Allomorfessor: Towards Unsupervised Morpheme Analysis

Oskar Kohonen, Sami Virpioja, and Mikaela Klami
Adaptive Informatics Research Centre, Helsinki University of Technology
{oskar.kohonen,sami.virpioja,mikaela.klami}@tkk.fi

Abstract

Many modern natural language processing applications would benefit from automatic morphological analysis of words, especially when dealing with morphologically rich languages. Consequently, there has been an increasing amount of research on the task of unsupervised segmentation of word forms into smaller useful units, i.e. morphs or morphemes. The linguistic phenomenon of allomorphy, where one morpheme has several different surface forms, places limits on the quality of morpheme analysis achievable by segmentation alone. We extend the morphological segmentation method, Morfessor Baseline, to model allomorphy. Our unsupervised method discovers common base forms for allomorphs from an unannotated corpus. We evaluate the method by participating in the Morpho Challenge 2008 competition, where automatic morphological analyses of corpora in English, German, Turkish and Finnish are compared against a linguistic gold standard. Our method achieves high precision, but low recall, and therefore low F-measure scores. We conclude that our method currently undersegments, but that the main approach is promising.

Categories and Subject Descriptors

I.2 [**Artificial Intelligence**]: I.2.6 Learning; I.2.7 Natural Language Processing

General Terms

Algorithms, Experimentation, Languages

Keywords

Morphology, Morphological Analysis, Unsupervised Learning

1 Introduction

Morphological analysis is crucial to many modern natural language processing applications, especially when dealing with morphologically rich languages where the enormous number of possible inflected word forms would otherwise lead to severe problems with data sparsity, computational efficiency, and general application performance. Consequently, there has recently been an increasing amount of research on the task of automatic unsupervised segmentation of word forms into smaller useful units, i.e. morphs or morphemes [8]. Many current morphological analysis tools are already capable of extracting ad-hoc morphological segmentations that are of good quality for the purposes of, e.g., speech recognition [2].

Ultimately, we would like to perform not morphological segmentation, but the more difficult task of morpheme analysis, where the aim is not only to segment the corpus word forms into subparts, but also to identify surface forms corresponding to morphological labels. For this task, the phenomenon of allomorphy limits on the quality of morpheme analysis achievable by segmentation alone. Allomorphy is defined in linguistics as a morphological phenomenon in which an underlying morpheme-level unit may have two or more morph-level surface realizations which only occur in a complementary distribution: The occurrences of the different allomorphs of a given morpheme are conditioned by the morpho- and phonotactical environment of the morpheme, and any given environment may therefore trigger the occurrence of only one of the allomorphic variants. For example, in Finnish, the singular genitive case is marked with a suffix `n`, e.g. `auto` (car) – `auton` (car's). Many Finnish nouns have allomorphic variation in their stems when producing the singular genitive: `kenkä` (shoe) – `kengän` (shoe's), `pappi` (priest) – `papin` (priest's), `tapa` (habit) – `tavan` (habit's). A segmentation based approach does not model the fact that the allomorphic variation in the stems of the singular genitive represent the same morpheme as their base form.

Thus, instead of merely splitting the word forms into morph-like units, a morpheme analysis system should also be capable of reducing the potentially morphologically transformed stems into their lexical base forms, and handling cases of allomorphic variation in both stems and affixes.

In addition to being more linguistically motivated, it is well reasonable to expect that such a morpheme-level analysis would also prove to be superior to a mere segmentation of word forms for the purposes of many natural language processing applications, due to reductions in lexicon size and identification of base forms. Indeed, in Morpho Challenge 2007 [7], the performances of the morphological analysis systems submitted to the contest were evaluated on an actual Information Retrieval task. However, even if the reduction of word forms into stems and adherent affixes should hypothetically be very valuable for an Information Retrieval task, the best challenge results were produced by methods that were mainly segmentation-based: Bernhard's method [1] and Morfessor Baseline [3, 4].

As can be seen from the examples above, the allomorphic variations of a certain morpheme commonly have very similar orthographic forms. In this paper, we describe an extension to the Morfessor Baseline algorithm towards analyzing allomorphic variation, using string similarity to identify allomorphs. Our method provides a morphological segmentation and identifies potential base forms for stems, but not suffixes.

2 Allomorfessor Model for Morpheme Analysis

Our learning problem is the following: We have an unannotated text corpus that contains words in their inflected forms. From this corpus we try to infer a morphological segmentation and a common base form for all inflected forms that stem from the same root word. Since our method is completely unsupervised, it cannot know which word form is considered the correct base form, but chooses one of the alternative forms as the base form. When successful, all allomorphic variants of a stem will be replaced by the chosen base form. Allomorphic variation in suffixes is not taken into account.

In this section, we describe a model that tries to solve the problem. We call the model *Allomorfessor*, as it is an extension to the Morfessor [4] method and tries to solve the problems caused by allomorphy. We start by discussing how *Maximum a Posteriori* estimation is applied to find the optimal probabilistic model. Then we show how instead of modeling the full corpus, we can model just a list of all the word forms in the corpus by using a very simple model for the word level. In subsections 2.4–2.5 we describe the morpheme-level model designed for finding allomorphic variants. Finally, we discuss the search algorithm for the model.

2.1 Maximum A Posteriori Estimation

Our model is probabilistic and generative, meaning that it can give a probability distribution over all possible text corpora. The model is denoted by $\mathcal{M}$ and the corpus used for training the model

as corpus.

With *Maximum a Posteriori* (MAP) estimation, we try to find the model $\mathcal{M}$ that is the most probable given the training corpus:

$$\mathcal{M}_{\text{MAP}} = \arg\max_{\mathcal{M}} P(\mathcal{M}|\text{ corpus}) = \arg\max_{\mathcal{M}} P(\mathcal{M})P(\text{corpus}|\mathcal{M}) \tag{1}$$

$P(\mathcal{M})$ is the Bayesian prior probability for the model and $P(\text{corpus}|\mathcal{M})$ is likelihood of the training corpus given the model. Compared to Maximum Likelihood (ML) estimation, MAP provides a systematic way of balancing the model complexity and accuracy, and thus helps with the problem of overlearning (see, e.g., chapter 3 in [6]).

2.2 Word and Morpheme Submodels

Assume that our model consists of two parts, word-level model $\mathcal{M}_W$ and morpheme-level model $\mathcal{M}_M$. Furthermore, both models can be divided into two parts: grammar $\mathcal{G}$ and lexicon $\mathcal{L}$. Word grammar describes how words are combined to form sentences (or some other multi-word constructions) in the corpus. Word lexicon gives the words that can be used by the grammar. Similarly, morpheme grammar models word-internal syntax and morpheme lexicon gives the morphemes that construct the words.

First we make two simplifying assumptions: word grammar is independent of the morpheme model, and morpheme grammar and morpheme lexicon are mutually independent. With this and the Bayes' rule with get the following:

$$\arg\max_{\mathcal{G}_W,\mathcal{L}_W,\mathcal{G}_M,\mathcal{L}_M} P(\text{corpus}|\mathcal{G}_W,\mathcal{L}_W)P(\mathcal{L}_W|\mathcal{G}_M,\mathcal{L}_M)P(\mathcal{G}_W|\mathcal{L}_W)P(\mathcal{G}_M)P(\mathcal{L}_M). \tag{2}$$

We are currently interested in morpheme-level modeling, so we can set $\mathcal{L}_W$ to be a lexicon consisting of all word forms in the training corpus and $\mathcal{G}_W$ to be a unigram model. As both are constants with respect to the morph model, the task simplifies to:

$$\arg\max_{\mathcal{G}_M,\mathcal{L}_M} P(\mathcal{L}_W|\mathcal{G}_M,\mathcal{L}_M)P(\mathcal{G}_M)P(\mathcal{L}_M). \tag{3}$$

This is equivalent to the approach used in Morfessor [4], but instead of modeling the original corpus, we are now modeling a lexicon of the words in the corpus.[1]

2.3 Model Overview

In its core, the Allomorfessor model is a probabilistic context-free grammar (PCFG). Terminals of the grammar are units resembling linguistical morphemes, specifically root stems and affixes. We call the non-terminals *virtual morphs*; they are units that have substructure. We denote both the virtual morphs and stems with μ. I.e., if μ does not have a substructure, it is a (root) stem, otherwise it is a virtual morph. For an illustration, see Figure 2.3.

The model is similar to Morfessor Baseline: A word can either be a stem with no further substructure, or it can be a virtual morph consisting of two parts that in turn may be stems or virtual morphs. However, compared to Morfessor Baseline we add the notion of *mutation* to model allomorphic variation. A virtual morph splits the surface string in two parts, which we here call *virtual prefix* and *virtual suffix*, to avoid confusing them with actual prefix and suffix morphemes. In case of a compound word the virtual prefix may be the first word in the compound, and similarly for the virtual suffix. These virtual prefixes and suffixes might also still themselves have substructure. Our model allows mutations only to the virtual prefix. This means that a virtual morph consists of a base form, a virtual suffix, and a mutation that transforms the base form into the observed virtual prefix. The mutation may be empty, in which case the base form

[1] This has been recommended to be done also with Morfessor by setting all the word counts to one. Otherwise, frequent word forms are often undersegmented.

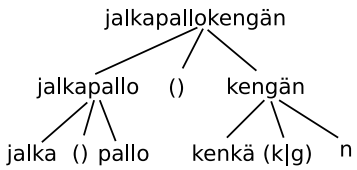

Figure 1: The analysis of the Finnish word `jalkapallokengän` (football shoe's). First the word is split in two with an empty mutation denoted as `()`, then the the virtual prefix `jalkapallo` is further split into the stems `jalka` and `pallo`. The virtual suffix `kengän` is split into the base form `kenkä`, the corresponding mutation `(k|g)` which transforms it into the virtual prefix `kengä`, and the stem `n`

is simply the virtual prefix. An empty mutation corresponds to a regular inflection or compound word, where the stem does not undergo any changes.

For generality, mutations to the virtual suffix should also be modeled. However, we ignore it for now, since mutations to the virtual prefix are much more common in the studied languages. In principle, an equivalent solution should work for that case as well.

2.4 Model Probabilities

In this subsection, we give a formal description of the probabilities of Equation 3 for the Allomorfessor model. (For simplicity, the subscript M of is left out from the symbols denoting morpheme model parts.) The formulation follows closely the work by Creutz and Lagus [4].

Every word form in the word lexicon is represented by one virtual morph. Thus the probability of the word lexicon is simply

$$P(\mathcal{L}_W|\mathcal{G},\mathcal{L}) = \prod_{j=1}^{M_W} P(\mu_j), \tag{4}$$

where M_W is the number of words in the lexicon. The probability of the morph μ is estimated from its frequency: the more often morph is referred to, either from the word lexicon, or from virtual morphs, the more probable it is.

The morph lexicon $\mathcal{L}$ consists of the virtual morphs. The probability of the morph lexicon is based on the properties of the virtual morphs

$$P(\mathcal{L}) = P(\text{size}(\mathcal{L}) = M)P(\text{properties}(\mu_1)\ldots\text{properties}(\mu_M))M! \tag{5}$$

If a non-informative prior is used for the probability of the lexicon size, its effect is minimal and it can be neglected. The factor $M!$ is explained by the fact that there are $M!$ possible orderings of M items, and the lexicon is the same regardless of the order in which the morphs are discovered.

The properties of the morphs are divided into two parts, usage and form:

$$\begin{aligned} P(\text{properties}(\mu_1)\ldots\text{properties}(\mu_M)) &= P(\text{usage}(\mu_1)\ldots\text{usage}(\mu_M)) \times \\ &\quad P(\text{form}(\mu_1)\ldots\text{form}(\mu_M)) \end{aligned} \tag{6}$$

The usage includes properties of the morph itself and the properties of its context. In this model, we use only morph frequencies. For the probability of the frequency distribution, we use a non-informative, implicit frequency prior

$$P(\text{usage}(\mu_1)\ldots\text{usage}(\mu_M)) = P(\text{freq}(\mu_1)\ldots\text{freq}(\mu_M)) = 1/\binom{N-1}{M-1}, \tag{7}$$

where N is the sum of the frequencies of the morphs.

The form of a morph is its symbolic representation. Forms of the morphs are assumed to be independent:

$$P(\text{form}(\mu_1)\ldots\text{form}(\mu_M)) = \prod_{i=1}^{M} P(form(\mu_i)) \tag{8}$$

As described, the morph lexicon $\mathcal{L}$, has a hierarchical structure. μ_i is either a root stem represented by a string of letters or a virtual morph consisting of a base form, a virtual suffix and a (possibly empty) mutation δ_k. The base form μ_j and virtual suffix μ_k may be stems or virtual morphs.

$$P(form(\mu_i)) = \begin{cases} 1 - P(\text{sub}) & P(\text{length}(\mu_i)) \prod_{j=1}^{\text{length}(\mu_i)} P(\hat{c}_{ij}), & \text{if } \mu_i \text{ is a stem} \\ P(\text{sub}) & P(\mu_j)P(\delta_k)P(\mu_k), & \text{if } \mu_i \text{ is a virtual morph.} \end{cases} \tag{9}$$

For stems, $\hat{c}_{ij}$ is the jth character of the stem, and $P(\text{sub})$ is the probability that a morph has substructure.

The lengths of the morphs are modeled explicitly using a gamma distribution:

$$P(\text{length}(\mu)) = \frac{1}{\Gamma(a)b^a}\text{length}(\mu)^{a-1}e^{-\text{length}(\mu)/b}. \tag{10}$$

Grammar $\mathcal{G}$ of the model contains the set of mutations Δ. Similarly to the lexicons,

$$P(\mathcal{G}) = P(\text{size}(\Delta) = M_\delta)P(\text{properties}(\delta_1)\dots\text{properties}(\delta_{M_\delta}))M_\delta!, \tag{11}$$

and properties can be divided into usage and form. Usage features include only the frequencies; the non-informative prior is applied. The probabilities for forms of mutations are given in the following sections that examine mutations more closely. Mutations that occur often are much more likely to correspond to correct, systematic analyses that could be seen as being part of the language. Therefore they are sensible to model in their own lexicon, rather than modeling each use of a mutation independently. Because of this modeling decision, it becomes important that the mutations are such that when the linguistically equivalent changes happen for two pairs of words, also the applied mutation is the same.

2.5 Mutations

When designing the mutations for modeling allomorphy, there are several considerations that need to be balanced. First, if we are not careful in how our mutations are allowed to change the stem, we easily get incorrect analyses. For example, the edit distance between `beat` and `peat` is only one, but they are certainly not allomorphs of the same morpheme, but rather distinct morphemes. As a general rule, with a few obvious exceptions, it appears that allomorphic variation happens close to the affix. Therefore we limit our mutations to modify the end of the base form, since our mutations only affect the virtual prefix.

Second, we would like our mutation lexicon to extract common orthographical rules. For example, in some English stems the ending letter `y` will change to `i`: `deny` – `denied` or `pretty` – `prettier`. A less regular example is the Finnish genitive, where there is a "softening" of the preceding consonants, for example: `kenkä` (shoe) – `kengän`, `tanko` (pole) – `tangon`, `pappi` (priest) – `papin`. In the ideal case the same mutation could express the common rule. More pragmatically, complex cases, such as the Finnish genitive, should be modeled with as few different mutations as possible.

Third, to be computationally feasible, the mutations need to be efficient to compute from a pair of strings. This makes restrictions especially to very flexible mutation types.

We consider two kinds of mutations: 1) morph-end mutations and 2) substitution-deletion mutations.

2.5.1 Morph-End Mutations

This class of mutations is very simple: Mutation describes what is to be removed from the end of the base form, and what needs to be added to yield the virtual prefix. For example if the base form is `deny` and the virtual prefix is `deni` (for example combined with the suffix `ed`). Then this mutation would be denoted `-y+i`. In other words, the mutation consists of two operations: Deleting `y` and then adding `i`. In case more needs to be removed or added, we simply add more operations, e.g. `ihminen` (human) `-nen+sen` → `ihmisen` (human's).

The probability of the form for these mutations is set to be

$$P(\text{form}(\delta_i)) = P((\text{length}(\text{ops}_{\delta_i}))\prod_{j}^{\text{ops}} P(\text{op}_j) \quad (12)$$

$$P(\text{op}_j) = \begin{cases} P(\text{del})\frac{1}{\Sigma} & \text{if } op_j \text{ is a deletion} \\ P(\text{ins})\frac{1}{\Sigma} & \text{if } op_j \text{ is an insertion} \end{cases}, \quad (13)$$

In the equations above, ops_{δ_i} is the set of operations in the mutation δ_i, $P((\text{length}(\text{ops}_{\delta_i}))$ is a gamma distribution, similarly to the Equation 10, and Σ is the alphabet size, which is included because a specific letter is deleted or inserted. The probabilities $P(\text{del})$ and $P(\text{ins})$ denote the relative probabilities of deletions and insertions.

The weakness of this mutation type is that it cannot generalize mutations where a letter inside the base form changes. For example, the mutation is intuitively the same in the following case: `kenkä` – `kengän` and `tanko` – `tangon`. The second to last letter `k` is changed into a `g`. However, with the simple mutations presented the mutations would be different, `-kä+gä` and `-ko+go`, respectively.

2.5.2 Substitution-Deletion Mutations

This is a more complicated mutation type which solves the problem described at the end of the previous section. In this mutation type, there are two operations: substitutions and deletions of individual letters. If a base form candidate cannot be transformed into the target stem using these operations only, that base form candidate is discarded. The reasoning behind this is that the virtual suffix should be the one adding letters to the end of the virtual morph, not the mutation.

The operations themselves also work as follows. The operation contains a target letter to modify. The execution of the mutation starts scanning from the end of the base form until the target letter is found. Once found, the operation is applied. In case of a substitution, the target letter is replaced, in case of deletion, it is deleted. The next operation starts scanning for its target from the position before the one where the previous operation was applied. This is continued until no operations remain. For example, mutating `kenkä` into `kengä` is done using the mutation `k|g`, where `x|y` denotes an operation where the letter `x` is substituted with the letter `y`. First we scan from the end of the string searching for the target letter `k`. Once we find it, we replace it with a `g`. Note that the same mutation also changes `tanko` into `tango`. For a more complicated example, consider changing `ranta` (shore) into `rann`. The needed mutation is `-a t|n`, where the operations are separated by spaces, and `-x` means deletion of the target letter `x`. Thus the whole execution proceeds: Find the `o` that is last in the base form and delete it. Then find the previous `t` from the position left of the first operation, and replace it with an `n`.

There are some cases where the scanning process as described thus far is inadequate. For example `ihminen` (human) – `ihmisen` (human's). For these cases we augment the scanning with a possibility to skip letters. To express "change the second last `n` to an `s`" we write `2n|s`, i.e., a number expresses which letter of it's kind to match. When the first one is desired the number may be omitted.

Calculating the smallest mutation of this kind for arbitrary strings is not a trivial task. We build on work in approximate string matching. There exists powerful dynamic programming based algorithms to calculate the Levenshtein (edit) distance (see, e.g., [9], section 5.1.1 and 5.1.3). The Levenshtein distance measures the distance between two strings by how many insertion, deletion and substitution operations are needed to convert one string to another. It is also possible to assign different costs to the different operations. It is fairly easy to modify the basic algorithm (described in [9] section 5.1) to not only produce the distance itself, but also the optimal sequence of operations. This can be done by iterating backwards through the cost matrix produced by the calculation, and always choose the cheapest parent node, in a similar way to how the Viterbi algorithm finds the maximum likelihood path in a Hidden Markov Models.

In our case, we want the optimal path, but it should not contain insertions. Therefore we set the cost on the insertion operation to a number larger than $2 * \max(\text{length}(s), \text{length}(t))$, where s is the source string, that is the base form and t is the target string, that is the virtual prefix. The

weights on substitution and deletion are set to 2 and 1 respectively, to favor deletions, but since we don't currently use the resulting distance, they could also both be set to 1. If there are alternative paths of which some contain insertions and others do not, the ones that do not will always be cheaper, because of the high cost of insertions. If the edit distance returned is larger than the weight set for the insertions, then we know that insertions were necessary, and can discard the candidate base form immediately. Once we have a path of optimal substitutions and deletions, given the above constraints, we can easily transform those into the mutation format described earlier.

The prior probability for the substitution-deletion mutations was selected to be as follows:

$$P(\text{form}(\delta_i)) = P(\text{length}(\text{ops}_{\delta_i})) \prod_{j}^{\text{ops}} P(\text{skip}_j) P(\text{op}_j) \quad (14)$$

$$P(\text{skip}_j) = P(\text{length}(\text{skip}(\text{op}_j))) \quad (15)$$

$$P(\text{op}_j) = \begin{cases} P(\text{del}) \frac{1}{\Sigma} & \text{if } op_j \text{ is a deletion} \\ P(\text{sub}) \frac{1}{\Sigma^2} & \text{if } op_j \text{ is an substitution} \end{cases} \quad (16)$$

The formulas are similar to Equations 13 and 13, but there is a separate gamma distribution $P(\text{length}(\text{skip}(\text{op}_j)))$ for the skips. Since the substitution operation requires the choice of both the target letter and the replacing one, the alphabet size Σ is squared in the denominator.

2.6 Learning the Model

Our model learning scheme is based on iteratively improving the model probability $P(\mathcal{L}_W|\mathcal{G}_M, \mathcal{L}_M)$ $P(\mathcal{G}_M)P(\mathcal{L}_M)$, by analyzing one word at the time, and selecting the analysis of that word that maximizes the probability.

First, we look at the case where the mutation is empty. That case is equivalent to the search algorithm used in Morfessor Baseline. We analyze the word w which can be split between any pair of letters. If follows that each word w has $2^{\text{len}(w)-1}$ possible analyses. In order to reduce the search space, we search greedily for the split that maximizes the model probability. Unless w is best most probable to be a stem, we recursively repeat the process for the parts of the virtual morph. In other words, for each word w, we evaluate the model probability when w is analyzed either as a stem or a virtual morph split at position $i \in 1, \text{len}(w) - 1$. Should the virtual morph maximize model probability, we recursively perform the same analysis on both virtual prefix and virtual suffix.

Now we look at the case where the mutation is nonempty. Because we are trying to find common base forms for different allomorphs, we restrict the stem to be a word that exists in the dictionary. Alternatively, one could restrict to any morph that exists in the lexicon. We chose the former as a conservative alternative. The words in the dictionary are assumed to be real words, whereas the morph lexicon might contain incorrect analyses.[2] As before, we split the word w at position i so that the virtual prefix $v_{\text{pre}} = w_{0..(i-1)}$ and the virtual suffix $v_{\text{suf}} = w_{i..(len(w)-1)}$, where $w_{i..j}$ denotes the part of the string w starting at position i and ending at position j. A brute-force algorithm would for each v_{pre} consider each other word w' from the lexicon as a candidate base form. This would lead to $(len(w) - 1)\frac{N(N-1)}{2}$ comparisons, where N is the number of words in the corpus. This is clearly infeasible for large data sets.

The majority of words are poor candidates for base forms, since they do not share much in common with the target virtual prefix v_{pre}. Therefore, we further assume that the base form needs to share a common prefix with the target word. Finding candidates with a shared prefix from an ordered list can be done in $O(k \log(N))$ time, where k is the number of found candidates. We can further reduce the search time by doing it only once for each word w, rather than for each v_{pre}. For the whole search algorithm we thus have time complexity $O(Nk \log(N))$. Unfortunately k also grows somewhat with N, and it therefore becomes necessary limit the amount of candidates

[2] We did also some preliminary experiments using the morph lexicon, but it appeared not to improve the results, so it was discarded over the simpler implementation of considering only words in the word lexicon.

considered per word. We limit it by a parameter K. To alleviate the effect of this limitation, we try to consider the best candidates first, so that worse ones are eliminated.

Our exact search for candidate base forms is the following. For virtual morph μ:

- If length(μ) < 4 don't consider any base forms
- If $4 <$ length(μ) < 6, search for candidates that share prefix $\mu_{0..\text{length}(\mu)-3}$
- If length(μ) > 6, search candidates that share prefix $w_{0..\text{length}(\mu)-4}$

Then for each $v_{\text{pre}} = \mu_{0..(i-1)}, v_{\text{suf}} = \mu_{i..(\text{length}(\mu)-1)}$:

- If length(v_{suf}) ≤ 5 and v_{suf} exists in the lexicon and we have considered less than K candidates for virtual morph μ: Consider candidates one at the time, starting with those of the same length as stm, and working towards shorter ones, but never shorter than length(v_{pre})-2. Calculate the needed mutation, and calculate model probability.
- Otherwise don't consider any base forms

In one epoch we go through each word in the corpus in random order. The process is repeated until the model probability stops improving significantly. In practice, 2–6 epochs are usually sufficient.

3 Experiments

The model was evaluated in Morpho Challenge 2008 competition. Here we describe the datasets, the evaluation method, and the results.

3.1 Datasets

We trained our model on four of the datasets provided in Morpho Challenge 2008: English, German, Turkish and Finnish. Table 1 shows the sizes of the sets. Further details of the datasets are found from the web page[3] of the challenge.

Table 1: Sizes of the datasets used in training.

Language	*Word tokens*	*Word types*
English	62 185 728	384 903
Finnish	36 207 308	2 206 719
German	46 338 213	1 266 159
Turkish	12 862 393	617 298

3.2 Evaluation

The method was evaluated in Competition 1 of the Morpho Challenge 2008. The task was to return a list where each word form in the training data was followed by its analysis. This analysis was compared to linguistic gold standard. In the gold standard analyses we have analyses such as `flying FLY_V +PCP1`, where `FLY_V` is the base form and `+PCP1` refers to an inflectional morpheme. As the labels of the morphemes are not relevant (or even possible to obtain for an unsupervised algorithm), the closeness of the gold standard was evaluated by taking word pairs which share a morpheme in one analysis (either gold standard or the submitted one) and checking if they share some morpheme also in the other. Details are provided in Morpho Challenge web page and [7].

[3] `http://www.cis.hut.fi/morphochallenge2008`

3.3 Model Parameters

Although our algorithm is mainly unsupervised, some general prior parameters must be set by hand. As parameters for the morph lexicon length distribution in Equation 10 we used for shape $a = 5$ and scale $b = 1$, which expresses the prior belief that morphs in the lexicon are most likely to be 5 characters long. We had to modify the model search method to the one described in Section 2.6, and therefore we were only able to train models using substitution-deletion mutations, and not on morph-end mutations. The parameter K denoting the number of candidates considered at for each virtual morph was set to 20. For the mutations in Equation 15, we used parameters $a = 1$ and $b = 1$ of the gamma length prior to model prefer short mutations, and the probability deletions and insertions in Equation 16 was set to 0.5.

3.4 Results

The results provided by the Morpho Challenge organizers are summarized in Table 2. It can be seen that our precision is quite good, especially for Turkish and Finnish. However, the recall is very poor. Low recall numbers mean that the model undersegments heavily, i.e., most of the relevant morphemes are missing from the analysis.

Table 2: Results of the Morpho Challenge evaluation. Precision measures how well of the morphemes found by the algorithm match the ones in gold standard analysis. Recall measures how extensively the gold standard morphemes are found by the algorithm. F-measure is the harmonic mean of precision and recall.

Language	*Precision*	*Recall*	*F-Measure*
English	83.39%	13.43%	23.13%
German	87.92%	7.44%	13.71%
Turkish	93.25%	6.15%	11.53%
Finnish	92.55%	6.89%	12.82%

Mutations were not used very frequently in the analyses. E.g., of the cases where substructure was found in the word form, 98% of the mutations were empty for English and 96% for Finnish. The five most common mutations for English and Finnish are given in Table 3, with one typical example analysis for each. Part of the usages are desired (e.g., `-e` in `abjure`, `-a` in `haljeta`), but there are many cases where the mutation is clearly unnecessary. E.g., a simpler analysis for `suspicions` would be `suspicion () s`. Mutations are also used quite commonly in misspelled words. E.g., both `contructed` and `contructive` exist in the English corpus, and mutation `-d-e` is used to get the missing base form `contruct`.

Table 3: The five most frequent mutations found by the algorithm for English (left side) and Finnish (right side).

Mutation	*Freq.*	*Example*	*Mutation*	*Freq.*	*Example*
`-e`	2033	abjure (-e) ed	`(-n)`	27510	antiikin (-n) lle
`-s`	537	actress (-s) s'	`(-n-e)`	15830	edustajien (-n-e) esi
`-y`	386	inequity (-y) able	`(-a)`	6241	haljeta (-a) essa
`-n`	243	suspicion (-n) ns	`(-i)`	4203	kliimaksi (-i) in
`-d-e`	183	contructed (-d-e) ive	`(-a-t)`	2792	alokkaita (-a-t) lle

4 Discussion and Future Work

As our model suffers from undersegmentation, we need to address that problem first. In practice, undersegmentation means that the cost of the substructure for a word is too large compared to direct coding as letters. The reason might be an unwanted bias in the model probabilities, or just an error in the implementation. Also the search algorithm of the model is currently quite heuristic, and a more principled approach needs to be developed. To improve the greedy optimization, we are considering learning procedures based on simulated annealing.

The model described in this paper uses no global information at all. In contrast, Dasgupta and Ng [5] extract orthographic rules in a global fashion, and Creutz and Lagus [4] use perplexity to classify morphs into stems, prefixes and suffixes. There may be a need for including this kind of information into the training procedure.

It is clear that the correct morphological analysis, or even segmentation, of a word is dependent on its word-level context. (E.g., `thought` can be either an inflected verb or a noun.) At some stage, we want to incorporate sentence and word analysis to the model. I.e., instead of setting the word-level parts in Equation 2 to constants, we may use them to model the context. Especially useful may be combining part-of-speech tagging and morphological analysis: On one hand, if POS of the word is known, the possible morphemes are restricted. On the other hand, if our analysis contain a certain suffix that occurs only in, e.g., verbs, selecting the POS for the word is trivial.

We conclude that despite the current method's disappointing results, we find the general approach to be promising and the problem worth further research.

References

[1] Delphine Bernhard. Simple morpheme labelling in unsupervised morpheme analysis. In *Working notes for the CLEF 2007 Workshop*, Budapest, Hungary, 2007.

[2] Mathias Creutz, Teemu Hirsimäki, Mikko Kurimo, Antti Puurula, Janne Pylkkönen, Vesa Siivola, Matti Varjokallio, Ebru Arisoy, Murat Saraçlar, and Andreas Stolcke. Morph-based speech recognition and modeling of out-of-vocabulary words across languages. *ACM Transactions on Speech and Language Processing*, 5(1):1–29, 2007.

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

[4] Mathias Creutz and Krista Lagus. Unsupervised models for morpheme segmentation and morphology learning. *ACM Transactions on Speech and Language Processing*, 4(1), January 2007.

[5] Sajib Dasgupta and Vincent Ng. High-performance, language-independent morphological segmentation. In *In the annual conference of the North American Chapter of the Association for Computational Linguistics (NAACL-HLT)*, 2007.

[6] C. G. de Marcken. *Unsupervised Language Acquisition*. PhD thesis, MIT, 1996.

[7] Mikko Kurimo, Mathias Creutz, and Ville Turunen. Overview of Morpho Challenge in CLEF 2007. In *Working notes for the CLEF 2007 Workshop*, Budapest, Hungary, 2007.

[8] Peter H. Matthews. *Morphology (Cambridge Textbooks in Linguistics), 2nd edition*. Cambridge University Press, 1991.

[9] Gonzalo Navarro. A guided tour to approximate string matching. *ACM Comput. Surv.*, 33(1):31–88, 2001.