Word Segmentation Algorithms

Ruth Talbot
Swarthmore College
500 College Ave,
Swarthmore, PA, 19081
rtalbot1@swarthmore.edu

Razi Shaban
Swarthmore College
500 College Ave,
Swarthmore, PA, 19081
rshaban1@swarthmore.edu

Alec Pillsbury
Swarthmore College
500 College Ave,
Swarthmore, PA, 19081
apillsb1@swarthmore.edu

Abstract

This paper discusses the implementation of various word segmentation algorithms over morpheme boundaries. Using minimally-supervised learning approaches discussed by previous authors, we implemented and analyzed various approaches for separating morphemes, with most approaches utilizing a trie data structure. We expanded these basic algorithms to combine their functions in an attempt to improve perforance, altough our results were less than stellar. Our best-functioning algorithms include a combination of reverse and forwards entropy and a more intelligent complete-word segmenter.

1 Introduction

The task of separating words into individual morphemes has many applications in processing natural text and speech. Due to the complexity and irregular nature of the English language, however, separating a word into morphemes is not a straightforward process. Most morpheme dictionaries are currently made with a lot of human input, either by manually segmenting words or via rules and heuristics designed specifically for the English language. We implemented many common morpheme segmenting algorithms described in Harris (1955), Harris (1967), and especially Hafer and Weiss (1974). We tested our segmenters against the gold standard provided by the MorphoChallenge 2010 as well as a gold standard of our own construction. Finally, we implemented several additional algorithms and improvements on algorithms not described in any of the papers to determine if we could increase performance of the algorithms given or create superior algorithms. In this paper we describe the algorithms we implemented, the cutoffs we chose for those algorithms, and the results they gave us as well. [1] Our approach was largely based off of the methods described in Hafer, 1974.

2 Methods

In this section, we describe each algorithm we implemented and the cutoffs we used for each, as well as the corpora used and the methods of testing.

2.1 Basic Cutoff Algorithms

We began by implementing the basic algorithm techniques described by Hafer and Weiss and their variants, and then tested each of them to find the most effective cutoffs and implementations for each.

2.2 Successor Cutoff

Successor cutoff corresponds to the basic cutoff algorithm described in 3.a in Hafer and Weiss (373). It uses a successor trie to generate the successor frequencies associated with each substring of a word. It then iterates through the word, splitting it when the substring is at a cutoff point. While Hafer and Weiss recommend 5 as a cutoff, we devised a program that tested each of our algorithms to find the cutoff

[1] http://nlp.cs.swarthmore.edu/~richardw/papers/hafer1974-word.pdf

that returned the highest F-score relative to the MorphoChallenge 2010 golden standard. Our successor cut-off algorithm peaked at an F-score of 0.2318, sacrificing precision (0.1563) for recall (0.5246).

2.3 Predecessor Cutoff

Predecessor cutoff corresponds to the basic cutoff algorithm described in 3.a) in Hafer and Weiss. However, it uses a predecessor trie instead of a successor trie to generate the predecessor frequencies associated with each substring of a word. It then iterates through the word, splitting it when the substring is at a cutoff point. Hafer and Weiss recommended 17 as a cut-off, but our algorithm peaked at a cutoff of 19, with a far superior precision to successor cutoff (0.4055) but a diminished recall (0.2978) for an F-score of 0.3298.

2.4 Successor and Predecessor Cutoffs

Successor and Predecessor cutoff combines the successor and predecessor cutoff algorithms. It uses a predecessor and successor trie to generate the predecessor and successor frequencies associated with each substring of a word. It then iterates through the word, splitting it when the substring is at either a successor or predecessor cutoff point. As this meant interactions between multiple cutoffs, our cutoff testing algorithm proved very useful. Hafer and Weiss recommended 5 for successor and 17 for predecessor, and our algorithm maximized F-score at 5 for successor and 1 for predecessor, returning a precision of 0.2617 and a recall of 0.3132 for an F-score of 0.2656. We attribute the wide variance in cut-offs to a different means of counting predecessor cutoffs.

2.5 Successor plus Predecessor Cutoff

This algorithm simply sums the successor and predecessor cutoff algorithms. It uses a predecessor and successor trie to generate the predecessor and successor frequencies associated with each substring of a word. It then iterates through the word, splitting it when the substring is at a combined successor and predecessor cutoff point. We experimented with several different cutoff points. Hafer and Weiss recommend a combined total of 23. However, our tests proved this to be a poor cutoff for our implementation of the algorithm, which maximized F-Score with a precision of 0.3585 and a recall of 0.2928 for an F-Score of 0.3117.

2.6 Successor Peak or Plateau

The plateau algorithms are based on the recommendations made in 4.b)bii in Hafer and Weiss. Plateau determines whether or not a stubstring is at a peak or plateau in its frequency, meaning that the frequency of the substring before and the substring after are either equal or less than that of the current substring. If the freqencies are equal or less than, the algorithms splits the word. Successor plateau was more successful than the cutoff functions, with a precision of 0.2876 and a recall of 0.5534 for an F-score of 0.3547.

2.7 Predecessor Peak or Plateau

This algorithm is the same as the Successor Peak or Plateau, except that it implements predecessor peak and plateau as opposed to successor. Our implementation of this algorithm proved less successful than its successor variant, with a precision of only 0.1050 and a recall of 0.3137 for an F-score of 0.1500.

2.8 Loose Cutoff

This algorithm corresponds to 4.b.vi in Hafer and Weiss (378). It implements a less restrictive version of predecessor and successor freqency cutoff, splitting when the substring is a complete word and predecessor freqency is moderate or when predecessor count is strong and successor frequency is moderate. This algorithm improved on successor but not predecessor cutoff, with a precision of 0.3486 and a recall of 0.2880 for an F-score of 0.3043.

2.9 Complete Word - Simple

Our complete word algorithm corresponds to the algorithm described in 3.c) of Hafer and Weiss (374). Essentially, the simple form of this algorithm (we developed a more intelligent version that will be described below) iterates through a word and breaks the word at the end of any complete words it finds. The dictionary it used is the entirety of the corpus we used to populate our tries, in this case, the Brown corpus. It produced a precision of 0.3683, recall 0.3195, and an F-score of 0.3292.

2.10 Basic Entropy Algorithms

2.11 Successor Entropy Cutoff

This algorithm corresponds to 3.d) in Hafer and Weiss. It uses a function to get successor entropy for each substring and then iterates through each substring in the word, cutting when successor entropy reaches a cutoff. Hafer and Weiss seem to indicate that 3 is a good cutoff for entropy. Our testing found that a cutoff of 3.1 gave us the highest F-score, with a precision of 0.1580 and a recall of 0.5216 for an F-Score of 0.2330.

2.12 Predecessor Entropy Cutoff

This algorithm is essentially the same as Successor Entropy Cutoff, except that it uses predecessor entropy instead of successor entropy. Hafer and Weiss recommend a cutoff of 3, but we found that using a cutoff of 3.6 maximized our F-score, with a better recall (0.4545) and slightly lower precision (0.3741) leading to significantly better improvement over successor entropy (F-score 0.3981).

2.13 Loose Entropy

This algorithm corresponds to to 4.b)xiv) in Hafer and Weiss. The algorithm splits a word if the word is a complete word and predecessor entropy is moderate or if predecessor entropy is strong and successor entropy is moderate. Because this algorithm relaxes the standards, it was supposed to outperform the stricter entropy algorithm; however, it did not outperform the Predecessor Entropy Cutoff, with a precision of 0.3585 and a recall of 0.2928 for an F-Score of only 0.3117.

2.14 Advanced Algorithms

2.15 Complete Word or Predecessor Frequency at Peak or Plateau

This algorithm corresponds to 4.b)x) in Hafer and Weiss. It determines whether a substring of a word is a complete word or predecessor freqency is at a peak or plateau. If it is, it splits the word at that point. For example, it would split 'restful' at 'rest', as it was a complete word, but would also split anywhere where it found a peak or plateau in predecessor freqency. This algorithm is slightly better than Complete Word - Simple, with a precision of 0.2546, recall of 0.6925, and an F-score of 0.3508.

2.16 Complete Word or Predecessor Entropy Cutoff

This algorithm corresponds to 4.b)vii). It checks to see if a complete word is present or predecessor entropy reaches a cutoff. If either of these conditions are true, it splits the word. Both predecessor entropy and checking for a complete word are supposed to produce good results. Therefore, this algorithm was expected to perform well. However, it performed worse than both Complete Word and Predecessor Entropy, even after finding a cutoff (1) that maximized F-score. At cutoff 1, this algorithm had precision of 0.1544, recall of 0.6955, and an F-score of 0.2473.

2.17 Complete Word - Complicated

We designed this algorithm as a more intelligent improvement on Complete Word - Simple. It begins by attempting to recognize a dictionary word from the back of the word to be segmented, iterating backwards from the end of the word. If it cannot find any word breaks, it performs Complete Word - Simple from the beginning of the word. Following this, the algorithm completes another run through the segments to try to break out any further words. This approach proved more successful at distinguishing word roots from affixes, with a precision of 0.5430, a recall of 0.4087, and an F-score of 0.4505.

2.18 Ricky

This algorithm was designed to help segment off prefixes and postfixes. It begins by running predecessor entropy backwards on the word to try and clip off the postfix. After this, the algorithm runs successor entropy forwards on the initial segment of the word to try and break off a prefix. Following this, the algorithm searches through the segments for any dictionary words to further segment. We found this algorithm optimized F-score at predecessor cutoff 3.1 and successor cutoff of 3.5 to give precision of 0.4669, recall of 0.6959, and an F-score of 0.4379.

3 Testing

3.1 Creating our Own Gold Standard Data

In addition to testing against the provided cut words, we randomly selected words and created a shorter list of words that we hand segmented to create our

own gold standard for segmentation, focusing on segmenting the words in a logical manner. Words of Latin origin were explicitly cut on the whole stem, and we largely aimed to keep common suffixes intact (e.g fad ed as opposed to fade d).

3.2 Testing Program

We created a testing program that looped through each of our algorithms, comparing our segmentation to both our gold standard and the gold standard provided. We calculated precision and recall by comparing the indexes of the cuts made in the gold standard provided and the output from our algorithms. The number of true positives was calculated as the intersection of our guess and the key provided. False positives were calculated as the number of cuts made in our algorithm output that were not in the gold standard, and false negatives were the number of cuts that were in the gold standard but not our algorithm output. Precision was then calculated as the number of true positives divided by the sum of the true and false positives. Recall was calculated as the number of true positives divided by the number of true positives and false negatives. Our F-score was defined as the harmonic mean of precision and recall, found by multiplying precision and recall by two and then dividing by the sum of precision and recall. We also trained each algorithm that used cutoffs on a range of cutoff values to find the best cutoff value for each algorithm. This involved iterating through steps of cutoff values (steps of 1 for frequency cutoffs and steps of 0.1 for entropy cutoffs). After running all cutoff values, the testing program outputed the highest fscore value found and the cutoff value that produced it. Our best cutoff values are listed in the Algorithms section in each algorithm description.

3.3 Corpora

We created a test program that allowed us to use multiple corpora and change between corpora to determine which gave us the best results. The majority of test were done on the Brown corpus, Scrabble corpus, and a combination of several different corpora. Our results section includes the tests on several different corpora. We chose our cutoff values based on the numbers given by the Brown corpus. Interestingly, our tests, especially those based on entropy, performed better on the Brown corpus than on the Scrabble corpus. This indicates that entropy does better on tokens rather than types, because the Brown corpus allowed for multiple of the same words to be entered into the trie, whereas the Scrabble corpus did not. In general, increasing the size of the corpus used beyond a corpus this size of the Brown Corpus did not yield better results. This finding may indicate that after a certain amount of training data, more data is ineffectual.

4 Results

Our results with the provided gold standard are shown on Table 1. The results using our gold standard are shown on table 2.

5 Analysis

Our algorithms were plagued by a trade-off between precision and recall, a trade-off that may not be reflected in our final data. Either our algorithms cut too frequently, giving low precision but high recall, or they did not cut enough, giving high precision but low recall. In an attempt to balance these, we decided to use F-score, the harmonic mean of precision and recall, to try to find cutoffs that were more effective, but for most of our poorly-performing algorithms this meant many cuts and low precision. Our best-performing algorithms, Complete Word - Complicated and Ricky, managed to use multiple runs through a word in a way that seems to have balanced recall and precision for our best results of any algorithm. We were unable to look to any of our sources for specifics on how to address the precision and recall balance, as Hafer and Weiss simply recommend ”softening the cutoffs” to help deal with this issue (380). However, we were unable to implement this suggestion effectively. There are several specifics about which all authors were unclear, and therefore we had to make our best guess as to their method. For instance, while Hafer and Weiss specify several specific cutoff points for successor and predecessor entropy as well as frequency, they do not always specify cutoff points. Some of their descriptions of algorithms are vague. For example, successor/predecessor frequency at peak or plateau does not specify if a plateau or peak can be at extremely low frequencies, such as 1 1 1 or 1 2 1. We

initially allowed a peak or a plateau to be at very low freqencies, which led to many cuts being made. Additionally, Hafer and Weiss did not specify whether or not they included non-alphabetic characters in their Tries. We allowed all letters to be followed by any character, alphabetical or not, which Hafer and Weiss may not have done. We did not make any exceptions to account for punctuation marks but if they did, it may have improved their algorithms' performance. Below we report the general trend for all the algorithms implemented and the best numbers they produced.

5.1 Cutoff

Successor cutoff behaves as Hafer and Weiss indicate it does. It makes many cuts directly at the beginning of the words, and very few towards the end. Therefore, on a word such as 'restful', successor cutoff segments off 'r' 'e', etc., and does not segment the end of the word. This is because r and e have many letters that can follow them. If the cutoff is less than 4 or 5, a word is cutoff at every single one of the first few letters, and none after that. Adjusting the cutoff makes little difference.

5.2 Reverse Cutoff

Predecessor cutoff performs very well. Among the best of our algorithms actually. It does segment too much towards the end, but with a high enough cutoff, it does a good job of identifying suffixes and splitting at them. However, it does a fairly poor job after that, identifying almost no cutoff points.

5.3 Successor and Predecessor at Cutoff

Successor and predecessor cutoff does not actually do well. It appears that almost no words in the corpus have a peak in predecessor and successor frequency simultaneously and therefore few words get split. In fact, examining our data it appears that almost none of our words was split at any point. This is why our test program recommended a very low cutoff, to increase cuts.

5.4 Successor plus Predecessor at Cutoff

Successor plus predecessor does not do very well, but it performs reasonably. Since both successor and predecessor counts are very high at the beginning and end of the words, almost no substrings did not make the cutoff point and get split unless the cutoff point is set very high. But even then, the cuts it makes are non optimal.

5.5 Complete Word Simple

Simple complete word cutoff works decently, yielding similar results most of the better generic cutoff methods. However, it fails to search for dictionary words that have a prefix attached to them, and reversing the direction of iteration does not help sort out postfixes. In an attempt to help remove prefixes and postfixes to help find dictionary words, we created algorithms to build lists of prefixes and postfixes, but with this isolated task they did not prove very effective. Instead, they lead us to the idea behind Complete Word - Complicated, which was far more effective than Complete Word - Simple.

5.6 Complete Word and Predecessor Plateau

Complete word and predecessor frequency produces quite good results, with a reasonable precision. As would be expected, this algorithm has higher recall than the simple complete word algorithm, due to the fact common suffixes that would have been missed by complete word are picked up as well. However, the plateau does not seem to do well with low-frequency strings, a weakness that we have been unable to fix from within the algorithm. We experimented with various internal cutoffs and conditions, but could not find a way to accurately quantify the kind of plateau we are looking for.

5.7 Loose Cutoff

Our relaxed form of cutoff does reasonably well, but not as well as Hafer and Weiss's in comparison to other tests. In fact, it does about average compared to all other tests run, which we weren't expecting given Hafer and Weiss's finding it to be one of their best algorithms. One of the reasons that this might be the case is because we used the cut-offs that they recommended, instead of potentially varying the range of cut-offs we used as we trained our other cutoff algorithms.

5.8 Entropy

Our generic entropy function returned mediocre precision results but decent recall results. Many of the entropy offshoots achieved better precision at

the cost of some recall. In general, most entropy algorithms outperformed their cutoff counterparts, which is somewhat expected due to the increased complexity of entropy algorithms.

5.9 Loose Entropy

Loose entropy showed a significant increase in precision, at the cost of a fairly large decrease in recall. Interestingly enough, this algorithm performed very similarly to the generic cutoff algorithm.

5.10 Reverse Entropy

Like loose entropy, the reverse version of entropy improved over the original in precision at the cost of recall.

5.11 Complete Word - Complicated

Our more complex implementation of complete word performs significantly better than our simple implementation. On both gold standards this algorithm outperformed all other algorithms by a significant margin. At the heart of this improved performance is the act of iterating both backwards and forwards to try to find a word, allowing the algorithm to separate out prefixes and postfixes more effectiely than Complete Word - Simple. Because this algorithm still uses a dictionary, however, it remains far more accurate than frequency or entropy algorithms, which cannot recognize the forms of the common words which lie inside complex words.

5.12 Ricky

Our custom algorithm Ricky was the only algorithm to rival Complete Word - Complicated in performance. Ricky was the product of experimenting with using entropy to duplicate the three-step process used in Complete Word - Complicated. Thus, it used predecessor entropy to try to peel off the postfix, followed by successor entropy to try to segment off the prefix, followed by Complete Word - Simple on what was identified as the remaining root of the word. We were able to almost match with entropy the results yielded via dictionary by Complete Word - Complicated, which gives us hope that further experimentation with the entropy framework, including layering entropy and dictionaries, will allow us to produce far more accurate algorithms.

6 Conclusion

Our results seem to suggest that running multiple passes over a single word is a step towards implementing a better word-segmenting algorithm. The two functions where we stepped furthest from previous sources' techniques and layered approaches were the two with the best results. Our more advanced complete word segmenter showed that a dictionary is one of the most powerful ways we can segment words, but the performance of our custom function Ricky shows that a dictionary is not necessary for a high-performing algorithm.

6.1 Our Improvements

There are many improvements that we added to our various algorithms. Besides going through a couple iterations of our underlying data structure, the Trie, we also generalized the structure of most of our algorithms so that they all returned comparable outputs. The biggest ostensible improvement we implemented was with the complete word algorithm. We implemented two complete word algorithms, Complete Word Simple and Complete Word Complex, the former being a naive implementation of the algorithm and the latter utilizing various heuristics that we found helpful. Layering algorithms for multiple run-throughs appears to be the biggest improvement we implemented. Comparing the F-scores of Complete Word - Simple (0.3913) and Complete Word - Complicated (0.4709) shows that large gains can be had without changing the underlying mechanism of the algorithm. This F-score jump was accompanied by a more significant increase in precision – from 0.5016 to 0.6600.

6.2 Future Suggestions

The larger implication of our experiments show that heuristics based on general observations about the English language improve the accuracy of more general algorithms. English is a language that uses both postfixes and prefixes, so it makes sense that running algorithms matched to this pattern will be more effective. A future improvement could be to formalize heuristics so that other algorithms could simply interact with an API to recognize the word-formation patterns and apply the most helpful algorithms. This could also help with the recall-precision imbalance,

as picking the algorithm that will be most effective on a specific word will give a more important measure of segmentation quality than F-score.

Table 1: Algorithm performance over segments.eng test set

Algorithm	Precision	Recall	F-Score
Cutoff	0.1563	0.5246	0.2318
Reverse Cutoff	0.4055	0.2978	0.3298
Pred and Succ Cutoff	0.2617	0.3132	0.2656
Cutoff	0.3485	0.2880	0.3043
Loose Entropy	0.3585	0.2928	0.3117
AddPS Cutoff	0.1732	0.8317	0.2800
Entropy	0.1580	0.5216	0.2330
Reverse Entropy	0.4545	0.3741	0.3981
Complete Word - Simple	0.3683	0.3195	0.3293
Complete Word - Comp	0.543	0.4087	0.4505
Plateau	0.2876	0.5534	0.3547
Predecessor Plateau	0.1990	0.3573	0.2353
Comp. Word and Pred Plateau	0.2546	0.6925	0.3508
Comp. Word and Pred Entropy	0.1544	0.6956	0.2473
Ricky	0.4669	0.4494	0.4379

Table 2: Various Algorithm Results over our Gold Standard

Algorithm	Precision	Recall	F-Sc
Cutoff	0.2166	0.6058	0.309
Reverse Cutoff	0.3100	0.1725	0.21
Loose Cutoff	0.3300	0.2241	0.25
Loose Entropy	0.4200	0.2583	0.308
AddPS Cutoff	0.2037	0.8508	0.323
Entropy	0.2195	0.5933	0.309
Reverse Entropy	0.4550	0.2783	0.327
Complete Word - Simple	0.5016	0.3533	0.39
Plateau	0.3744	0.6141	0.446
Predecessor Plateau	0.0940	0.0624	0.069
Complete Word and Pred Plateau	0.1874	0.7366	0.293
Complete Word and Pred Entropy	0.1718	0.7033	0.27
Complete Word - Comp	0.6600	0.3916	0.470
Ricky	0.4833	0.4150	0.41