A Segmentation Approach to Morpheme Analysis

Emily Pitler
Yale University
New Haven, CT 06520
emily.pitler@aya.yale.edu

Samarth Keshava
Yale University
New Haven, CT 06520
samarth.keshava@aya.yale.edu

Abstract

We present a simple algorithm that is also psychologically plausible to perform unsupervised learning of morphemes. The algorithm is most suited to Indo-European languages with a concatenative morphology, and in particular English. We will describe the two approaches that work together to detect morphemes: 1) finding words that appear as substrings of other words, and 2) detecting changes in transitional probabilities. This algorithm, while most suited to the task of segmenting words into morphemes, also suffices to analyze morphemes.

1 Introduction

This paper addresses the problem of segmenting natural language words into morphemes, the smallest units of language that still contain meaning. While one cannot extract meanings from lists of words and their frequencies, we can nevertheless still use statistical information to make useful predictions about likely morphemes.

There is a large body of literature on morpheme induction, and while it is impossible to give a complete survey, see (Goldsmith, 2001) for a good summary of previous approaches. Goldsmith divides these past attempts into four categories: identification of morpheme boundaries using transitional probabilities; identification of morpheme-internal bigrams or trigrams; discovery of relationships between pairs of words; and an information-theoretic approach to minimize the number of letters in the morphemes of the language. Our work combines ideas from several of these approaches and does not fit neatly into any one of the categories.

The key idea in this paper is to use words that appear as substrings of other words and transitional probabilities together to detect morpheme boundaries. The first approach derives from the observation that, often, the stem left over after removing prefixes and suffixes is a legitimate word itself. Due to spelling changes, this is not always the case and therefore this method should not be used to actually segment a word. However, given a large enough corpus, the most common morphemes can be found in this way. The other idea, using transitional probabilities, was initially presented by (Harris, 1955). Given an utterance, Harris proposed finding how many other utterances in the corpus shared each starting fragment of that utterance. He hypothesized then that peaks in these counts correspond to morpheme boundaries.

(Hafer and Weiss, 1974) further developed the ideas presented in Harris's paper. Using Harris's transitional probability technique as a starting point, Hafer and Weiss created 15 different algorithms that achieved various levels of precision and recall. One issue however with their approach is their heavy reliance on empirically determined parameters. For example, their best algorithm (with a precision of 91.0% and a recall of 61.0%) posited a morpheme boundary if the suffix is a word and the predecessor count is at least 5, or if the predecessor count is at least 17 and the successor count is at least 2.

Our goal was to design a simple algorithm given our intuition that simpler algorithms are more likely to approximate human processes. We consciously limited both the number of language-specific assumptions that our program makes and "magic numbers", parameters arbitrarily tuned to make the program work. We did not limit the length of morphemes, the number of morphemes per word, nor the total number of morphemes.

2 Methodology

Our algorithm has four basic steps. We

1. build trees with probabilities based on the corpus,
2. score word fragments using these trees to obtain a large list of morphemes,
3. prune this list of morphemes, and
4. segment the test words using the morpheme list and the lexicographic trees.

Each of these steps is described in further detail below.

2.1 Building the Lexicographic Trees

At the beginning of the algorithm, we create two trees of letters and their associated counts: the "forward tree" and "backward tree". We explain here the construction of the "forward tree" (the other construction is symmetric). Suppose the alphabet of the language has b letters, and the longest word in the corpus consists of d letters. Then conceptually, we construct a complete b-way tree with depth d. At each node, each of the b branches represents one of the letters in the language. Thus, any path from the root to some node spells out the starting fragment of some word(s), and the node itself contains the frequency of that string. (Note that in practice, actually creating such a tree would be prohibitive as well as wasteful since most letter combinations never occur; thus we actually only store nodes with non-zero counts.)

The forward and backward trees allow us to calculate conditional probabilities in O(1) time given a starting or ending substring of a word. For example, we would use the forward tree to calculate $Pr_f(\text{s}|\text{report})$ (by dividing the frequency of words starting with "reports" by the frequency of words starting with "report"). In the opposite direction, we would use the backward tree to calculate $Pr_b(\text{e}|\text{ports})$ (by dividing the frequency of words ending in "eports" by the frequency of words ending in "ports").

2.2 Scoring Potential Morphemes

Once we have finished constructing the trees as described above, we begin finding morphemes. We maintain two lists of morphemes: a prefix list and a suffix list.[1] To populate the suffix list, for each word, we scan from the end of the word and consider every possible suffix in order of increasing length. Suppose we are considering the suffix $B\beta$ in the word $\alpha AB\beta$. We hypothesize the proposed suffix is correct if

1. αA is also a word in the corpus,
2. $Pr_f(A|\alpha) \approx 1$, and
3. $Pr_f(B|\alpha A) < 1$.

Similarly, the criteria for determining if αA is a prefix in the word $\alpha AB\beta$ is as follows

1. $B\beta$ is also a word in the corpus,
2. $Pr_b(B|\beta) \approx 1$, and
3. $Pr_b(A|B\beta) < 1$.

The first criterion corresponds to the observation that prefixes and suffixes are often added on to root words. For example, after removing the suffix "ed" from "corresponded", the resulting fragment "correspond" is still a word. The second and third criteria are checked using the forward and backward trees. They check that the stem has multiple children (thus implying other prefixes or suffixes can be joined to the stem) but that the stem's parent has only one child (thus identifying it as a true stem). Using the same example as before, the algorithm would check that $Pr_f(\text{d}|\text{correspon}) \approx 1$, and that $Pr_f(\text{e}|\text{correspond}) < 1$. If a given morpheme passes all three tests, we increase its score by 19 points; otherwise, we decrease its score by 1. After we have iterated through the entire corpus, we consider all strings with positive scores morphemes.

The rule of rewarding word fragments by 19 and punishing by 1 may seem arbitrary, but the constants were chosen so that a string has a positive final score only if it passed our tests at least 5% ($= \frac{1}{1+19}$) of the times it appeared. Moreover, the numbers 19 and 1 are not special; any positive numbers x and y such that $\frac{y}{x+y} = .05$ would produce identical results.[2] The rewarding and

[1]We use the terms prefix and suffix loosely, to denote any morpheme generally found at the beginning or end of words. For example, "man" is not technically a suffix, but it is a morpheme that often appears at the end of a word.

[2]Suppose that we rewarded and punished by $x > 0$ and $y > 0$ respectively, satisfying $y/(x + y) = 0.05$. Then

punishing scheme is more effective than checking the percentage of tests passed because given two morphemes with the same percentage, the more common morpheme will have a higher score. Thus, the punishing/rewarding scheme takes into account both the reliability and the frequency of the string appearing as a morpheme. Single letters such as ‘t’, which sometimes deceivingly appear to be prefixes, are punished far more often than they are rewarded. Strings such as ‘psycho’, which do not appear often but are almost always true morphemes when they do appear, are rewarded more often than they are punished. Suffixes like ‘s’ are punished occasionally but rewarded very frequently, and are ranked at the top of the list.

2.3 Pruning

Clearly, this method is not perfect. In particular, one problem that often arises is that the final list of morphemes includes strings that are the concatenation of two other morphemes. For example, the list might include all of ‘er’, ‘s’, and ‘ers’. This is undesirable since the final step of segmenting words may process the word “throwers” as throw+ers instead of as throw+er+s. Fortunately, though, this problem has a relatively simple solution (which we refer to as “pruning”). We scan each list of morphemes, and if any morpheme is composed of two others with better scores, then it is thrown out.

2.4 Segmenting Words

Finally, we come to the actual segmenting of words. Given the list of morphemes, one possible approach is to simply peel morphemes off the ends of words as they are found. But words such as “politeness” pose a problem: should it be segmented as politenes+s or as polite+ness? Neither the scores nor the lengths of morphemes can be reliably used to answer this question. In this case, ‘s’ would have a higher score, while ‘ness’ is a longer morpheme. They key observation is that the same probability criteria that was used earlier to detect morphemes can be applied here to measure the appropriateness of segmenting at a particular position. In this example, we expect $Pr_f(\text{n}|\text{polite})$ to be lower than $Pr_f(\text{s}|\text{politenes})$ which leads to the correct segmentation.

Thus, our method for segmenting is as follows. First, we scan each word from the end, and find all morphemes $B\beta$ from the suffix list such that our word can be written as $\alpha B\beta$ (for some α). The values for $Pr_f(B|\alpha)$ are compared for all of these and the morpheme (if any) with the lowest value smaller than 1 is chosen. If such a morpheme was found, it is removed and the processed is repeated until no more morphemes can be removed. We then repeat the same process, attempting to peel off morphemes in the prefix list from the beginning of the word (using Pr_b instead of Pr_f).

$y = 0.05\,(x + y) \Rightarrow 0.05x = 0.95y \Rightarrow x = 19y$. Thus, if a string is rewarded r times and punished p times, it would have a score of $xr - yp = 19yr - yp = y(19r - p)$, which is exactly y times our score. In particular, a string has a positive score if and only if it had a positive score in our algorithm.

Table 1: Overall evaluation results of *RePortS*

Language	Precision	Recall	F-Score
English	74.73 %	40.62 %	**52.63 %**

Table 2: Gold-standard sample results of *RePortS*

Measurement	Total	Non-affixes	Affixes
Precision	69.72 %	77.21 %	66.33 %
Recall	82.03 %	80.07 %	88.29 %
F-Score	**75.38 %**	78.61 %	75.75 %

3 Results

The algorithm described above was implemented as a Perl program called *RePortS*[3]. The English frequency-word list provided by the Neural Networks Research Centre at the Helsinki University of Technology containing 384,903 words was used for training. To determine the performance of the algorithm, we submitted our proposed segmentations of the corpus to the MorphoChallenge 2007 and we ran our program on a “gold standard” of 484 words (again, provided by the Neural Networks Research Centre) and evaluated our proposed segmentation against the human-determined standard (see Tables 1 and 2).

Our program identified a total of 8573 morphemes (4833 in the prefix list and 3740 in the suffix list). Table 3 contains the ten highest-scoring morphemes from each list.

The program was tested on a dual 2.8 GHz processor with 2 GB of memory. We monitored the total running time, i.e. training and segmentation

[3]The earliest versions of the algorithm determined that the most common prefix, stem and suffix are ‘re’, ‘port’ and ‘s’, respectively; hence, the name *RePortS*.

Table 3: Top English morphemes

Morpheme	Score	Morpheme	Score
non-	19670	s	56416
un	14600	ly	18680
re	10147	-based	9297
anti-	9168	ness	5826
re-	7724	-like	3990
pre-	6423	-style	3127
ex-	5536	ism	2905
self-	5147	al	2831
pro-	5057	-type	2732
over	4290	-led	2566

Table 4: Resource usage for different test data

# Words	Time
484	0m 51sec
467,667	93m 23sec

time of *RePortS*. They are reported in Table 4 for test data of different sizes.

4 Discussion

First, note that the precision is actually higher on the larger sample than on the gold-standard subset, while the recall is significantly lower. One possible explaination is that the large sample contained a large number of hyphenated words. Unlike compound words, where each half often appears in other compound words, words like "we-can-do-business-together" do not have any similar words. There is no "we-can-do-business-alone" or any other "we-can-do-business-"(.*) string that the algorithm can use. Thus, the tree approach does significantly worse when presented with words that are relatively unique, and so the recall is low for the large data set.

This algorithm was designed with the goal of segmentation(Keshava and Pitler, 2006), not analyzation. Thus, the plural "s" and the third person singular verb ending "s" are treated as identical morphemes. "Queens" is segmented as "queen"+"s", rather than analyzed as "queen"+plural; "walks" is segmented as "walk"+"s", rather than "walk"+3SG. When the word is not strictly the concatenation of morphemes, this algorithm does poorly. For example, "boxes" is segmented as "box"+"e"+"s", rather than "box"+plural. A good morpheme analyzer would have to be able to realize that the "s" in "queens" is actually the same as the plural ending "es" in "boxes", and that the verb ending in "walks" is a different type of morpheme.

5 Future Work

As discussed above, the program described in this paper only segments words based on their surface spelling. If part of speech information were used, then the program would be able to detect spelling changes. For example, "box" is a noun, and if it is determined that most of the other nouns can be concatenated with a plural morpheme, and there is an unknown morpheme "es" that does not fit the usual noun paradigm, one could hypothesis that "es" in the word "boxes" is also a plural morpheme. Furthermore, a part-of-speech tagger should not be required for this task. One could use clustering of the affixes to determine categories of words. For example, "ing" is commonly going to be found with the verb stems, but would rarely be found with the noun stems. By iterating the process of determining part-of-speech of the stems and categorizing the affixes into morphemes, a more nuanced morpheme analyzer could be created.

6 Conclusion

We described an efficient algorithm that uses statistical relationships within and between words to predict morpheme boundaries. The algorithm performed very well at segmenting words and performs better than expected on the morpheme analyzation task. We believe that this approach, when combined with a machine learning algorithm to find parts-of-speech, shows promise in decomposing words into their morphological units.

References

John Goldsmith. 2001. Unsupervised Learning of the Morphology of a Natural Language. *Computational Linguistics*, 27(2):153–198.

Margaret A. Hafer and Stephen F. Weiss. 1974. Word Segmentation by Letter Successor Varities. *Information Storage and Retrieval*, 10:371–385.

Z. Harris. 1955. From Phoneme to Morpheme. *Language*, 31(2):190–222.

Samarth Keshava and Emily Pitler. 2006. A simpler, intuitive approach to morpheme induction. *Proceedings of the PASCAL Challenges Workshop*, pages 31–35, April.