A Lexeme-Clustering Algorithm for Unsupervised Learning of Morphology

Maciej Janicki
macjan@o2.pl
University of Leipzig, Master’s programme in Computer Science

Abstract: This paper presents an algorithm for grouping morphologically related words, i.e. different inflected forms of the same base word, for the purpose of unsupervised learning of a natural language morphology. Initially, a trie of words is built and each node in the trie is considered a candidate for stem. The suffixes, with which it occurs, are clustered according to mutual information in order to identify inflectional paradigms. The algorithm has been implemented and evaluated for Polish, German, Finnish and Latin testing sets. Applications and topics for further research have been pointed out.

1 Introduction

Morphology is the field of linguistics studying the shape of words, or how words are built from the smallest meaningful components (morphemes). With morphology of a given language, we mean the rules of forming words in that language and the relations between words in terms of morphemes used. There exist numerous textbooks for morphology. In this paper, [Has02] was used as a linguistic background.

With unsupervised learning of morphology, we mean computer algorithms, that try to derive patterns of word formation and identify morphologically related words using as little data as a plain wordlist or an unannotated corpus, without any knowledge of the language itself. Though its results are still insufficient for practical applications [HB11], developing this field has important implications, both practical – the possibility of building morphological parsers working for many languages with low cost, and theoretical – giving us knowledge about how much of a natural language grammar can be inferred out of plain textual data [Gol06].

This paper is an attempt to solve one of the subproblems of unsupervised learning of morphology: grouping words into lexemes. The implementation of the presented algorithm, as well as the data used for evaluation, are publicly available1.

1https://bitbucket.org/macjan/automorph/
1.1 Basic definitions

A group of words, which have essentially the same meaning, but occur in different grammatical contexts, is called a lexeme, e.g. \{sing, sings, singing, sang, sung\}. Most often, lexemes (rather than all possible words) are the units described in dictionaries.

There are two important morphological relations between words: inflection and derivation. Inflection refers to the rules of forming words, which belong to the same lexeme, like adding the English 3rd person singular suffix -s or the progressive suffix -ing to a verb. Derivation means forming new lexemes, of which both the meaning and the phonological form is derived from the old lexeme, like adding in English the suffix -er to the verb sing to form a new noun lexeme singer. Since the borders between inflection and derivation are not always sharp, in this paper we’ll assume, that inflection is an operation, that doesn’t change the word’s part of speech (e.g. verb to noun), while derivation does.

Inflection and derivation can be performed with morphological operations, which include affixation and alternation. Both operate on the stem – the morpheme, that is common to all the words belonging to one lexeme and that carries the meaning of the lexeme. Affixation means adding a new morpheme (affix) to the stem. Depending on its position, the affix can be called prefix (before the stem, e.g. English re-make), suffix (after the stem, e.g. English sing-s), circumfix (before and after the stem, e.g. German ge-mach-t), or infix (inside the stem). Alternation means the change inside the stem, like English sing : sang, or German umlauting: Vater : Väter.

1.2 Goals of this paper

One of the approaches to unsupervised learning of morphology is the so-called Group and Abstract approach [HB11]. It consists of two tasks: in the first, the words are grouped in clusters according to some measure, which should yield groups of morphologically related words. The second task is to generalize patterns, that are followed by many clusters, into morphological rules.

This paper presents an algorithm, which performs the first task of the Group and Abstract approach. It gets a plain list of words as input and returns a list of lexemes. The algorithm is targeted to work on European languages with suffix-based morphologies with various degrees of complexity. The possibility to apply a similar approach to different types of morphology is left open, but requires some additional research, which is described at the end of the paper.

1.3 Related work

Much work done on the field of unsupervised learning of morphology is described in the survey article [HB11], ranging from the first works in 1960s up to present. Recently, a
lot of research has been done in this field, among which the most attention is given to
the methods based on finding the segmentation into morphemes by means of frequency
or information-theoretical measures, like [Gol06]. Various methods for lexeme clustering
are discussed in [YW00], along with the possibility to combine them. A semantics-based
approach is presented in [SJ00].

2 The algorithm

The algorithm for lexeme clustering consists of four steps. In the first, we analyze the
frequency of n-grams occurring word-finally, taking those with high frequency as possible
candidates for suffixes. The second step is transforming the wordlist into a trie structure,
in which we type candidates for stems and their suffix sets. In the third step, suffix sets are
clustered into morphological paradigms using the mutual information measure. Finally,
for each word, morphologically-related words are extracted from the trie.

2.1 Suffix filtering

Before we start the actual algorithm, we can easily filter out n-grams, that occur at word-
final positions, but are certainly not morphological suffixes. We will make use of the
following assumption:

Assumption 2.1 Morphological suffixes need to occur with a variety of stems, i.e. their
frequency must be of significant size.

Experiments show, that all the real morphological suffixes in all tested languages have
a frequency greater than 0.001 among n-grams in word-final positions (we measure the
frequency separately for each n, i.e. separately for unigrams, bigrams and so on). By
filtering out n-grams below that frequency, we get rid of about 99.8% word-final n-grams
(for example, out of 530498 endings of length ≤ 9 found in Polish data set, only 773 have
the frequency above 0.001). We will consider the remaining n-grams as candidates for
morphological suffixes.

2.2 Trie with suffix sets

After filtering out improbable suffixes, we build up a trie of words. Tries are well-known
data structures, that have already been successfully used in unsupervised learning of mor-
phology (cf. for example [SJ00], [Gol06]). Trie is a tree data structure, in which each node
represents the set of words, which share a certain common prefix. The children of the node
correspond to prefixes, which are longer by one letter, than the prefix of their parent, so
that moving from parent to child in the trie means moving one letter forward in a word.
In every node of the trie, we will also store the suffix set of its prefix, i.e. the set of suffixes which can be concatenated to the prefix to form a word, that appears in the wordlist. For example, if the word *machen* appears in the wordlist, the ending *-en* should be included in the suffix set of *mach-* and so should *-n* for *mache-* and Ø (empty ending) for the node containing the prefix *machen-* . However, in suffix sets, we include only endings, that were not filtered out in the previous step. An example fragment of a trie with suffix sets, containing the German words { *machen*, *mache*, *machst*, *macht*, *machbar*, *machbare*, *machbaren*, *machbarer* }, is shown in Figure 1.

Because we don’t know where the actual border between stem and suffix is, we can consider every prefix occurring in some node of a trie a candidate for stem and its suffix set members candidates for morphological suffixes.

### 2.3 Partitioning suffix sets

In each node of a trie, we have a candidate stem and a corresponding suffix set. It’s easy to see, that some of the suffix sets will yield words belonging to different lexemes, like the set { *-en*, *-e*, *-st*, *-t*, *-bar*, *-bare*, *-baren*, *-barer* } for the stem *mach-* in German (see Figure 1). Therefore, there arises a need to partition suffix sets into clusters, so that suffixes from every cluster will yield words belonging to the same lexeme, like { *-en*, *-e*, *-st*, *-t* , *-bar*, *-bare*, *-baren*, *-barer* } . Some of such clusters, like { *-en*, *-e*, *-st*, *-t* ,

---

2For simplicity, only a few possible endings were shown.
immediately are morphological paradigms. Others, like {-bar, -bare, -baren, -barer}, lead to a morphological paradigm {-ø, -e, -en, -er} lower in the trie. With such partitioning of the suffix set, we can group words sharing a node in the trie into different lexemes.

We will use the mutual information measure to measure dependencies between subsets of a suffix set. Mutual information of two random variables is the amount of information, that they share. For random variables \(X, Y\) taking values from the sets \(\mathcal{X}, \mathcal{Y}\), respectively, it is defined as [CT91]:

\[
I(X; Y) = \sum_{x \in \mathcal{X}} \sum_{y \in \mathcal{Y}} p(x, y) \log \frac{p(x, y)}{p(x)p(y)}
\]  

(1)

The following assumption will allow us to apply mutual information for suffix set partitioning:

**Assumption 2.2** Parts of the same morphological paradigm share a lot of information, i.e. are highly dependent on each other, while two different morphological paradigms are nearly independent, i.e. the mutual information among them is low.

Using this assumption, we will define the partitioning of the suffix set, that we want to achieve. To each set of suffixes \(S\), we can assign a random variable \(X_S\), telling us what the probability of finding some subset of \(S\) with a random stem is. More precisely, let the elements of \(S\) be indexed as follows: \(S = \{s_1, s_2, \ldots, s_n\}\). Then, the values of the variable \(X_S\) are vectors \(v = v_1v_2\ldots v_n \in \{0,1\}^n\), where for each \(v_i\), the value 1 represents presence of the corresponding suffix \(s_i\), while the value 0 represents absence of this suffix.

Thus, \(p(v)\) is defined as the probability, that for a randomly chosen suffix set \(R\) holds:

\[
\forall i \in \{1, \ldots, n\} (s_i \in R \iff v_i = 1).
\]

The entropy of the variable \(X_S\) corresponds to the amount of information contained in knowing, which suffixes of \(S\) some stem takes or does not take. The mutual information \(I(X_S; X_T)\) between variables corresponding to different suffix sets means the amount of information, by which knowing which suffixes of \(S\) some stem takes helps to determine which suffixes of \(T\) it should take and reverse. In the further part we will use the notion \(I(S; T)\) for \(I(X_S; X_T)\). Note, that for sets of suffixes \(S, T, U\) and their random variables \(X_S, X_T, X_U\), holds: \(I(S \cup T; U) = I(X_S, X_T; X_U) = I(X_S; X_T; X_U)\).

Let’s now define the target partitioning:

**Definition 2.1** Let \(S\) be a suffix set (belonging to some stem), \(\mathcal{S} = \{S_1, S_2, \ldots, S_n\}\) a partitioning of this set and \(\kappa\) a fixed constant. We call \(\mathcal{S}\) a partitioning into paradigms, iff:

1. for each \(i \neq j\): \(I(S_i, S_j) < \kappa\)
2. for each disjoint \(A, B\): \(A \cup B = S_i \Rightarrow I(A, B) \geq \kappa\)

Those two conditions represent the statements of assumption 2.2: the mutual information is low between two different paradigms and high between two parts of the same paradigm.
Data: a suffix set $S = \{s_1, s_2, \ldots, s_n\}$
Result: $\mathcal{S} = \{S_1, S_2, \ldots, S_k\}$, being the partitioning of $S$ into paradigms
$\mathcal{S} \leftarrow \{\{s_1\}, \{s_2\}, \ldots, \{s_n\}\}$;
$k \leftarrow n$;
while true do
    $i_{\text{max}}, j_{\text{max}} \leftarrow 1, 1$;
    for $i \leftarrow 1$ to $k$ do
        for $j \leftarrow 1$ to $k$ do
            if $I(S_i, S_j) > I(S_{i_{\text{max}}}, S_{j_{\text{max}}})$ then
                $i_{\text{max}} \leftarrow i$;
                $j_{\text{max}} \leftarrow j$;
            end
        end
        if $I(S_{i_{\text{max}}}, S_{j_{\text{max}}}) \geq \kappa$ then
            $S_{i_{\text{max}}} \leftarrow S_{i_{\text{max}}} \cup S_{j_{\text{max}}}$;
            $\mathcal{S} \leftarrow \mathcal{S} \setminus \{S_{j_{\text{max}}}\}$;
            $k \leftarrow k - 1$;
        else
            terminate and return $\mathcal{S}$;
        end
    end
end

Algorithm 2.1: Computing the partitioning into paradigms

The $\kappa$ parameter represents the maximum mutual information of two independent inflectional paradigms (we will call it independence threshold) and has to be given at input. Formally, if two random variables are independent, then their mutual information should be 0, but also different paradigms are never completely independent. Some paradigms tend to occur together with others (e.g. verb with a corresponding deverbal noun), other exclude each other (e.g. two different verb conjunction paradigms).

The algorithm 2.1 is used to compute the partitioning into paradigms. Initially, each suffix is made a distinct part of the partitioning. Then, in a loop, the parts, which share the highest amount of information are merged. The algorithm terminates when the highest dependency among some two parts is lower than the independency threshold, which means, that all parts are already independent. The following theorem will be used to prove the correctness of the algorithm:

**Theorem 2.1** The algorithm 2.1 computes the partitioning into paradigms, as defined in the definition 2.1.

**Proof.** The low dependency between member sets of $\mathcal{S}$ is obvious: the algorithm terminates exactly when there is no more pair of sets with mutual information $\geq \kappa$.

We will now show the high dependency inside parts. Let $S_i \in \mathcal{S}$ and $A, B$ be disjoint subsets of $S_i$, for which holds: $A \cup B = S_i$. At some iteration of the algorithm, there must
have been such \( C \subseteq A, D \subseteq B \), that \( C \) and \( D \) have been merged, which means \( I(C;D) \geq \kappa \).

Let \( A' = A \setminus C \) and \( B' = B \setminus D \).

Using the chain rule for mutual information [CT91] and the non-negativity of mutual information, we obtain:

\[
I(A;B) = I(C \cup A';D \cup B') = I(C;D \cup B') + I(A';D \cup B'|C) \\
= I(C;D) + I(C;B'|D) + I(A';D \cup B'|C) \geq I(C;D) \geq \kappa
\]

\(\square\)

**Optimization**  
Because the algorithm 2.1 requires computing mutual information many times, which takes a very long time (iterating over all stems is necessary each time mutual information is computed), some optimizations had to be performed. They include:

- storing mutual information between each pair of parts in a matrix and calculating new values only there, where some sets were merged;
- merging more than one pair of sets in a single iteration: if \( S_i \) and \( S_j \) are merged, one can merge \( S_k \) and \( S_l \) in the same iteration, provided that \( I(S_k,S_l) \) is greater than each of the values: \( I(S_k,S_i \cup S_j), I(S_j,S_i \cup S_j) \), because then merging \( S_i \) and \( S_j \) will not affect the behaviour of \( S_k \) and \( S_l \);
- storing entropies of all sets seen before in a hash table and computing mutual information from the formula [CT91]:

\[
I(X;Y) = H(X) + H(Y) - H(X,Y)
\]

Note also, that for a set of suffixes \( S \) and its random variable \( X_S \), computing \( H(X_S) \) out of the definition would require iterating over all possible values of \( X_S \), which is \( 2^{|S|} \). Instead, we iterate over all stems in the trie and count all subsets of \( S \), that we find in suffix sets. Non-occurrent values don’t change the entropy, so they don’t have to be included in computation.

### 2.4 Extracting lexemes

The last step of the algorithm is extracting lexemes from a trie with partitioned suffix sets. Given a word split into stem and suffix, we can look at the node of the trie corresponding to the stem and the part of its suffix set corresponding to the suffix, to find other possible morphological suffixes of the same paradigm, which can occur with the same stem, yielding other words belonging to the same lexeme. There is one problem with this approach: we don’t know where the border between stem and suffix is. However, we can iterate over all possible stem-suffix pairs for a given word. The intuition lying behind this approach can be expressed in the following assumption, which turned out to work in practice:
Assumption 2.3 If two words \( w_1, w_2 \) belong to different lexemes, at each level of the trie where they share the stem, their endings will belong to different parts of the suffix set.

Formally, for a word \( w \), let \( \sigma(w) := \{ (r, s) : |r| > 0 \land w = rs \} \) denote the set of splittings of the word \( w \), i.e. possible stem-suffix pairs. Let \( \mathcal{S}_r = \{ S_{r,1}, S_{r,2}, \ldots, S_{r,n} \} \) denote the partitioned suffix set corresponding to stem \( r \). Then the set \( \text{lex}(w) \), containing all words belonging to the same lexeme as \( w \), is computed as follows:

\[
\text{lex}(w) = \bigcup_{(r,s) \in \sigma(w)} \{ rs' : \exists \{ s, s' \} \subseteq S_{r,i} \}\]

(3)

The assumption 2.3 ensures, that we will not overgenerate the lexeme. Note that the nodes in trie, where we pick new words to the lexeme, are likely to be morpheme boundaries. Perhaps one could use the same trie to determine the segmentation into morphemes, which would be an important step towards abstracting morphological paradigms from lexemes.

3 Evaluation

For the purpose of evaluation, testing datasets for four languages were established. Each dataset contains a list of lexemes, which is expected at the output of the algorithm (reference data). The corresponding input data can easily be obtained by splitting the list of lexemes into a list of words.

The datasets for Finnish, German and Latin were extracted from Wiktionary. The pages with inflected word forms usually contain a template of a form:

```
{{form-of|base-form|inflectional-data}}
```

(or similar), from which base forms were extracted and words with the same base form were grouped into lexemes. English Wiktionary\(^3\) has been used for Latin and Finnish, while German Wiktionary\(^4\) has been used for German.

As Polish dataset, the List of inflected words of a freeware Polish language dictionary\(^5\) was used. The list was slightly modified: words containing non-letter characters (such as apostrophes or hyphens) were removed and adjective forms with prefix nie- (eng. un-) were separated as different lexemes, since their relation to the original adjective is derivational, not inflectional [Has02].

Note that not all existing forms of a given lexeme occur in the datasets, because not all of them appear in Wiktionaries. Let’s call the number of forms of a given lexeme appearing in the data divided by the number of all possible forms of this lexeme, lexeme coverage. The average lexeme coverage in the datasets is likely to be higher than that of a corpus\(^6\).

---

\(^3\)http://en.wiktionary.org/
\(^4\)http://de.wiktionary.org/
\(^5\)http://www.sjp.pl/słownik/odmiany/
\(^6\)See [HB11], section 3.2.3 for more detailed remarks about lexeme coverage in corpora.
The evaluation was performed as follows: for each word, its lexeme was computed and matched against the lexeme in the reference data. Matches (TP), overgenerated words (FP) and undergenerated words (FN) were counted. These results were summed up for all the words in the wordlist and used to calculate Precision, Recall and F-measure as follows:

\[
\text{Precision} = \frac{TP}{TP + FP} \quad (4)
\]
\[
\text{Recall} = \frac{TP}{TP + FN} \quad (5)
\]
\[
\text{F-measure} = \frac{2 \cdot TP}{2 \cdot TP + FP + FN} \quad (6)
\]

The optimal \( \kappa \) values were determined experimentally for each dataset and correspond to the maximal mutual information value between two different paradigms observed in this dataset. It is not clear, whether optimal \( \kappa \) values could be automatically derived from datasets.

For Latin data set, three tests were performed: one only for declension (nouns and adjectives, without verbs) and two for the full data, with two different \( \kappa \) values. For every other data set, a single test was performed.

Table 1 shows the results of the evaluation. Additional information about the size of testing sets and the value of the \( \kappa \) parameter were included. The full Latin set turned out to be most challenging, with results strongly depending on the \( \kappa \) value. The reason is, that Latin verbal stems take a great number of endings (often over 100) and it’s hard to decide, which of them are derivational and which inflectional (e.g. participles).

<table>
<thead>
<tr>
<th>Testing set</th>
<th>Words</th>
<th>Lexemes</th>
<th>( \kappa )</th>
<th>Precision</th>
<th>Recall</th>
<th>F-measure</th>
</tr>
</thead>
<tbody>
<tr>
<td>Finnish</td>
<td>9k</td>
<td>1k</td>
<td>0.002</td>
<td>98.6 %</td>
<td>81.3 %</td>
<td>89.1 %</td>
</tr>
<tr>
<td>German</td>
<td>54k</td>
<td>20k</td>
<td>0.01</td>
<td>97.7 %</td>
<td>79.1 %</td>
<td>87.4 %</td>
</tr>
<tr>
<td>Polish</td>
<td>980k</td>
<td>61k</td>
<td>0.002</td>
<td>96.1 %</td>
<td>79.0 %</td>
<td>86.7 %</td>
</tr>
<tr>
<td>Latin (decl.)</td>
<td>210k</td>
<td>22k</td>
<td>0.05</td>
<td>95.4 %</td>
<td>92.4 %</td>
<td>93.9 %</td>
</tr>
<tr>
<td>Latin (full)</td>
<td>586k</td>
<td>26k</td>
<td>0.015</td>
<td>85.7 %</td>
<td>41.3 %</td>
<td>55.8 %</td>
</tr>
<tr>
<td>Latin (full)</td>
<td>586k</td>
<td>26k</td>
<td>0.02</td>
<td>67.1 %</td>
<td>81.7 %</td>
<td>73.7 %</td>
</tr>
</tbody>
</table>

(because we extract forms from a dictionary), but still far from 1.0. The testing data sets are not very different from real corpora in this feature. However, for the purpose of unsupervised learning of morphology, the lexeme coverage in input data should be as high as possible.

Table 1: Evaluation results (\( k = 1000 \)).
4 Conclusion

The results show, that the mutual information measure defined on sets of affixes is a good means to discover morphologically related words. This method can be further explored in order to abstract morphological paradigms and thus provide a complete algorithm for unsupervised learning of morphology.

4.1 Problems and further work

The major problem of the algorithm is that it takes a very simple view on morphology: it treats words as concatenations of a stem and a single suffix\(^7\). Even in the four tested languages there are cases, where this approach is insufficient:

- alternations inside stem, e.g. German umlauting: \textit{Haus-}ø : \textit{Häus-}er
- prefixing, e.g. German \textit{mach-en} : \textit{ge-mach-t}
- multiple affixes, e.g. Polish \textit{mówi-ć} (to speak) : \textit{mówi-li-by-śmy} (we would speak)
- affixes shared across multiple paradigms, e.g. Latin \textit{Aureli-is} : \textit{Aureli-}us vs \textit{Aureli-is} : \textit{Aureli-}a (in our approach the suffix set is partitioned into disjoint paradigms)

Although the evaluation results, especially for Finnish with its multi-slot morphology, have shown, that treating multiple suffixes as a single one (a concatenation of those) works well enough, this behaviour still needs to be justified.

Dealing with prefixing and alternations is to be subject of further work. Let us observe, that this would significantly improve the results (more specifically, recall), which are already high. That lets us expect, that some additional research could make this algorithm achieve much better results, possibly above 90% of F-measure.

Another problem is the presence of the \(\kappa\) parameter. The ideal approach requires, that no other data than a list of words need to be supplied. The behaviour of \(\kappa\), e.g. its dependency on language, and the possibilities to compute a near-to-optimal value out of the wordlist, should be examined. However, it’s only one parameter and its meaning is well defined.

The approach of determining morphological classes only out of surface forms has also its limitations. For instance, there are cases in Latin, where one stem can be shared by a masculine and a feminine noun, like name pairs \textit{Aurelius} : \textit{Aurelia}. Such pairs, which should be classified as distinct lexemes, are hard or impossible to distinguish from adjectives, which can take both masculine and feminine endings, like \textit{altus} : \textit{alta}.

\(^7\)As it is also done in some other approaches, for example [Gol06].
4.2 Applications

As mentioned before, the algorithm was developed for the purpose of unsupervised learning of morphology. The next task would be to find or develop a suitable method to abstract morphological paradigms from lexemes. The final result would then be a general method to construct morphological analyzers and morphology tables for many languages with a little effort.

Clustering lexemes can also have applications on its own. For example, it can be used in simple corpus search engines to augment search patterns with inflected forms of words. A clustered wordlist can also be used for language-statistical purposes: statistical measures over number of inflected forms of a lexeme, suffix length etc. could be interesting for linguists.

References


