Unsupervised Morpheme Segmentation and Morphology Induction from Text Corpora Using Morfessor 1.0

Mathias Creutz
Mathias.Creutz@hut.fi

Krista Lagus
Krista.Lagus@hut.fi

Neural Networks Research Centre, Helsinki University of Technology,
P.O.Box 5400, FIN-02015 HUT, Finland

Abstract

In this work, we describe the first public version of the *Morfessor* software, which is a program that takes as input a corpus of unannotated text and produces a segmentation of the word forms observed in the text. The segmentation obtained often resembles a linguistic morpheme segmentation. Morfessor is not language-dependent. The number of segments per word is not restricted to two or three as in some other existing morphology learning models. The current version of the software essentially implements two morpheme segmentation models presented earlier by us (Creutz and Lagus, 2002; Creutz, 2003).

The document contains user's instructions, as well as the mathematical formulation of the model and a description of the search algorithm used. Additionally, a few experiments on Finnish and English text corpora are reported in order to give the user some ideas of how to apply the program to his own data sets and how to evaluate the results.

1 Introduction

In the theory of linguistic morphology, morphemes are considered to be the smallest meaning-bearing elements of language. Any word form can be expressed as a combination of morphemes, as for instance the following English words: 'arrange+ment+s, foot+print, mathematic+ian+'s, un+fail+ing+ly'.

It seems that automated morphological analysis would be beneficial for many natural language applications dealing with large vocabularies, such as speech recognition and machine translation; see e.g., Siivola et al. (2003), Hacioglu et al. (2003), Lee (2004). Many existing applications make use of words as vocabulary base units. However, for highly-inflecting languages, e.g., Finnish, Turkish, and Estonian, this is infeasible, as the number of possible word forms is very high. The same applies (possibly less drastically) to compounding languages, e.g., German, Swedish, and Greek.

There exist morphological analyzers designed by experts for some languages, e.g., based on the two-level morphology methodology of Koskenniemi (1983). However, expert knowledge and labor are expensive. Analyzers must be built separately for each language, and the analyzers must be updated on a continuous basis in order to cope with language change (mainly the emergence of new words and their inflections).

As an alternative to the hand-made systems there exist algorithms that work in an unsupervised manner and autonomously discover morpheme segmentations for the words in unannotated text corpora. *Morfessor* is a general model for the unsupervised induction of a simple morphology from raw text data. Morfessor has been designed to cope with languages having predominantly a concatenative morphology and where the number of morphemes per word can vary much and is not known in advance. This distinguishes Morfessor from resembling models, e.g., Goldsmith (2001), which assume that words consist of one stem possibly followed by a suffix and possibly preceded by a prefix.

Morfessor is a unifying framework for four morphology learning models presented earlier by us. The model called "Recursive MDL" in Creutz and Lagus (2002) has been slightly modified and we now call it the *Morfessor Baseline* model. The follow-up (Creutz, 2003) is called *Morfessor Baseline-Freq-Length*. The more elaborated models (Creutz and Lagus, 2004; Creutz and Lagus, 2005) are called *Morfessor Categories-ML* and *Morfessor Categories-MAP*, respectively.

In this work, we present the publicly available Morfessor program (version 1.0), which segments the word forms in its input into morpheme-like units that we call *morphs*. The Morfessor program only implements the Baseline models, that is the Morfessor Baseline and Morfessor Baseline-Freq-Length model variants. Additionally, the hybrids *Morfessor Baseline-Freq* and *Morfessor Baseline-Length* are supported. Details on how these model variants differ from each other can be found in Section 3 and particularly in Section 3.5.

The Morfessor program is available on the Internet at `http://www.cis.hut.fi/projects/morpho/` and can be used freely under the terms of the GNU General Public License (GPL)[1]. If Morfessor is used in scientific work, please refer to the current document and cite it as follows:

[1]URL: `http://www.gnu.org/licenses/gpl.html`

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, March. URL: `http://www.cis.hut.fi/projects/morpho/`

In case of commercial interest in conflict with the GNU GPL terms, contact the authors.

1.1 Structure of the document

It has been our intention to write a document with a “modular” structure. Readers can focus on the topics that are of interest to them and pay less attention to the rest.

Section 2 is a user’s manual of the software addressed to those interested in applying the program to their own text corpora.

Section 3 describes the mathematics behind the Morfessor Baseline models. The derivation of some formulas are presented separately in Appendices A, B, and C.

The search algorithm for finding the optimal morph segmentation is described in Section 4.

Some experiments are reported in Section 5. These may inspire the users to perform similar experiments on their own data and may give an idea of how to possibly evaluate the result.

The work is concluded in Section 6.

2 User’s instructions for the Morfessor program

Version 1.0 of the Morfessor program can be used for two purposes: (1) learning a model, that is a morph lexicon and morph probabilities, in an unsupervised manner from text and segmenting the words in the text using the learned model (see Section 2.1), (2) using a model learned earlier for segmenting word forms into morphs (see Section 2.2). In the latter case, the word forms to be segmented can be entirely new, that is, they were not observed in the data set that the morph model was trained on.

The Morfessor program is implemented as a Perl script and it consists of one single file. In order to run the program a Perl interpreter must thus be available on the computer. We have tested Morfessor only on a Linux operating system, but cost-free Perl interpreters exist also for Windows and other operating systems.

2.1 Learning a morph segmentation for the words in text

2.1.1 Input data file

The Morfessor program requires that a data file is fed to it using the `-data` option:

```
morfessor1.0.pl -data filename
```

The data file must be a word list with one word per line. The word may be preceded by a word count (frequency) separated from the word by whitespace; otherwise a count of one is assumed. If the same word occurs many times, the counts are accumulated. The following is an excerpt of the contents of a possible data file:

```
...
13 autistic
14 auto
1 autobiographical
1 autobiography
1 autocrats
2 autograph
3 autoloader
1 automate
1 automated
...
```

Note that Morfessor does not perform any automatic character conversion except that it removes superfluous whitespace and carriage return characters. For instance, if you want an upper-case and its corresponding lower-case letter to be treated as the same character *you need to perform the desired conversion and filtering* prior to using the Morfessor program.

Morfessor can be run on different-sized data sets. The number of words in the data can vary from a few thousands of words to millions of words. For more details on possible data set sizes, memory consumption and running time estimates, see Section 5.

2.1.2 Optional parameters

The program can be run without setting any further parameters. However, the following parameters are available for fine-tuning the behavior of the program:

`-finish` *float*	This sets the convergence threshold for the search algorithm. The program stops and outputs the result, if from one training epoch to the next the overall code length (logprob) of the representation of the model and data decreases less than this value multiplied by the number of word types (distinct word forms) in the data (see Section 4). The default value is 0.005 (bits) per word type. To make the program run faster increase this value, but some accuracy may be lost. The value must be within the range: $0 < \mathit{float} < 1$.

`-rand` *`integer`*	This sets the random seed for the non-deterministic search algorithm (see Section 4). The default value is zero.
`-savememory` [*`int`*]	This option can be used for reducing the memory consumption of the program (by approximately 25 %). When using the memory saving option every word in the input data is not guaranteed to be processed in every training epoch. This leads to slower convergence and longer (nearly doubled) processing time. The integer parameter is a value that affects the randomness of the order in which words are processed. High values increase randomness, but may slow down the processing. The default value is 8. If the `-savememory` option is omitted, the memory saving feature is not used, which is the best choice in most situations.
`-gammalendistr` [*`float1`* [*`float2`*]]	A gamma distribution is used for assigning prior probabilities to the lengths of morphs (see Section 3.5). *Float1* corresponds to the most common morph length in the lexicon and *float2* is the β parameter of the gamma pdf (probability density function). The default values for these parameters are 7.0 and 1.0, respectively. If this option is omitted, morphs in the lexicon are terminated with an end-of-morph character, which corresponds to an exponential pdf for morph lengths (for more details consult Section 3.5).
`-zipffreqdistr` [*`float`*]	A pdf derived from Mandelbrot's correction of Zipf's law is used for assigning prior probabilities to the frequencies (number of occurrences) of the morphs. The number (*float*) corresponds to the user's belief of the proportion of *hapax legomena*, that is morphs that occur only once in the morph segmentation of the words in the data. The value must be within the range: $0 <$ *float* < 1. The default value is 0.5. If this option is omitted a (non-informative) morph frequency distribution derived from combinatorics is used instead (consult Section 3.5 for details).

`-trace` *`integer`*	The progress of the processing is reported during the execution of the program. The integer consists of a sum of any or all of the following numbers, where the presence of the number in the sum triggers the corresponding functionality: 1: Flush output, i.e., do not buffer the the output stream, but output it right away. 2: Output progress feedback (how many words processed etc.) 4: Output each word when processed and its segmentation. 8: Trace recursive splitting of morphs. 16: Upon termination output the length distribution of the morphs in the lexicon obtained.

2.1.3 Output

The Morfessor program writes its output to standard output. Some of the lines output by the program are preceded by a number sign (#). These can be considered as "comments" and include, e.g., all information produced using the `-trace` option. The main part of the output consists of the morph segmentations of the words in the input. The format used is exemplified by the following morph segmentations of some words. The words are preceded by their occurrence count (frequency) in the data:

```
...
13 autistic
14 auto
1 auto + biograph + ical
1 auto + biography
1 auto + crat + s
2 autograph
3 autoloader
1 automate
1 automate + d
...
```

The output can be directed to a file using the following syntax:

```
morfessor1.0.pl -data inputfilename > outputfilename
```

If additionally some optional parameters are defined, Morfessor can be invoked, e.g., like this:

```
morfessor1.0.pl -savememory -gammalendistr 10 -trace 19
-data inputfilename > outputfilename
```

2.2 Using an existing model for segmenting new words

Once a morph segmentation model has been learned from some text data set, it can be used for segmenting new word forms. In this segmentation mode of the Morfessor program no model learning takes place. Each input word is segmented into morphs by the Viterbi algorithm, which finds the most likely segmentation of the word into a sequence of morphs that are present in the existing model. (In order to ensure that there is always at least one possible segmentation, every individual character in the word that does not already exist as a morph can be suggested as a morph with a very low probability.)

In order to use an existing model for segmenting words, Morfessor is invoked as follows:

```
morfessor1.0.pl -data filename1 -load filename2.
```

Filename1 is the input word list containing the word forms to be segmented. It has exactly the same format as in Section 2.1.1 above. *Filename2* refers to the segmentation model and is simply the output of an earlier run of the Morfessor program (see Section 2.1.3 above).

No further parameters apply to the segmentation mode of the Morfessor program. The output is written to standard output and is of exactly the same format as in Section 2.1.3.

It may be of interest to some users that repeatedly running the same word list through the Morfessor program while always replacing *filename2* with the most recent output corresponds to performing Expectation-Maximization (EM) using the Viterbi algorithm on the morph segmentations.

3 Mathematical formulation

The formulation of the Morfessor model is presented in a probabilistic framework, inspired by the work of Brent (1999). Since the currently released Morfessor software only implements the Morfessor Baseline models, which have been presented earlier in Creutz and Lagus (2002) and Creutz (2003), the following presentation only contains the mathematical formulation of these models variants. For the later Morfessor Category models the interested reader is referred to Creutz and Lagus (2004) and Creutz and Lagus (2005).

The mathematics of the original publications has undergone some modifications in order to enable us to produce a unifying framework and in order to correct a few inaccuracies. In addition, the Minimum Description Length (MDL) formulation in Creutz and Lagus (2002) has been replaced by a probabilistic *maximum a posteriori* (MAP) formulation in the current work. Conveniently, MDL and MAP are equivalent and produce the same result, as is demonstrated, e.g., by Chen (1996).

3.1 Maximum a posteriori estimate of the overall probability

The task is to induce a model of language in an unsupervised manner from a corpus of raw text. The model of language ($\mathcal{M}$) consists of a morph vocabulary, or a *lexicon of morphs,* and a *grammar*. We aim at finding the optimal model of language for producing a segmentation of the corpus, i.e., a set of morphs that is concise, and moreover gives a concise representation for the corpus. The maximum a posteriori (MAP) estimate for the parameters, which is to be maximized, is:

$$\begin{aligned} \arg\max_{\mathcal{M}} P(\mathcal{M} \,|\, \textit{corpus}) &= \arg\max_{\mathcal{M}} P(\textit{corpus} \,|\, \mathcal{M}) \cdot P(\mathcal{M}), \text{ where} && (1) \\ P(\mathcal{M}) &= P(\textit{lexicon}, \textit{grammar}). && (2) \end{aligned}$$

As can be seen above (Eq. 1), the MAP estimate consists of two parts: the probability of the model of language $P(\mathcal{M})$ and the *maximum likelihood* (ML) estimate of the corpus conditioned on the given model of language, written as $P(\textit{corpus} \,|\, \mathcal{M})$. The probability of the model of language (Eq. 2) is the joint probability of the probability of the induced lexicon and grammar. It incorporates our assumptions of how some features should affect the morphology learning task. This is the *Bayesian* notion of probability, i.e., using probabilities for expressing degrees of prior belief rather than counting relative frequency of occurrence in some empirical test setting.

In the following, we will describe the components of the Morfessor model in greater detail, by studying the representation of the lexicon, grammar and corpus.

3.2 Lexicon

The lexicon contains one entry for each distinct morph (morph type) in the segmented corpus. We use the term “lexicon” to refer to an inventory of whatever information one might want to store regarding a set of morphs, including their interrelations.

Suppose that the lexicon consists of M distinct morphs. The probability of coming up with a particular set of M morphs $\mu_1 \ldots \mu_M$ making up the lexicon can be written as:

$$P(\textit{lexicon}) = M! \cdot P(\textit{properties}(\mu_1), \ldots, \textit{properties}(\mu_M)). \qquad (3)$$

This indicates the joint probability that a set of morphs, each with a particular set of properties, is created. The factor $M!$ is explained by the fact that there are $M!$ possible orderings of a set of M items and the lexicon is the same regardless of the order in which the M morphs emerged. (It is always possible to afterwards rearrange the morphs into an unambiguously defined order, such as alphabetical order.)

In the Baseline versions of Morfessor, the only properties stored for a morph in the lexicon is the *frequency* (number of occurrences) of the morph in the corpus and the *string of letters* that the morph consists of. The representation of the string of letters naturally incorporates knowledge about the *length* of the morph, i.e., the number of letters in the string.

We assume that the frequency and morph string values are independent of each other. Thus, we can write:

$$P(\textit{properties}(\mu_1), \ldots, \textit{properties}(\mu_M)) = P(f_{\mu_1}, \ldots, f_{\mu_M}) \cdot P(s_{\mu_1}, \ldots, s_{\mu_M}), \quad (4)$$

where f represents the morph frequency and s the morph string. Section 3.5 describes the exact pdf:s (probability density functions) used for assigning probabilities to particular morph strings and morph frequency values.

3.3 Grammar

Grammar can be viewed to contain information about how language units can be combined. In the later versions of Morfessor a simple morphotactics (word-internal syntax) is modeled. In the Baseline models, however, there is no context-sensitivity and thus no grammar to speak of. The probability $P(\textit{lexicon}, \textit{grammar})$ in Equation 2 reduces to $P(\textit{lexicon})$. The lack of a grammar implies that a morph is as likely regardless of which morphs precede or follow it or whether the morph is placed in the beginning, middle, or end of a word.

The probability of a morph μ_i is a maximum likelihood estimate. It is the frequency of the morph in the corpus, f_{μ_i}, divided by the total number of morph tokens, N. The value of N equals the sum of the frequency of each of the M morphs types:

$$P(\mu_i) = \frac{f_{\mu_i}}{N} = \frac{f_{\mu_i}}{\sum_{j=1}^{M} f_{\mu_j}}. \quad (5)$$

3.4 Corpus

Every word form in the corpus can be represented as a sequence of some morphs that are present in the lexicon. Usually, there are many possible segmentations of a word. In MAP modeling, the one most probable segmentation is chosen. The probability of the corpus, when a particular model of language (lexicon and non-existent grammar) and morph segmentation is given, takes the form:

$$P(\textit{corpus} \,|\, \mathcal{M}) = \prod_{j=1}^{W} \prod_{k=1}^{n_j} P(\mu_{jk}). \quad (6)$$

Products are taken over the W words in the corpus (token count), which are each split into n_j morphs. The k^{th} morph in the j^{th} word, μ_{jk}, has the probability $P(\mu_{jk})$, which is calculated according to Eq. 5.

3.5 Properties of the morphs in the lexicon

A set of properties is stored for each morph in the lexicon, namely the frequency of the morph in the corpus and the string of letters the morph consists of. Each particular set of properties is assigned a probability according to given probability distributions.

The models presented in Creutz and Lagus (2002) and Creutz (2003) differ in the way they assign probabilities to different morph length and frequency values. Version 1.0 of the Morfessor program supports both previous models, slightly modified. What we here call the *Morfessor Baseline* model is based on Creutz and Lagus (2002) and what we call *Morfessor Baseline-Freq-Length* is based on Creutz (2003). Additionally there are hybrids of the two, which we call *Morfessor Baseline-Freq* and *Morfessor Baseline-Length*.

The Baseline-Freq and Baseline-Freq-Length model variants differ from the other models in that an explicit prior probability distribution is used that affects the frequency distribution of the morphs and that incorporates the user's estimate of the *proportion of hapax legomena*, i.e., morph types that only occur once in the corpus.

The Baseline-Length and Baseline-Freq-Length model variants differ from the other models in that a prior probability distribution is used that affects the length distribution of the morphs and that incorporates the user's estimate of the *most common morph length* in the lexicon.

The pdf:s used for modeling morph frequency and morph length are described below. For the Baseline model, Sections 3.5.1 and 3.5.3 apply. For the Baseline-Freq model, Sections 3.5.2 and 3.5.3 apply. For the Baseline-Length model, Sections 3.5.1 and 3.5.4 apply, and for the Baseline-Freq-Length model, Sections 3.5.2 and 3.5.4 apply.

3.5.1 Frequency modeled implicitly

By implicit modeling of morph frequency we mean that the user does not input his prior belief of a desired morph frequency distribution. In this approach, we use one single probability for an entire morph frequency distribution. The value of $P(f_{\mu_1}, \ldots, f_{\mu_M})$ in Eq. 4 is then:

$$P(f_{\mu_1}, \ldots, f_{\mu_M}) = 1 / \binom{N-1}{M-1} = \frac{(M-1)!(N-M)!}{(N-1)!}, \tag{7}$$

where N is the total number of morph *tokens* in the corpus, which equals the sum of the frequencies of the M morph *types* that make up the lexicon. The derivation of the formula can be found in Appendix A. This probability distribution corresponds to a *non-informative prior* in the sense that only the total number of morph tokens and types matter, not the individual morph frequencies.

3.5.2 Frequency modeled explicitly

By explicit modeling of morph frequency we mean that a probability distribution is used that assigns a particular probability to every possible morph frequency value. Here we assume that the frequency of one morph is independent of the frequencies of

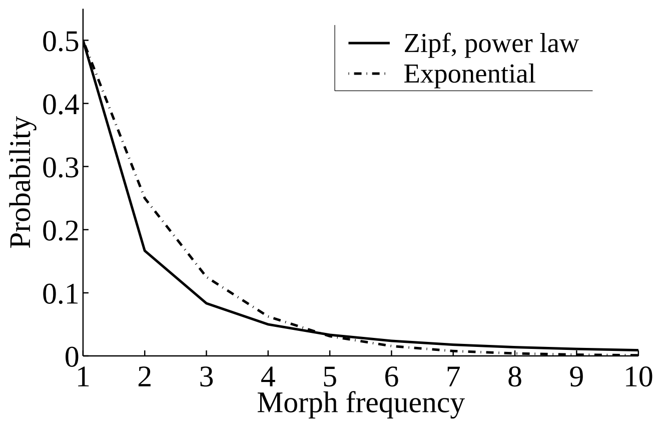


Figure 1. Probabilities of morph frequencies according to (1) a pdf derived from Mandelbrot's correction of Zipf's law ($h = 0.5$) and (2) an approximately exponential pdf resulting from applying the non-informative frequency prior ($N = 10\,000, M = 5000$).

the other morphs. Thus,

$$P(f_{\mu_1}, \ldots, f_{\mu_M}) = \prod_{i=1}^{M} P(f_{\mu_i}). \tag{8}$$

An expression for $P(f_{\mu_i})$ is derived in (Creutz, 2003) and it is based on Mandelbrot's correction of Zipf's law. However, that derivation is unnecessarily complicated and incomplete. A better derivation is given in Appendix B and the result is:

$$P(f_{\mu_i}) = f_{\mu_i}^{\log_2(1-h)} - (f_{\mu_i} + 1)^{\log_2(1-h)}. \tag{9}$$

The parameter h represent the user's prior belief of the proportion of hapax legomena, i.e., morph types that occur only once in the corpus. Typically, the proportion of hapax legomena is about half of all morph types.

In order to use this type of frequency prior in the Morfessor program, the `-zipffreqprior` switch must be activated. Optionally, a value for h can be entered. If omitted, a h value of 0.5 will be used by default. However, the difference obtained when using this Zipfian prior instead of the noninformative prior in Section 3.5.1 is small.

Figure 1 illustrates the probability distributions obtained using the two approaches. The Zipfian prior corresponds to a *power law curve*. The non-informative prior approximately results in an *exponential distribution* for the probability of the frequency of an individual morph (see derivation in Appendix C). The curves are different, but not radically different for small frequency values, which may explain why neither approach performs significantly better than the other.

3.5.3 Length modeled implicitly

We make the simplifying assumption that the string that a morph consists of is independent of the strings that the other morphs consist of. Therefore, the joint probability $P(s_{\mu_1}, \ldots, s_{\mu_M})$ in Eq. 4 reduces to the product of the probability of each individual morph string:

$$P(s_{\mu_1}, \ldots, s_{\mu_M}) = \prod_{i=1}^{M} P(s_{\mu_i}). \tag{10}$$

We simplify further and assume that the letters in a morph string are drawn from a probability distribution independently of each other. Thus, the probability of the string of μ_i is the product of the probability of the individual letters, where c_{ij} represents the j^{th} letter of the morph μ_i:

$$P(s_{\mu_i}) = \prod_{j=1}^{l_{\mu_i}} P(c_{ij}). \tag{11}$$

The probability distribution over the alphabet $P(c_{ij})$ is estimated from the corpus by computing relative frequencies of each of the letters observed. The length of the string is represented by l_{μ_i}.

Now, implicit modeling of morph length implies that there is a special *end-of-morph* character that is part of the alphabet and is appended to each morph string in the lexicon and marks the end of the string. The probability that a morph of a particular length l will emerge in this scheme is:

$$P(l) = [1 - P(\#)]^l \cdot P(\#), \tag{12}$$

where $P(\#)$ is the probability of the end-of-morph marker. The probability is the result of first choosing l letters other than the end-of-morph marker and finally the end-of-morph marker. This is an *exponential distribution*, that is, the probability of observing a morph of a particular length decreases exponentially with the length of the morph.

3.5.4 Length modeled explicitly

Instead of using an end-of-morph marker for the morphs in the lexicon, one can first decide the length of the morph according to an appropriate probability distribution and then choose the selected number of letters according to Eq. 11.

As a probability distribution for modeling morph length, a Poisson distribution could be applied. Poisson distributions have been used for modeling word length, when word *tokens* (words of running text) have been concerned, e.g., by Nagata (1997). However, here we model the length distribution of morph *types* (the morphs in the lexicon) and we have chosen to use a *gamma distribution*. This produces the following formula for the prior probability of the morph length l_{μ_i}:

$$P(l_{\mu_i}) = \frac{1}{\Gamma(\alpha)\beta^{\alpha}} {l_{\mu_i}}^{\alpha-1} e^{-l_{\mu_i}/\beta}, \tag{13}$$

where

$$\Gamma(\alpha) = \int_0^{\infty} z^{\alpha-1} e^{-z} \mathrm{d}z. \tag{14}$$

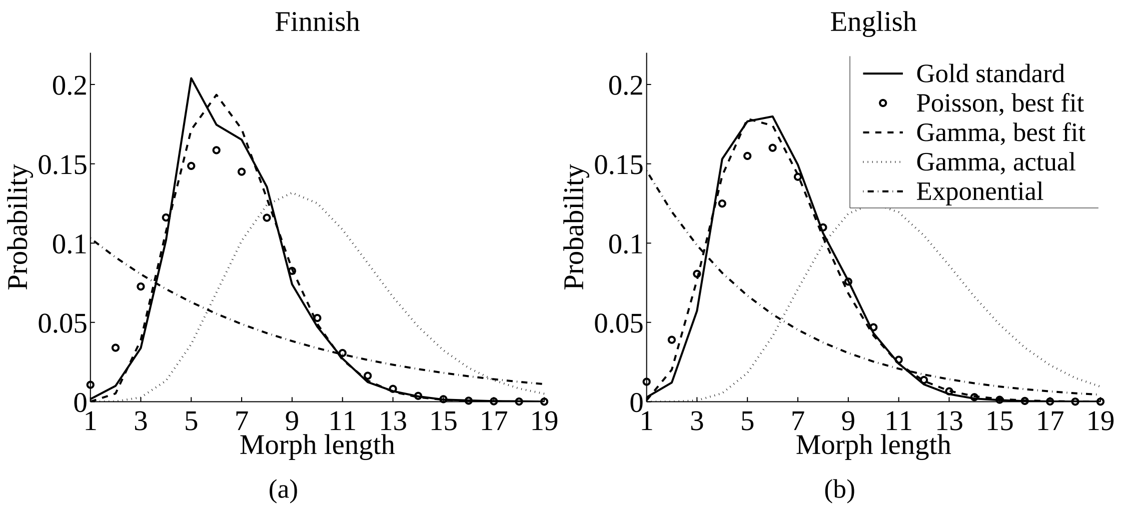


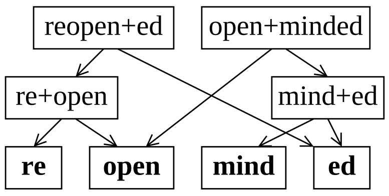
Figure 2: Length distributions for the morph types in Finnish and English lexicons (for data sets comprising 250 000 words). A distribution describes the proportion of morphs of a particular length, measured in letters. The solid line corresponds to the empirical distribution of morph lengths in the gold standard segmentation of the words, i.e., the desired result. The other curves are pdf:s that are suggested for modeling morph length (see Section 3.5.4).

There are two constants, α and β that determine the exact shape of the gamma pdf. The maximum value of the density occurs at $l_{\mu_i} = (\alpha - 1)\beta$, which corresponds to the most common morph length in the lexicon. The value of β governs the spikiness of the curve, the higher β the flatter and less discriminative is the pdf.

The Morfessor program makes use of a gamma prior distribution for morph length when the `-gammalendistr` switch is activated. As optional parameters the user can enter his prior belief of the most common morph length and a value for the β parameter. If these are omitted the default values 7.0 and 1.0 will be used, respectively. The effect of using a gamma pdf instead of the implicit exponential prior of Section 3.5.3 is usually beneficial for rather small data sizes (up to tens of thousands of words), but the difference diminishes for larger data sizes.

Figure 2 shows morph length distributions for Finnish and English data sets comprising 250 000 words. The gold standard corresponds to the length distribution obtained for a correct linguistic morpheme segmentation. As can be seen, it is possible to fit a gamma distribution ("Gamma, best fit") rather well to the gold standard curve. The attempt to fit a Poisson distribution ("Poisson, best fit") is less successful, because too much probability mass is allocated to the smallest values. However, the best segmentation results in practice are obtained, when an over-corrected gamma distribution is used as a prior ("Gamma, actual"). The gamma pdf can be compared to the implicit length prior ("Exponential") that results from using an end-of-morph marker (Section 3.5.3) instead of an explicit length distribution. The exponential distribution

Figure 3. Hypothetical splitting trees for two English words.



is clearly unrealistic when compared to the desired result in the gold standard, but it nevertheless often works almost as well as the gamma prior.

4 Search algorithm

To find the optimal morph lexicon and segmentation a greedy search algorithm is utilized. Initially each word in the corpus is a morph of its own. Different morph segmentations are proposed and the segmentation yielding the highest probability is selected. The procedure continues by modifying the segmentation, until no significant improvement is obtained.

Instead of computing probabilities as such, the negative logarithm of the probabilities (logprobs) are utilized and all products are replaced by sums. The negative logprobs can be considered as *code lengths* in the MDL framework. The code length of some observation x is thus related to the probability of x as follows: $L(x) = -\log P(x)$.

The search algorithm makes use of a data structure, where each distinct word form in the corpus has its own binary splitting tree. Figure 3 shows the hypothetical splitting trees of the English words ‘reopened’ and ‘openminded’. The leaf nodes of the structure are unsplit and they represent morphs that are present in the morph lexicon. The leaves are the only nodes that contribute to the overall code length of the model, whereas the higher-level nodes are used solely in the search. Each node is associated with an occurrence count (i.e., frequency) indicating the number of times it occurs in the corpus. The occurrence count of a node always equals the sum of the counts of its parents. For instance, in Figure 3 the count of the morph ‘open’ would equal the sum of the counts of ‘reopen’ and ‘openminded’.

During the search process, modifications to the current morph segmentation are carried out through the operation `resplitnode`, which is presented as pseudo-code in Algorithm 1. (Note that the pseudo-code does not correspond exactly to the structure of the actual program code, but the difference is small. Note also that the probabilities presented in Section 3 have been replaced by the corresponding code lengths.)

In the search, all distinct word forms in the corpus are sorted into random order and each word in turn is fed to `resplitnode`, which produces a binary splitting tree for that word. First, the word as a whole is considered as a morph to be added to the lexicon. Then, every possible split of the word in two substrings is evaluated. The split (or no split) yielding the lowest code length is selected. In case of a split, splitting

Algorithm 1 resplitnode(*node*)

Require: *node* corresponds to an entire word or a substring of a word

```
// REMOVE THE CURRENT REPRESENTATION OF THE NODE //
if node is present in the data structure then
    for all nodes $m$ in subtree rooted at node do
        decrease count($m$) by count(node)
        if $m$ is a leaf node, i.e., a morph then
            decrease $L(corpus \mid \mathcal{M})$ and $L(f_{\mu_1}, \dots, f_{\mu_M})$ accordingly
        if count($m$) $= 0$ then
            remove $m$ from the data structure
            subtract contribution of $m$ from $L(s_{\mu_1}, \dots, s_{\mu_M})$ if $m$ is a leaf node
// FIRST, TRY WITH THE NODE AS A MORPH OF ITS OWN //
restore node with count(node) into the data structure as a leaf node
increase $L(corpus \mid \mathcal{M})$ and $L(f_{\mu_1}, \dots, f_{\mu_M})$ accordingly
add contribution of node to $L(s_{\mu_1}, \dots, s_{\mu_M})$
bestSolution $\leftarrow [L(\mathcal{M} \mid corpus), node]$
// THEN TRY EVERY SPLIT OF THE NODE INTO TWO SUBSTRINGS //
subtract contribution of node from $L(\mathcal{M} \mid corpus)$, but leave node in data structure
store current $L(\mathcal{M} \mid corpus)$ and data structure
for all substrings pre and suf such that $pre \circ suf = node$ do
    for subnode in $[pre, suf]$ do
        if subnode is present in the data structure then
            for all nodes $m$ in the subtree rooted at subnode do
                increase count($m$) by count(node)
                increase $L(corpus \mid \mathcal{M})$ and $L(f_{\mu_1}, \dots, f_{\mu_M})$ if $m$ is a leaf node
        else
            add subnode with count(node) into the data structure
            increase $L(corpus \mid \mathcal{M})$ and $L(f_{\mu_1}, \dots, f_{\mu_M})$ accordingly
            add contribution of subnode to $L(s_{\mu_1}, \dots, s_{\mu_M})$
    if $L(\mathcal{M} \mid corpus) <$ code length stored in bestSolution then
        bestSolution $\leftarrow [L(\mathcal{M} \mid corpus), pre, suf]$
    restore stored data structure and $L(\mathcal{M} \mid corpus)$
// SELECT THE BEST SPLIT OR NO SPLIT //
select the split (or no split) yielding bestSolution
update the data structure and $L(\mathcal{M} \mid corpus)$ accordingly
if a split was selected, such that $pre \circ suf = node$ then
    mark node as a parent node of pre and suf
    // PROCEED BY SPLITTING RECURSIVELY //
    resplitnode(pre)
    resplitnode(suf)
```

of the two parts continues recursively and stops when no more gains in overall code length can be obtained by splitting a node into smaller parts. After all words have been processed once, they are again shuffled by random, and each word is reprocessed using `resplitnode`. This procedure is repeated until the overall code length of the model and corpus does not decrease significantly from one epoch to the next. (The threshold value for when to terminate the search can be set using the `-finish` switch in the Morfessor program. When the overall code length decreases less than the given value from one epoch to the next, the search stops. The default threshold value is 0.005 bits multiplied by the number of word types in the corpus.)

Every word is processed once in every epoch, but due to the random shuffling, the order in which the words are processed varies from one epoch to the next. It would be possible to utilize a deterministic approach, where all words would be processed in a predefined order, but the stochastic approach (random shuffling) has been preferred, because we suspect that deterministic approaches might cause unforeseen bias. If one were to employ a deterministic approach, it seems reasonable to sort the words in order of increasing or decreasing length, but even so, words of the same length ought to be ordered somehow, and for this purpose random shuffling seems much less prone to bias than, e.g., alphabetical ordering.

However, the stochastic nature of the algorithm means that the outcome depends on the series of random numbers produced by the random generator. The effect of this indeterminism can be studied by running the Morfessor program on the same data, but using different random seeds. The random seed is set using the `-rand` switch. The default random seed is zero.

5 Experiments

We report the results of two experiments carried out on Finnish and English data sets of different sizes. In the first experiment (Section 5.2) we study the differences that are obtained if the Morfessor model is trained on a word token collection (a corpus, where a word can occur many times) compared to a word type collection (a word list or a corpus vocabulary, where each distinct word form only occurs once). In the second experiment (Section 5.3) we study the effect of using the gamma length prior with different amounts of data.

Additionally, we report the measured memory consumption and running times of the Morfessor program on a PC for some of the data sets (Section 5.4).

First, we briefly describe the data sets and evaluation measures used in the experiments.

5.1 Experimental setup

We are concerned with a *linguistic morpheme segmentation task*. The goal is to find the locations of morpheme boundaries as accurately as possible. As a gold standard for the desired locations of the morpheme boundaries, *Hutmegs* is used (see Section 5.1.2). Hutmegs consists of fairly accurate conventional linguistic morpheme segmentations for a large number of Finnish and English word forms.

5.1.1 Finnish and English data sets

The Finnish corpus consists of news texts from the CSC (The Finnish IT Center for Science)[2] and the Finnish News Agency (STT). The corpus contains 32 million words. It has been divided into a development set and a test set, each containing 16 million words.

For experiments on English we use a collection of texts from the Gutenberg project (mostly novels and scientific articles)[3], and a sample from the Gigaword corpus and the Brown corpus[4]. The English corpus contains 24 million words. It has been divided into a development and a test set, each consisting of 12 million words. The *development sets* are utilized for selecting parameter values; in the current experiments the only parameter concerned is the most common morph length used in connection with the gamma length prior. The results that are reported are based on the *test sets*, which are used solely in the final evaluation.

Typically, a comparison of different algorithms on one single data set size does not give a reliable picture of how the algorithms behave when the amount of data changes. Therefore, we evaluate our algorithm with increasing amounts of test data. The amounts in each subset of the test set are shown in Figure 4a, both as number of word tokens (words of running text) and number of word types (distinct word forms). Figure 4b further shows how the number of word types grows as a function of the number of word tokens for the Finnish and English test sets. As can be seen, for Finnish the number of types grows fast when more text is added, i.e., many new word forms are encountered. In contrast, with English text, a larger proportion of the words in the added text has been observed before.

5.1.2 Morphological gold standard segmentation

The Helsinki University of Technology Morphological Evaluation Gold Standard (*Hutmegs*) (Creutz and Lindén, 2004) contains morpheme segmentations for 1.4 million Finnish word forms and 120 000 English word forms. The morpheme segmentations

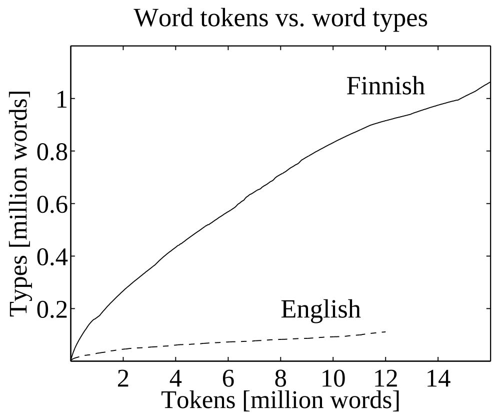
[2]`http://www.csc.fi/kielipankki/`

[3]`http://www.gutenberg.org/browse/languages/en`

[4]The Gigaword sample and the Brown corpus are available at the Linguistic Data Consortium: `http://www.ldc.upenn.edu/`.

word tokens	Finnish word types	English word types
10 000	5 500	2 400
50 000	20 000	7 200
250 000	65 000	17 000
12 000 000	–	110 000
16 000 000	1 100 000	–

(a)



(b)

Figure 4: (a) Sizes of the test data subsets used in the evaluation. (b) Curves of the number of word types observed for growing portions of the Finnish and English test sets.

have been produced semi-automatically using the two-level morphological analyzer FINTWOL for Finnish (Koskenniemi, 1983) and the CELEX database for English (Baayen et al., 1995). Both inflectional and derivational morphemes are marked in the gold standard. The Hutmegs package is publicly available on the Internet[5]. For full access to the Finnish morpheme segmentations, an inexpensive license must additionally be purchased from Lingsoft, Inc.[6] Similarly, the English CELEX database is required for full access to the English material[7].

As there can sometimes be many plausible segmentations of a word, Hutmegs provides several alternatives when appropriate, e.g., English 'evening' (time of day) vs. 'even+ing' (verb). There is also an option for so called "fuzzy" boundaries in the Hutmegs annotations, which we have chosen to use. Fuzzy boundaries are applied in cases where it is inconvenient to define one exact transition point between two morphemes. For instance, in English, the stem-final 'e' is dropped in some forms. Here we allow two correct segmentations, namely the traditional linguistic segmentation in 'invite', 'invite+s', 'invit+ed' and 'invit+ing', as well as the alternative interpretation, where the 'e' is considered part of the suffix, as in: 'invit+e', 'invit+es', 'invit+ed' and 'invit+ing'.

[5]`http://www.cis.hut.fi/projects/morpho/`

[6]`http://www.lingsoft.fi`

[7]The CELEX databases for English, Dutch and German are available at the Linguistic Data Consortium: `http://www.ldc.upenn.edu/`.

5.1.3 Evaluation measures

As evaluation measures, we use *precision* and *recall* on discovered morpheme boundaries. Precision is the proportion of correctly discovered boundaries among all discovered boundaries by the algorithm. Recall is the proportion of correctly discovered boundaries among all correct boundaries. A high precision thus tells us that when a morpheme boundary is suggested, it is probably correct, but it does not tell us the proportion of missed boundaries. A high recall tells us that most of the desired boundaries were indeed discovered, but it does not tell us how many incorrect boundaries were suggested as well. In order to get a comprehensive idea of the performance of a method, both measures must be taken into account.

The evaluation measures can be computed either using word tokens or word types. If the segmentation of word tokens is evaluated, frequent word forms will dominate in the result, because every occurrence (of identical segmentations) of a word is included. If, instead, the segmentation of word types is evaluated, every distinct word form, frequent or rare, will have equal weight. When learning the morphology of a language, we consider all word forms to be as important regardless of their frequency. Therefore, in this paper, precision and recall for word types is reported.

For each of the data sizes 10 000, 50 000, and 250 000 words, the algorithms are run on five separate subsets of the test data, and the average results are reported. Furthermore, statistical significance of the differences in performance have been assessed using T-tests. The largest data sets, 16 million words (Finnish) and 12 million words (English) are exceptions, since they contain all available test data, which constrains the number of runs to one.

5.2 Learning a morph lexicon from word tokens vs. word types

Different morph segmentations are obtained if the algorithm is trained on a collection of *word tokens* vs. *word types*. The former corresponds to a *corpus*, a piece of text, where words can occur many times. The latter corresponds to a *corpus vocabulary*, where only one occurrence of every distinct word form in the corpus has been listed.

We use the Morfessor Baseline model variant (see Section 3.5) to study how these two different types of data lead to different morph segmentations. In the Baseline, no explicit prior pdf:s are used for modeling morph frequency or length.

Figure 5 shows how precision and recall of segmentation develop for the two approaches (word tokens vs. word types) with different amounts of data. The general trend for both languages is that when a larger data set is utilized, precision increases while recall decreases. Furthermore, for every data size, learning from word types leads to clearly higher recall and only slightly lower precision than learning from word tokens[8]. Table 2 shows the segmentation of some words segmented by the Baseline

[8]According to T-tests the differences between measured precision and recall values are statistically

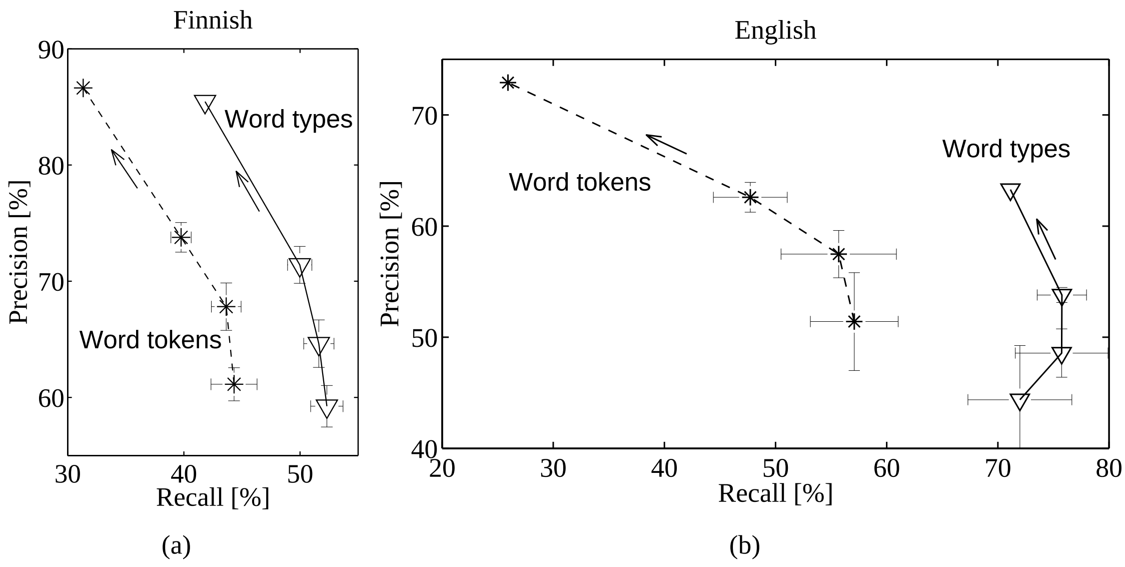


Figure 5: Comparison between the use of word tokens and word types as data for the Morfessor Baseline algorithm. The evaluation is performed on Finnish and English data sets of different sizes. (The exact sizes are reported in Figure 4a.) Arrows indicate the direction of increasing data size. The data points are averages over five runs on different subsets of the test data, except for the largest data set, where only one set was available (the entire test data). The standard deviations are shown as intervals around the data points. The graphs show that word types are generally a better choice, because the points on the word type curves lie on a higher level of recall and on almost as high a level of precision as the corresponding points on the word token curves.

algorithm when applied to the largest data sets, using both word tokens and word types as input. As can be seen, splitting is less common in the word token approach. If word tokens are used as data, common word forms tend to be added to the morph lexicon as entries of their own, regardless of whether a sensible morph segmentation could be found for them. This solution is of course the optimal one if one wishes to code a large corpus (instead of a vocabulary). But this is not optimal, if one wishes the method to be able to identify the inner structure of words regardless of how frequent they are.

We conclude that to obtain the linguistically best segmentation, it is better to learn using word types rather than word tokens. Furthermore, in order to obtain higher precision, the size of the data set can be increased.

5.3 Evaluation of the explicit gamma length prior

To study the effect of the explicit gamma length prior (Section 3.5.4), we compared the Morfessor Baseline-Length model variant to the Morfessor Baseline model variant which implicitly assumes an exponential length prior (Section 3.5.3). The experiments

signifi cant at the 0.01 level for both Finnish and English and all tested data sizes.

Table 2: Morph segmentations learned by the Morfessor Baseline variant for a few Finnish and English words. The frequency of the word forms in the data is indicated as well as the segmentation produced both when the word frequencies are utilized in the training (“wtoken”) and when not (“wtype”). The Finnish examples are inflections of the word ‘hellä’ (tender, affectionate). The segmentations in the “wtype” column happen to be correct, except for the plural marker ‘i’ that has been attached to the stem in the two last words.

Finnish			English		
freq	wtoken	wtype	freq	wtoken	wtype
41	hellä	hellä	624	tender	tender
6	hellää	hellä + ä	12	tenderer	tender + er
2	he + llään	hellä + än	25	tenderest	tender + est
1	hellä + ksi	hellä + ksi	1	tender + heartedness	tender + hearted + ness
6	hellänä	hellä + nä	3	tender + ize	tender + ize
42	hellästi	hellä + sti	124	tenderly	tender + ly
2	helli + ksi	helli + ksi	427	tenderness	tender + ness
3	hellinä	helli + nä	1	tenderness + es	tender + ness + es

were carried out on several data set sizes using both Finnish and English word types.

Figure 6 depicts the resulting precisions and recalls for each experiment. Evidently the use of an appropriate length prior improves the results markedly in particular with small data sets. With larger data sets the effect of the gamma prior is weaker. Nevertheless, the differences are statistically significant for all tested data set sizes (T-test level 0.01). In general the tendency that prior information is most useful with small data sets, is to be expected, since the more data there is, the more one can rely on the data to speak for itself.

5.4 Memory consumption and running time estimates

The memory consumption and running time of the Morfessor 1.0 program are roughly proportional to the number of word types in the data. The statistics of running the program on some data sets have been collected into Table 3. The measurements have been obtained on a PC with an AMD Duron 900 MHz processor and 512 MB RAM. In short, the memory consumption is moderate, but the running time for very large data sets may seem long. Note that the program is a Perl script that is not precompiled.

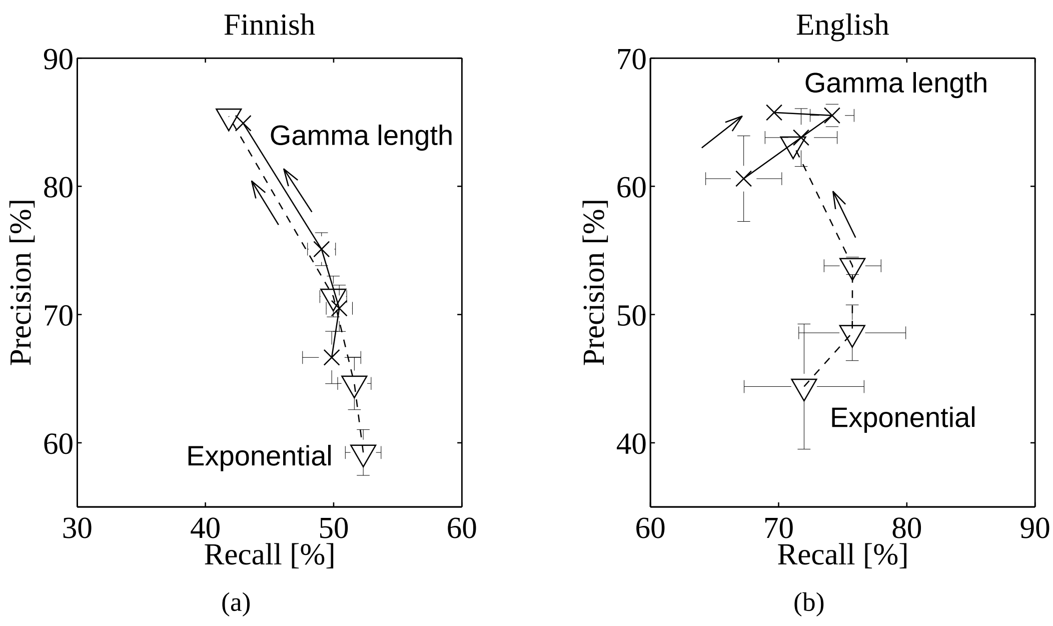


Figure 6: Evaluation of the use of a gamma distribution as a prior for morph length as opposed to using an implicit exponential length distribution. Arrows indicate the direction of increasing data size and the intervals around the data points are standard deviations. A gamma distribution for morph length is beneficial especially for small data sizes, because the “gamma curves” start off at a higher level of precision than the “exponential curves”.

6 Conclusive remarks

We hope that the Morfessor program will be useful to other researchers in their work. Comments and questions are welcome. Currently, only the Baseline and Baseline-Length versions of Morfessor exist as public resources. The later models (Morfessor Categories-ML and Categories-MAP) may be released in the future. By supplying public benchmarking resources, we wish to contribute to the promotion of research in the fascinating field of unsupervised morphology discovery and morpheme segmentation.

Acknowledgments

We thank Sami Virpioja for his feedback on the derivation of formulas in the appendices. We are grateful for the inspirational discussions with Mikko Kurimo, Teemu Hirsimäki, Vesa Siivola and Panu Somervuo from the Speech group of the NNRC. Our special thank goes to Teemu Hirsimäki for commenting and modifying the pseudo-code part of the description of the search algorithm. Furthermore, we would like to thank Krister Lindén for his continuous interest towards our work and his valuable feedback.

Table 3: Measured memory consumption and running times for some Finnish and English data sets. The sizes of the data sets are indicated as both the number of word tokens and the number of word types in the data. The figures marked with an asterisk (*) were obtained using the `-savememory` switch of the Morfessor program. Compared to the corresponding figures that result from not using the memory saving option, the memory consumption is lower, but the running time is considerably longer.

Language	Word tokens	Word types	Memory consumpt.	Running time
English	250 000	20 000	8 MB	7 min.
Finnish	250 000	70 000	19 MB	35 min.
English	12 000 000	110 000	25 MB	35 min.
Finnish	16 000 000	1 000 000	220 MB	8 hours
Finnish	16 000 000	1 000 000	160 MB*	15 hours*

We express our gratitude to the Graduate School of Language Technology in Finland for funding part of this work, and we thank the Finnish IT Center for Science and the Finnish National News Agency for providing text corpora.

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`.

R. Harald Baayen. 2001. *Word Frequency Distributions*. Kluwer Academic Publishers.

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

Stanley F. Chen. 1996. *Building Probabilistic Models for Natural Language*. Ph.D. thesis, Harvard University.

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. 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 (AKRR'05)*. (to appear).

Mathias Creutz and Krister Lindén. 2004. Morpheme segmentation gold standards for Finnish and English. Technical Report A77, Publications in Computer and Information Science, Helsinki University of Technology.

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

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

K. Hacioglu, B. Pellom, T. Ciloglu, O. Ozturk, M. Kurimo, and M. Creutz. 2003. On lexicon creation for Turkish LVCSR. In *Proc. Eurospeech'03*, pages 1165–1168, Geneva, Switzerland.

Kimmo Koskenniemi. 1983. *Two-level morphology: A general computational model for word-form recognition and production*. Ph.D. thesis, Two-level morphology: A general computational model for word-form recognition and production, University of Helsinki.

Young-Suk Lee. 2004. Morphological analysis for statistical machine translation. In *Proc. Human Language Technology, Conference of the North American Chapter of the Association for Computational Linguistics*, Boston, Massachusetts, USA.

Masaaki Nagata. 1997. A self-organizing Japanese word segmenter using heuristic word identification and re-estimation. In *Proc. Fifth workshop on very large corpora*, pages 203–215.

V. Siivola, T. Hirsimäki, M. Creutz, and M. Kurimo. 2003. Unlimited vocabulary speech recognition based on morphs discovered in an unsupervised manner. In *Proc. Eurospeech'03*, pages 2293–2296, Geneva, Switzerland.

A Derivation of a noninformative frequency prior

Suppose that there are a total number of N morph tokens in the segmented corpus and that these morphs represent M different morph types. What is the probability of coming up with a particular frequency distribution, i.e., a set of M frequencies that sum up to N?

We further suppose that the probability distribution is a noninformative prior, that is, all frequency distributions are equally likely. It follows that the probability of one particular distribution is one divided by the number of possible ways of choosing M positive integers (the M frequencies) that sum up to N.

Imagine that the N morph tokens are sorted into alphabetical order and each morph is represented by a binary digit. Since some morphs occur more than once, there will be sequences of several identical morphs in a row. Now, initialize all N bits to zero. Next, every location, where the morph *changes*, is switched to a one, whereas every location, where the morph is identical to the previous morph, is left untouched. There are $\binom{N}{M}$ possibilities of choosing M bits to switch in a string of N bits. However, as the value of the first bit is known to be one, it can be omitted, which leaves us with $\binom{N-1}{M-1}$ possible binary strings. Therefore the probability of the frequency distribution is:

$$P(\text{frequency distribution}) = 1/\binom{N-1}{M-1} = \frac{(M-1)!(N-M)!}{(N-1)!}. \qquad (15)$$

B Derivation of a Zipfian frequency prior

Zipf has studied the relationship between the frequency of a word, f, and its rank, z. The rank of a word is the position of the word in a list, where the words have been sorted according to falling frequency. Zipf suggests that the frequency of a word is inversely proportional to its rank. Mandelbrot has refined Zipf's formula, and suggests a more general relationship; see, e.g., (Baayen, 2001):

$$f = C(z+b)^{-a}, \qquad (16)$$

where C, a and b are parameters of a text.

Let us derive a probability distribution from Mandelbrot's formula. The rank of a word as a function of its frequency can be obtained by solving for z from Eq. 16:

$$z = C^{\frac{1}{a}} f^{-\frac{1}{a}} - b. \qquad (17)$$

Suppose that one wants to know the number of words that have frequency f rather than the rank of a word with frequency f. We denote this *frequency of frequency* f by $n(f)$. An estimate for $n(f)$ is obtained as the difference in rank between a word with frequency f and a word with frequency $f+1$:

$$n(f) = z(f) - z(f+1) = C^{\frac{1}{a}}\big(f^{-\frac{1}{a}} - (f+1)^{-\frac{1}{a}}\big). \qquad (18)$$

A probability distribution for f is obtained by dividing $n(f)$ by the total number of word tokens, which equals the sum of frequencies over all possible frequencies. We denote the highest frequency by F. Thus,

$$P(f) = \frac{n(f)}{\sum_{f'=1}^{F} n(f')} = \frac{C^{\frac{1}{a}}\left(f^{-\frac{1}{a}} - (f+1)^{-\frac{1}{a}}\right)}{\sum_{f'=1}^{F} C^{\frac{1}{a}}\left(f'^{-\frac{1}{a}} - (f'+1)^{-\frac{1}{a}}\right)} = \frac{f^{-\frac{1}{a}} - (f+1)^{-\frac{1}{a}}}{1 - (F+1)^{-\frac{1}{a}}}. \quad (19)$$

When the highest frequency F is assumed to be big, we can make the approximation $F \approx \infty$ without any loss of accuracy that is of practical significance:

$$P(f) \approx \lim_{F\to\infty} \frac{f^{-\frac{1}{a}} - (f+1)^{-\frac{1}{a}}}{1 - (F+1)^{-\frac{1}{a}}} = f^{-\frac{1}{a}} - (f+1)^{-\frac{1}{a}}. \quad (20)$$

Rather than setting a value for the parameter a we want to shape the probability distribution according to our prior belief of the proportion of *hapax legomena* (h), i.e., the proportion of words occurring only once in the corpus:

$$h = P(1) = 1^{-\frac{1}{a}} - 2^{-\frac{1}{a}} = 1 - \left(\frac{1}{2}\right)^{\frac{1}{a}}. \quad (21)$$

Substituting a in Eq. 20 by h yields:

$$P(f) = f^{\log_2(1-h)} - (f+1)^{\log_2(1-h)}. \quad (22)$$

The exponent $\log_2(1-h)$ is always negative. Therefore the resulting probability distribution follows a *power law* and it is represented by a straight line when plotted in a graph with logarithmic scales on both axes. We assume that the derived probability distribution applies to morphs as well as to words.

C Probability of the frequency of individual morphs

It is difficult to compare the implications of the Zipfian frequency prior in Eq. 22 to those of the noninformative prior in Eq. 15. The Zipfian prior separately assigns a probability to the frequency of each morph, whereas the noninformative prior at once assigns a probability for the whole frequency distribution. In the following we will derive an approximation of the probability of the frequency of an individual morph in the noninformative prior scheme. This facilitates a comparison between the Zipfian and noninformative prior approaches.

Suppose that there are N morph tokens and M morph types. Next, f occurrences of a new morph are added, which increases the number of morph tokens to $N+f$ and the number of morph types to $M+1$. We compute the conditional probability of adding a morph with frequency f when the initial position (N, M) is given:

$$P(f \mid N, M) = \frac{P(f, N, M)}{P(N, M)} = \frac{P(\text{freq. distr.}(N+f, M+1))}{P(\text{freq. distr.}(N, M))}. \quad (23)$$

According to Eq. 15 this equals:

$$P(f \mid N, M) = \binom{N-1}{M-1} \Big/ \binom{N+f-1}{M} = \frac{(N-1)!M!(N-M+f-1)!}{(N+f-1)!(M-1)!(N-M)!}. \tag{24}$$

The factorials are rewritten using Stirling's approximation: $n! \approx (n/e)^n \sqrt{2\pi n}$:

$$P(f \mid N, M) = \frac{(N-1)^{N-1/2} M^{M+1/2} (N-M+f-1)^{N-M+f-1/2}}{(N+f-1)^{N+f-1/2}(M-1)^{M-1/2}(N-M)^{N-M+1/2}}. \tag{25}$$

The factors that are constant with respect to f are rewritten as C_1:

$$P(f \mid N, M) = C_1 \cdot \frac{(N-M+f-1)^{N-M+f-1/2}}{(N+f-1)^{N+f-1/2}}. \tag{26}$$

For f values that are much smaller than N and M approximately the following holds for the bases: $N-M+f-1 \approx N-M$ and $N+f-1 \approx N$. Thus,

$$P(f \mid N, M) = C_1 \cdot \frac{(N-M)^{N-M+f-1/2}}{N^{N+f-1/2}} = C_1 \cdot \frac{(N-M)^{N-M-1/2}(N-M)^f}{N^{N-1/2} N^f}. \tag{27}$$

The factors that are now constant with respect to f are combined with C_1 into C_2:

$$P(f \mid N, M) = C_2 \cdot \frac{(N-M)^f}{N^f} = C_2 \cdot \Big(\frac{N-M}{N}\Big)^f. \tag{28}$$

This results in an *exponential distribution*. That is, the probability decreases exponentially with the value of the frequency. (This only applies to f values that are small compared to the total number of tokens N and types M.) The exponential distribution can be directly compared to the power-law distribution that results from applying the Zipfian prior in the previous section.