CS65: Midterm Report

Bronwyn Woods
Swarthmore College
500 College Ave
Swarthmore, PA 19081
bwoods1@cs.swarthmore.edu

Phil Katz
Swarthmore College
500 College Ave
Swarthmore, PA 19081
katz@cs.swarthmore.edu

Abstract

This paper describes an implementation and extension of algorithms described in Hafer and Weiss (1974) and Déjean (1998) to perform morphological segmentation. The algorithms were evaluated with respect to a gold standard that was hand-tagged by a native speaker specifically for this task. The results were in line with expectations, as they paralleled the results in the original papers, and some novel extensions improved the results.

1 Introduction

This paper presents our implementation of algorithms taken from studies in morphological segmentation (Hafer and Weiss, 1974; Harris, 1955; Harris, 1967). We implemented and extended the algorithms on a 300,000 word corpus generated from the Brown Corpus. We created a "Gold Standard" to help evaluate the accuracy of our morphological segmenters. We implemented some of the algorithm presented in Déjean (1998), but did not fully duplicate all of the results presented therein.

2 Methods

We implemented the algorithms using a combination of Python and Perl. To begin, we pre-processed a list of every word in the Brown Corpus and removed every word that occurred only once, every word that contained one or more capital letters, and every word of length less than three letters. This significantly reduced the percentage of invalid English words in the corpus.

The algorithms that we implemented depended in large part on the calculation of predecessor and successor counts. The successor count of a sequence of letters is the number of letters that follow the given sequence in the corpus. We calculated the successor count for every possible prefix, up to complete words, in the corpus. The predecessor count is the same calculation, only beginning from the end of the word and counting the letters than can precede a substring. Another method discussed in Hafer and Weiss (1974) that we implemented depended on the calculation of entropy for each prefix and suffix in the corpus. Entropy is an information theoretic measure that is calculated based upon the distribution of possible successors or predecessors. For the exact calculation of entropy that we implemented, see Hafer and Weiss (1974).

Hafer and Weiss (1974). describe fifteen algorithms that they employ to segment morphemes using the predecessor and successor counts, and entropy scores, discussed above. We implemented these algorithms as well as a few more with modifications that improved results on our data. The algorithms generally consisted of creating predecessor and successor score vectors for each word in the test corpus, and making cuts based on certain features. For example, a cut could be made if both predecessor and successor scores were over a certain threshold, or if the sum of the two was over a threshold. Other algorithms considered whether the predecessor or successor was a complete word, and made cuts accordingly. Some of the algorithms also

took into account whether the predecessor or successor score was at a "peak or plateau", namely if it was greater than or equal to the surrounding scores. The specifics of each algorithm will be explained in detail in the results section.

We made three major modifications to the algorithms. First, we defined a "peak or plateau" as having successor or predecessor counts not only greater than or equal to the surrounding values but also greater than one. This helped eliminate incorrect cuts in long words where sparse data produced a "plateau" of ones for long sequences at the beginning or end of the word. Second, we optimized the threshold values for each algorithm by exhaustive testing. This made consistent improvements to our results across nearly all algorithms. Finally, in order to correct mistakes where our algorithm segmented two one-letter morphemes at the end of a word, such as "segment-e-d", we explicitly checked for this case and removed the less probable cut.

After implementing these algorithms, we segmented the entire corpus using our best segmenter and used these splits to find the most frequent morphemes. Our method was based on ideas presented in Déjean (1998). We first generated a list of morphemes with counts higher than a certain threshold. For each morpheme, we checked if it was a substring of another morpheme on the list. If the count of the longer morpheme was more than three times that of the smaller, we removed the smaller from the list. Although we did not use this list to segment additional morphemes, it seems likely that we could do so, as the list corresponds to linguistic intution about English suffixes.

3 Results

The following is a detailed account of the algorithms that we implemented. For each algorithm, we calculated precision (number of correct cuts made/ number of cuts made), recall (number of correct cuts made/ total number of correct cuts), and F-Measure (2*Precision*Recall / Precision+Recall). These scores were calculated with respect to a gold standard of approximately 900 words, segmented by hand by a native English speaker. Experimental results are summarized in Table 1. The following are details of the implementation of each algorithm.

1: *Successor count reaches cutoff.* Cuts are made whenever the successor count of a substring surpasses a given cutoff. This algorithm is not very effective; the only way to get even a reasonable F-Measure is to set the cutoff so low that the recall is high, because the algorithm makes excessive cuts.

2: *Successor and predecessor counts reach cutoffs.* Cuts are made whenever the successor and predecessor counts surpass their respective cutoffs. This algorithm is an improvement over (1), but suffers from similar drawbacks.

3: *Successor plus predecessor counts reaches cutoff.* Cuts are made whenever the sum of the successor and predecessor counts surpasses a given cutoff. This algorithm is much more effective, as it makes cuts when either the predecessor or successor count is high enough, or if both are moderately high, combining the best elements of the previous two algorithms. This algorithm gains greater precision without sacrificing recall. This algorithm benefited from the use of entropy scores rather than successor and predecessor counts, with the reuslts listed as (3a).

4: *Prefix is a complete word.* Cuts are made whenever the substring up to that point is a word found in the corpus. This algorithm identifies certain cuts, like plural nouns, extremely well, but is oblivious to almost all prefixes and to any irregular spelling changes.

5: *Suffix is a complete word.* Cuts are made whenever the substring following that point is a word found in the corpus. This algorithm is excellent at prefix cuts, but loses nearly all verb endings, plural nouns, and other short, non-word suffixes.

6: *Prefix is a complete word or predecessor count reaches cutoff.* Cuts are made whenever the substring up to that point is a word found in the corpus, or when the predecessor count surpasses a given cutoff. This algorithm attempts to supplement (4) by also finding prefixes. This is an improvement over (4), as recall increases by 26% without sacrificing any precision.

$Method$	$Precision$	$Recall$	$F-Measure$
1	0.19	0.74	0.30
2	0.26	0.67	0.38
3	0.41	0.67	0.51
3a	0.45	0.73	0.56
4	0.40	0.64	0.49
5	0.38	0.44	0.41
6	0.40	0.81	0.53
7	0.27	0.70	0.39
8	0.39	0.51	0.44
8b	0.39	0.51	0.44
9	0.36	0.73	0.48
9b	0.35	0.78	0.48
10	0.36	0.73	0.48
10a	0.28	0.89	0.43
11	0.70	0.57	0.63
11a	0.81	0.54	0.65
11c	0.56	0.59	0.58
11d	0.77	0.56	0.65

Table 1: Experimental Results. (*a*) indicates algorithms using entropy, (*b*) indicates algorithms where we did not allow peaks at 1, (*c*) indicates an algorithm using the threshold values listed in Hafer and Weiss (1974), (*d*) indicates an algorithm that disallows multiple one-letter suffixes

7: *Successor count at peak/plateau*. Cuts are made whenever the successor count is greater than or equal to the successor counts on either side. This algorithm is an improvement over (1), but suffers from low precision.

8: *Successor and predecessor at peak/plateau*. Cuts are made whenever the successor and predecessor counts are both at a peak/plateau. This algorithm appears to make errors on sparse data, when there are sequences of ones in the successor or predecessor counts. However, when we tried to correct for that by not allowing a one to be considered a peak/plateau (results listed as 8b), it made no difference.

9: *Successor plus predecessor at peak/plateau*. Cuts are made whenever the sum of the successor and predecessor counts is at a peak/plateau. This algorithm also appears to make errors on sparse data, when there are sequences of ones in the successor or predecessor counts. However, when we tried to correct for that by not allowing a one to be considered a peak/plateau (results listed as 9b), it made a trivial difference.

10: *Prefix is a complete word or predecessor at peak/plateau.* Cuts are made whenever the substring up to that point is a word found in the corpus or the predecessor count is at a peak/plateau. This algorithm adds very little new information as compared to (9). When this algorithm is run with entropy values instead of predecessor scores, it gains recall at the price of a significant loss of precision (10a).

11: *Hybrid of (2) and (6*). This algorithm combines earlier algorithms in an intelligent way. Cuts are made either when the prefix is a complete word and the predecessor count is above a threshold, or when both predecessor and successor counts are above thresholds. We optimized the thresholds for our data, which gave a noticable improvement over (11c), which used the threshold values listed in Hafer and Weiss (1974). Finally (11d), we checked for consecutive one-letter suffixes at the end of a word, and eliminated the less probable cut. This resulted in one of our best results, along with (11a), which replaced predecessor and successor counts with entropy scores.

We used algorithm (11d) to segment the entire corpus and found the most frequent morphemes. We eliminated some of the most freqent morphemes using a method similar to Déjean (1998). First, we eliminated any morpheme that occured less than 500 times in the segmented corpus. We then checked each morpheme to see if it was a substring of another morpheme on the list which had a frequency that was three times greater. The resulting morphemes are shown in Table 2.

Morpheme	*Frequency*
s	19962
ing	7253
ed	7126
er	4853
t	3367
ly	2833
al	1957
on	1391
ness	950
es	931
en	736
or	731
ic	674
able	646
ist	588
an	556
ment	510

Table 2: Most frequent morphemes in the corpus segmented by (11d)

4 Discussion & Conclusions

Overall our experiments were very successful. It is difficult to measure the recall and precision of experiments such as these because those values are completely dependent on the specific gold standard used. In fact, the task of morphological segmentation is a difficult one to measure quantitatively because it relies completely on the gold standard, and the gold standard is, by definition, subjective. This does not mean that the task is not worth attempting, but it means that any precision and recall numbers should be considered at least somewhat subjective. For example, our best segmenter would segment the word "formatting" as format-t-ing, while our gold standard says format-ting. A solid linguistic case could be made for either segmentation being correct; however, our experiment marks the prior as incorrect and the latter as correct.

The list of most frequent morphemes that our algorithm generates is promising. It seems to correspond with linguistic intuition about English, ranking morphemes such as "s", "ing" and "ed" at the top. Although we did not use this information in any of our segmenters, it would be the first step in implementing algorithms such as that described in Déjean (1998). Conceivably, a reliable list of frequent morphemes could also be used to modify and improve algorithms that use prececessor and successor counts or entropy. This is a direction for further investigation.

In general, we attempted to take the algorithms described in Hafer and Weiss (1974) and analyze a small sample of the results by hand. This is what inspired the improvements that we made to the basic algorithms, some of which were successful, and others of which had little effect. In the end, our improved algorithms were able to do slightly better than the best of their algorithms, when measured on our gold standard.

References

Hervé Déjean. 1998. Morphemes as necessary concept for structures discovery from untagged corpora. In *Proceedings of the ACL-98 Workshop on New Methods in Language Processing and Computational Natural Language Learning*.

Margaret A. Hafer and Stephen F. Weiss. 1974. Word segmentation by letter success varieties. *Information Storage and Retrieval*, 10:371–385.

Zellig Harris. 1955. From phoneme to morpheme. *Language*, 31:190–222.

Zellig Harris. 1967. Morpheme boundaries within words: Report on a computer test. In *Transformations and Discourse Analysis Papers*. Department of Linguistics, University of Pennsylvania.