of morphology for a new language.

It seems that even approximative automated morphological analysis would be beneficial for many natural language applications dealing with large vocabularies. For example, in text retrieval it is customary to preprocess texts by returning words to their base forms, especially for morphologically rich languages.

Moreover, in large vocabulary speech recognition, predictive models of language are typically used for selecting the most plausible words suggested by an acoustic speech recognizer (see, e.g., Bellegarda, 2000). Consider, for example the estimation of the standard $n$-gram model, which entails the estimation of the probabilities of all sequences of $n$ words. When the vocabulary is very large, say 100 000 words, the basic problems in the estimation of the language model are: (1) If words are used as basic representational units in the language model, the number of basic units is very high and the estimated word $n$-grams are poor due to sparse data. (2) Due to the high number of possible word forms, many perfectly valid word forms will not be observed at all in the training data, even in large amounts of text. These problems are particularly severe for languages with rich morphology, such as Finnish and Turkish. For example, in Finnish, a single verb may appear in thousands of different forms (Karlsson, 1987).

The utilization of morphemes as basic representational units in a statistical language model instead of words seems a promising course. Even a rough morphological segmentation could then be sufficient. On the other hand, the construction of a comprehensive morphological analyzer for a language based on linguistic theory requires a considerable amount of work by experts. This is both slow and expensive and therefore not applicable to all languages.

The problem is further compounded as languages evolve, new words appear and grammatical changes take place. Consequently, it is important to develop methods that are able to discover a morphology for a language based on unsupervised analysis of large amounts of data.

As the morphology discovery from untagged corpora is a computationally hard problem, in practice one must make some assumptions about the structure of words. The appropriate specific assumptions are somewhat language-dependent. For example, for English it may be useful to assume that words consist of a stem, often followed by a suffix and possibly preceded by a prefix. By contrast, a Finnish word typically consists of a stem followed by multiple suffixes. In addition, compound words are common, containing an alternation of stems and suffixes, e.g., the word kahvinjuojallekin (Engl. ‘also for [the] coffee drinker’; cf. Table 1). Moreover, one may ask, whether a morphologically complex word exhibits some hierarchical structure, or whether it is merely a flat concatenation of stems and suffixes.

1.1 Previous Work on Unsupervised Segmentation

Many existing morphology discovery algorithms concentrate on identifying prefixes, suffixes and stems, i.e., assume a rather simple inflectional morphology.

Déjean (1998) concentrates on the problem of finding the list of frequent affixes for a language rather than attempting to produce a morphological analysis of each word. Following the work of Zellig Harris he identifies possible morpheme boundaries by looking at the number of possible letters following a given sequence of letters, and then utilizes frequency limits for accepting morphemes.

<table>
<thead>
<tr>
<th>Word</th>
<th>kahvinjuojallekin</th>
</tr>
</thead>
<tbody>
<tr>
<td>Morps</td>
<td>kahv</td>
</tr>
<tr>
<td>Transl.</td>
<td>coffee of drink -er for also</td>
</tr>
</tbody>
</table>

The problem is further complicated as languages evolve, new words appear and grammatical changes take place. Consequently, it is important to develop methods that are able to discover a morphology for a language based on unsupervised analysis of large amounts of data.

As the morphology discovery from untagged corpora is a computationally hard problem, in practice one must make some assumptions about the structure of words. The appropriate specific assumptions are somewhat language-dependent. For example, for English it may be useful to assume that words consist of a stem, often followed by a suffix and possibly preceded by a prefix. By contrast, a Finnish word typically consists of a stem followed by multiple suffixes. In addition, compound words are common, containing an alternation of stems and suffixes, e.g., the word kahvinjuojallekin (Engl. ‘also for [the] coffee drinker’; cf. Table 1). Moreover, one may ask, whether a morphologically complex word exhibits some hierarchical structure, or whether it is merely a flat concatenation of stems and suffixes.

Table 1: The morphological structure of the Finnish word for ‘also for [the] coffee drinker’.

Goldsmith (2000) concentrates on stem+suffix languages, in particular Indo-European languages, and tries to produce output that would match as closely as possible with the analysis given by a human morphologist. He further assumes that stems form groups that he calls signatures, and each signature shares a set of possible affixes. He applies an MDL criterion for model optimization.

The previously discussed approaches consider only individual words without regard to their contexts, or to their semantic content. In a different approach, Schone and Jurafsky (2000) utilize the context of each term to obtain a semantic representation for it using LSA. The division to morphemes is then accepted only when the stem and stem+suffix are sufficiently similar semantically. Their method is shown to improve on the performance of Goldsmith’s Linguistica on CELEX, a morphologically analyzed English corpus.

In the related field of text segmentation, one can sometimes obtain morphemes. Some of the approaches remove spaces from text and try to identify word boundaries utilizing, e.g., entropy-based measures, as in (Redlich, 1993).

Word induction from natural language text without word boundaries is also studied in (Deligne and Bimbot, 1997; Hua, 2000), where ML-based model optimization measures are used. Viterbi or the forward-backward algorithm (an EM algorithm) is used for improving the segmentation of the corpus.

Also de Marcken (1995; 1996) studies the problem of learning a lexicon, but instead of optimizing the cost of the whole corpus, as in (Redlich, 1993; Hua, 2000), de Marcken starts with sentences. Spaces are included as any other characters.

Utterances are also analyzed in (Kit and Wilks, 1999) where optimal segmentation for an utterance is sought so that the compression effect over the segments is maximal. The compression effect is measured in what the authors call Description Length Gain, defined as the relative reduction in entropy. The Viterbi algorithm is used for searching for the optimal segmentation given a model. The input ut-

\textsuperscript{1}The regular EM procedure only maximizes the likelihood of the data. To follow the MDL approach where model cost is also optimized, Hua includes the model cost as a penalty term on pure ML probabilities.
analyses of the words. Both segmentation methods are applied to the segmentation of both Finnish and English words. In Section 5, we compare the results obtained from our methods to results produced by Goldsmith’s *Linguistica* on the same data.

2 Method 1: Recursive Segmentation and MDL Cost

The task is to find the optimal segmentation of the source text into morphs. One can think of this as constructing a model of the data in which the model consists of a vocabulary of morphs, i.e. the codebook and the data is the sequence of text. We try to find a set of morphs that is concise, and moreover gives a concise representation for the data. This is achieved by utilizing an MDL cost function.

2.1 Model Cost Using MDL

The total cost consists of two parts: the cost of the source text in this model and the cost of the codebook. Let $M$ be the morph codebook (the vocabulary of morph types) and $D = m_1, m_2, \ldots, m_n$ the sequence of morph tokens that makes up the string of words. We then define the total cost $C$ as

$$C = \sum_{toks} \log p(m_t) + \sum_{types} k * l(m_j)$$

The cost of the source text is thus the negative log-likelihood of the morph, summed over all the morph tokens that comprise the source text. The cost of the codebook is simply the length in bits needed to represent each morph separately as a string of characters, summed over the morphs in the codebook. The length in characters of the morph $m_j$ is denoted by $l(m_j)$ and $k$ is the number of bits needed to code a character (we have used a value of 5 since that is sufficient for coding 32 lower-case letters). For $p(m_i)$ we use the ML estimate, i.e., the token count of $m_i$ divided by the total count of morph tokens.

2.2 Search Algorithm

The online search algorithm works by incrementally suggesting changes that could improve the cost function. Each time a new word token is read from the input, different ways of segmenting it into morphs are evaluated, and the one with minimum cost is selected.

Recursive segmentation. The search for the optimal morph segmentation proceeds recursively. First, the word as a whole is considered to be a morph and added to the codebook. Next, every possible split of the word into two parts is evaluated.

The algorithm selects the split (or no split) that yields the minimum total cost. In case of no split, the processing of the word is finished and the next word is read from input. Otherwise, the search for a split is performed recursively on the two segments. The order of splits can be represented as a binary tree for each word, where the leaves represent the morphs making up the word, and the tree structure describes the ordering of the splits.

During model search, an overall hierarchical data structure is used for keeping track of the current segmentation of every word type encountered so far. Let us assume that we have seen seven instances of *linja−auton* (Engl. ‘of [the] bus’) and two instances of *autonkuljetta jallakaan* (Engl. ‘not even by/at/with [the] car driver’). Figure 1 then shows a possible structure used for representing the segmentations of the data. Each chunk is provided with an occurrence count of the chunk in the data set and the split location in this chunk. A zero split location denotes a leaf node, i.e., a morph. The occurrence counts flow down through the hierarchical structure, so that the count of a child always equals the sum of the counts of its parents. The occurrence counts of the leaf nodes are used for computing the relative frequencies of the morphs. To find out the morph sequence that a word consists of, we look up the chunk that is identical to the word, and trace the split indices recursively until we reach the leaves, which are the morphs.

Adding and removing morphs. Adding new morphs to the codebook increases the codebook cost. Consequently, a new word token will tend to be split into morphs already listed in the codebook, which may lead to local optima. To better escape local optima, each time a new word token is encountered, it is resegmented, whether or not this word has been observed before. If the word has been observed (i.e. the corresponding chunk is found in the hierarchical structure), we first remove the chunk and decrease the counts of all its children. Chunks with zero count are removed (remember that removal of leaf nodes corresponds to removal of morphs from the codebook). Next, we increase the count of the observed word chunk by one and re-insert it as an unsplit chunk. Finally, we apply the recursive splitting to the chunk, which may lead to a new, different segmentation of the word.

“Dreaming”. Due to the online learning, as the number of processed words increases, the quality of the set of morphs in the codebook gradually improves. Consequently, words encountered in the beginning of the input data, and not observed since, may have a sub-optimal segmentation in the new model, since at some point more suitable morphs have emerged in the codebook. We have therefore introduced a ‘dreaming’ stage: At regular intervals the system stops reading words from the input, and instead iterates over the words already encountered in random order. These words are resegmented and thus compressed further, if possible.
continues for a limited time or until no considerable decrease in the total cost can be observed. Figure 2 shows the development of the average cost per word as a function of the increasing amount of source text.

1. Initialize segmentation by splitting words into morphs at random intervals, starting from the beginning of the word. The lengths of intervals are sampled from the Poisson distribution with $\lambda = 5.5$. If the interval is larger than the number of letters in the remaining word segment, the splitting ends.

2. Repeat for a number of iterations:
   
   (a) Estimate morph probabilities for the given splitting.
   
   (b) Given the current set of morphs and their probabilities, re-segment the text using the Viterbi algorithm for finding the segmentation with lowest cost for each word.
   
   (c) If not the last iteration: Evaluate the segmentation of a word against rejection criteria. If the proposed segmentation is not accepted, segment this word randomly (as in the Initialization step).

Note that the possibility of introducing a random segmentation at step (c) is the only thing that allows for the addition of new morphs. (In the cost function their cost would be infinite, due to ML probability estimates.) In fact, without this step the algorithm seems to get seriously stuck in suboptimal solutions.

**Rejection criteria.** (1) Rare morphs. Reject the segmentation of a word if the segmentation contains a morph that was used in only one word type in the previous iteration. This is motivated by the fact that extremely rare morphs are often incorrect. (2) Sequences of one-letter morphs. Reject the segmentation if it contains two or more one-letter morphs in a sequence. For instance, accept the segmentation halua + a + n (Engl. 'I want', i.e. present stem of the verb 'to want' followed by the ending for the first person singular), but reject the segmentation halua + a + n (stem of the noun 'desire' followed by a strange sequence of endings). Long sequences of one-letter morphs are usually a sign of a very bad local optimum that may even get worse in future iterations, in case too much probability mass is transferred onto these short morphs $^3$.

### 4 Evaluation Measures

We wish to evaluate the method quantitatively from the following perspectives: (1) correspondence with linguistic morphemes, (2) efficiency of compression of the data, and (3) computational efficiency. The efficiency of compression can be evaluated as the total description length of the corpus and the codebook (the MDL cost function). The computational efficiency of the algorithm can be estimated from the running time and memory consumption of the program. However, the linguistic evaluation is in general not so straightforward.

#### 4.1 Linguistic Evaluation Procedure

If a corpus with marked morpheme boundaries is available, the linguistic evaluation can be computed as the precision and recall of the segmentation. Unfortunately, we did not have such data sets at our disposal, and for Finnish such do not even exist. In addition, it is not always clear exactly where the morpheme boundary should be placed. Several alternatives may be possible, cf. Engl. *hope + d vs. hop + ed* (past tense of *hope*).

Instead, we utilized an existing tool for providing a morphological analysis, although not a segmentation, of words, based on the two-level morphology of Koskenniemi (1983). The analyzer is a finite-state transducer that reads a word form as input and outputs the base form of the word together with grammatical tags. Sample analyses are shown in Figure 3.

The following step consists in retrieving the segmentation of a word if the segmentation contains a morph that was used in only one word type in the previous iteration. We reject the segmentation if it contains two or more one-letter morphs in a sequence. However, these morphemes can be thought of as a group that belongs together: e.g., the Finnish *talo* + *j* (a plural partitive of 'house'); can also be thought of as *talo + ja*.

#### 3 Method 2: Sequential Segmentation and ML Cost

##### 3.1 Model Cost Using ML

In this case, we use as cost function the likelihood of the data, i.e., $P(\text{data} | \text{model})$. Thus, the model cost is not included. This corresponds to Maximum-Likelihood (ML) learning. The cost is then

$$\text{Cost(Source text)} = \sum_{\text{morph tokens}} - \log p(m_i),$$

where the summation is over all morph tokens in the source data. As before, for $p(m_i)$ we use the ML estimate, i.e., the token count of $m_i$ divided by the total count of morph tokens.

##### 3.2 Search Algorithm

In this case, we utilize batch learning where an EM-like (Expectation-Maximization) algorithm is used for optimizing the model. Moreover, splitting is not recursive but proceeds linearly.

![Figure 2: Development of the average word cost when processing newspaper text.](image)

![Figure 3: Morphological analyses for some English and Finnish word forms.](image)

![Figure 4: Alignment of obtained morph sequences with their respective correct morphemic analyses.](image)
analyses of the words. The remaining tags correspond to inflectional affixes (e.g., endings and markers) and clitics. Unfortunately, the parser does not segment the text into words. For each morph and morphemic label, the algorithm compares the morphological analysis of the word to the morphological analysis of the text. The distance between the two is calculated using a formula that takes into account the number of different morphs and morphemic labels used in the text. The formula is as follows:

$$d(M, L) = \log \frac{c_M}{c_L}$$

where $$c_M$$ is the number of word tokens in which morph $$M$$ was used, $$c_L$$ is the number of word tokens in which label $$L$$ was used, $$\log$$ is the logarithm base 2, and $$\frac{c_M}{c_L}$$ is the ratio of the number of occurrences of morph $$M$$ to the number of occurrences of label $$L$$.

The resulting distance is then used to determine the best alignment for the word. The algorithm searches for the alignment that minimizes the total distance for the text. The distance function is defined as the sum of the distances between each word and its corresponding morph and label:

$$d(M, L) = \sum d_i(M_i, L_i)$$

where $$d_i(M_i, L_i)$$ is the distance between the $$i$$th morph and label in the text.

The algorithm then uses the resulting alignments to compute the percentage of correctly segmented words. The percentage of correctly segmented words is calculated as follows:

$$\text{Percentage Correct} = \frac{\text{Number of correctly segmented words}}{\text{Total number of words}} \times 100$$

The results of the inspection for each of the three methods and the two languages are shown in Table 3. We observe different tendencies for English and Finnish. For English, there is a correlation between the morphological segmentation and the linguistic segmentation. The Recursive MDL method performs best in the English corpus on the test set, while the MDL cost function is clearly superior to the Separate MDL function.

5 Experiments and Results

We compared the two proposed methods as well as the Goldsmith's program Linguistica on both English and Finnish corpora. The results were then compared with the output of the Conexor FGD parser. All characters were converted to lower case, and words containing special characters (e.g., ñ, ç, and á) were removed. Other than morphological tags, some other tags were removed from the morphological label pairs. The tagging output of Linguistica was not available as Perl scripts run on Linux.

The tagging output of Linguistica was not available as Perl scripts run on Linux.
Table 2: Test results for the Finnish and English corpus. Method names are abbreviated: Recursive segmentation and MDL cost (Rec. MDL), Sequential segmentation and ML cost (Seq. ML), and Linguistica (Ling.). The total MDL cost measures the compression of the corpus. However, the cost is computed according to Equation (1), which favors the Recursive MDL method. The final number of morphs in the codebook (#morphs in codebook) is a measure of the size of the morph “vocabulary”. The relative codebook cost gives the share of the total MDL cost that goes into coding the codebook. The alignment distance is the total distance computed over the sequence of morph/morphemic label pairs in the test data. The unseen aligned pairs is the percentage of all aligned morph/label pairs in the test set that were not observed in the training set. This gives an indication of the generalization capacity of the method to new word forms.

<table>
<thead>
<tr>
<th>Method</th>
<th>Finnish</th>
<th>English</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Rec. MDL</td>
<td>Seq. ML</td>
</tr>
<tr>
<td>Total MDL cost [bits]</td>
<td>2.09M</td>
<td>2.27M</td>
</tr>
<tr>
<td>#morphs in codebook</td>
<td>6.302</td>
<td>10.977</td>
</tr>
<tr>
<td>Relative codebook cost</td>
<td>10.16%</td>
<td>15.27%</td>
</tr>
<tr>
<td>Alignment distance</td>
<td>768k</td>
<td>81k</td>
</tr>
<tr>
<td>Unseen aligned pairs</td>
<td>23.64%</td>
<td>20.20%</td>
</tr>
<tr>
<td>Time [sec]</td>
<td>620</td>
<td>390</td>
</tr>
</tbody>
</table>

Table 3: Estimate of accuracy of morpheme boundary detection based on visual inspection of a sample of 2500 Finnish word tokens.

<table>
<thead>
<tr>
<th>Method</th>
<th>Correct</th>
<th>Incomplete</th>
<th>Incorrect</th>
</tr>
</thead>
<tbody>
<tr>
<td>Rec. MDL</td>
<td>49.6%</td>
<td>29.7%</td>
<td>20.6%</td>
</tr>
<tr>
<td>Seq. ML</td>
<td>47.3%</td>
<td>15.3%</td>
<td>37.4%</td>
</tr>
<tr>
<td>Linguistica</td>
<td>43.1%</td>
<td>24.1%</td>
<td>32.8%</td>
</tr>
</tbody>
</table>

7 Conclusions

In the experiments the online method with the MDL cost function and recursive splitting appeared most successful especially for Finnish, whereas for English the compared methods were rather equal in performance. This is likely to be partially due to the model structure of the presented methods which is especially suitable for languages such as Finnish. However, there is still room for considerable improvement in the model structure, especially regarding the representation of contextual dependencies.

Considering the two examined model optimization methods, the Recursive MDL method performs consistently somewhat better. Whether this is due to the cost function or the splitting strategy cannot be deduced based on these experiments. In the future, we intend to extend the latter method to utilize an MDL-like cost function.

References


