

# IMPLEMENTATION OF AN AUTOMATED TEXT SEGMENTATION SYSTEM USING HEARST'S TEXT TILING ALGORITHM

**Brent Fitzgerald**

*brentf@stanford.edu*

CS224N Final Project - June 1, 2000

## **ABSTRACT**

This paper describes the implementation of a text segmentation system based on Hearst's TextTiling algorithm. Hearst is a pioneer in the field of text segmentation, and her algorithm has already been shown to provide good results. The algorithm uses lexical frequency and distribution information to recognize the level of cohesion between blocks of text, and then uses these cohesion estimates to judge which sections are likely to be different topics.

## **INTRODUCTION**

Most of the texts one comes across are composed of a number of topics, perhaps varying in their relevance to one another and their scope. A system that could automatically detect these subtopics would certainly be useful, allowing the reader to quickly skip to the topics most relevant to her purpose. The segmentation might also aid in tasks of information extraction and summarization, since it provides structural semantic information about the document. The ability to identify the various subtopics could let one quickly build outlines of the essential points.

More recently, the web's proliferation has led to an overwhelming increase in readily available information, but finding the information one needs can be a difficult task. Search engines and directories provide a means of classifying and organizing this information on a multi-document level, but there is still a need for a system that can provide organization within long, information rich documents. A good segmentation system, perhaps combined with summarization and information extraction technologies, could fill this niche quite nicely. Thus, any highly accurate segmentation system would certainly be useful in these times of overly abundant, undocumented data.

The system described in this paper is currently not up to this daunting task, but it is an interesting experiment in building a system that automatically locates topic boundaries. This paper will review the algorithm

behind the system as well as some of the practical aspects of the implementation, and will conclude with a discussion of the results and some possible extensions of the current system.

## **ALGORITHM AND IMPLEMENTATION**

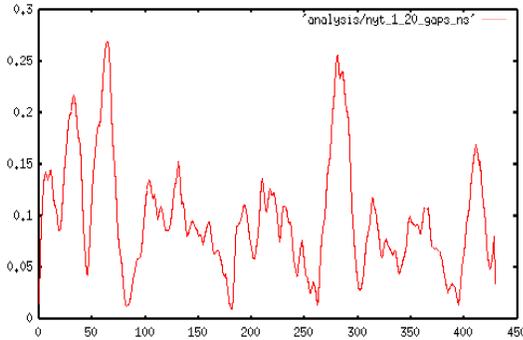
There are several different approaches that have been presented in the literature. The approach used in this paper is based on Hearst's TextTiling algorithm, a moving window approach that uses lexical overlap as a means of detecting topic coherence. Another approach called dotplotting, presented by Reynar (1994) and furthered by Choi (2000), finds the similarity between every pair of sentences in the document and uses these results to identify chunks of cohesive sentences. A very different strategy called Lexical Chaining uses lexical semantic similarity information to create "chains" of related words. Generally, a document will have at least several of these chains, allowing one to segment the document based on the features of the chains, such as start and end points. Hearst's algorithm is used in this system because it is relatively straightforward and well documented.

Hearst defines three main components of the TextTiling algorithm. First, it divides the input text into sequences of relevant tokens and calculates the cohesion at each potential boundary point. It then uses these cohesion scores to produce depth scores for each potential boundary point that has a lower cohesion than the neighboring boundary points. Using these depth scores, the algorithm is able to select boundary points where the depth is low relative to the other depth scores, indicating that that gap represents a topic shift in the text. The output is the text file with boundaries inserted at these gaps with sufficiently high depth scores, delineating the various topics by breaking at the least cohesive points.

The first task of this system, then, is to calculate the gap scores. In order to do so, it is first necessary to break the document into appropriately sized sequences of text. Gap cohesion is computed between a group of text sequences immediately prior to the gap and a group of text sequences immediately after. Hearst advocates various strategies regarding methods of breaking the text into sequences. One method is to use chunks of text that have some fixed number of valuable tokens. For this approach, Hearst recommends 20 tokens per sequence. The benefit to this approach is that each sequence carries the same amount of information as the other sequences. The other method is to assign each sentence in the document to its own sequence. One advantage to this approach is that the boundaries tested are sentence boundaries rather than mid-sentence boundaries, and thus are better representative of where a change in topic is most likely to occur. The other, more practical advantage of this approach is that if the system finds the gap scores at the sentence boundaries, then it is extremely straightforward to insert the segmentation break points. The other method requires deciding upon the nearest sentence boundary. This system

uses a one sentence per sequence approach. The system also takes a list of stop words, which are words that uninformative regarding the topic of a particular passage such as *the, and, they, we, a, will, can, have*, etc.

Eliminating these stopwords will prevent the system from getting distracted by irrelevant data.



**FIGURE 1:** Gap score results from analysis of concatenation of 10 New York Times articles. Horizontal axis is the gap number, vertical axis is the gap score measured by cohesion of adjacent blocks. Greater vertical axis values indicate higher levels of cohesion. The breaks between the various articles tend to correlate to the low points in the graph.

The gap cohesion score is found by creating a vector from the token counts found in some fixed number  $n$  of sentence sequences immediately prior to the gap, and another vector from the token counts found in the same number  $n$  of sequences immediately following the gap. Hearst suggests a number of sequences approximately equal to the average paragraph length in sentences. A vector similarity metric, such as the cosine similarity, is then applied to these two vectors to obtain an estimate of the cohesiveness between the two sections. The cosine similarity can be computed. This number is called the *gap score*, and it is calculated at each potential boundary location, obtaining a distribution of gap scores with a visual

representation of the form seen in Figure 1.

The next step is the smoothing process. As we see in Figure 1, the initial computation of gap scores leaves one wanting clearer boundary markers, since many small local minima might lead to too many small segments in our output. The system lessens the effect of these small local extremities using an average smoothing technique with a flexible window size. Using this system, gap score  $s_i$  is replaced by  $(s_{i-k/2} + \dots + s_i + \dots + s_{i+k/2}) / (k + 1)$ , for some optimally configured  $k$ . The size of  $k$ , of course, should depend on the type of document being segmented and granularity of segmentation desired. A smaller  $k$  value will leave more of the original information intact, making it a good choice for shorter texts like newspaper articles, but it can lead to too much fragmentation by failing to sufficiently eliminate undesired noise. Larger values of  $k$  eliminate the subtleties in the data, and thus are useful if one is planning to segment a larger text. Note that in this implementation, if there are not enough gap scores to smooth using the  $k$  value chosen, then the window size collapses to a suitable value. This allows us to smooth the score distribution near the beginning of the text. See Figure 2 for a visual representation of the effects of smoothing on the gap score

Now that the correlation scores have been calculated and smoothed, the next step is to locate the high and low points in this set of data. A list of the peaks is obtained by culling the scores for local maxima, and then each pair of adjacent peaks is used to find the lowest gap score in the valley between. Using these local minima and their neighboring local maxima, it is fairly straightforward to calculate the depth score, which is the difference in height of the left peak and the low point, plus the difference between the right peak and the low point. The depth score is an indication of the lack of correlation at that gap relative to the correlation at the nearest maxima. Thus, if the depth score is high, then the correlation is particularly low relative to the nearby preceding and successive gaps. If the depth score of a gap is low, then the gap is most likely not a break, since it's gap score does not differ from it's neighbors so much as the other depth scores.

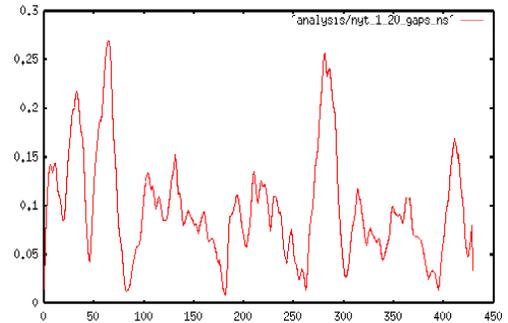
To find the boundary points the system finds the depth scores that are sufficiently large relative to the other candidate depth scores in the document. This is accomplished by including only those depth scores that exceed  $mean - c$  (*standard deviation*), for some optimally configured value  $c$ . Hearst recommends a value of 1/2 based on her experiments. Larger values of  $c$  increase the number of inserted boundary points.

## EVALUATION AND RESULTS

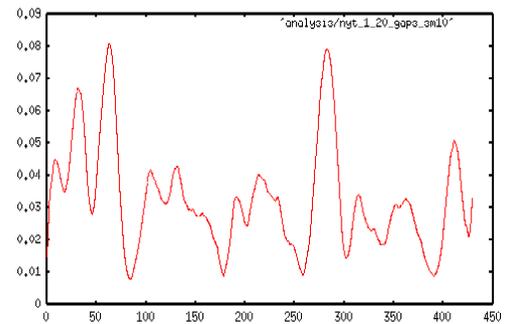
Evaluation of the system's performance consists of running the system on a concatenation of newspaper articles. Newspaper articles seem a decent choice of data because they are readily available and reasonably short, so they can be concatenated together to obtain longer documents where the topic structure is already known. One potential problem with the use of

**FIGURE 2:**  
SMOOTHING OF NEW YORK TIMES GAP SCORES

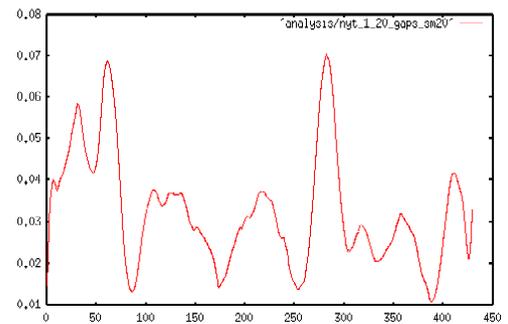
The following four figures show the effect of smoothing on the New York Times data with various window sizes.



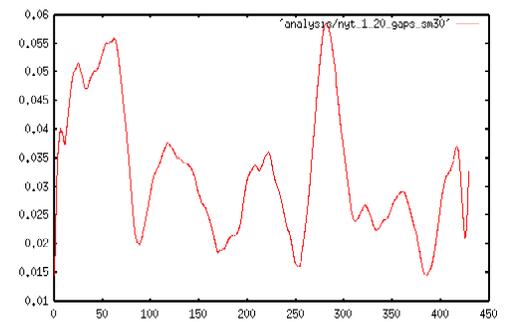
Concatenated Times Articles, no smoothing ( $k = 0$ )



Same data, smoothing with window size 10 ( $k = 10$ )



Same data, smoothing with window size 20 ( $k = 20$ )



Same data, smoothing with window size 30 ( $k = 30$ )

newspaper articles is that they don't necessarily contain only one major topic. An article might contain several subtopics, each of which might be relevant to one another but no more so than the other articles in the data, which could lead to boundaries inserted mid article.

Ideally, it would have been informative and worthwhile to test the system against the segmentation choices made by human judges, as Hearst did in the original evaluation of her system. Hearst's evaluations compared her implementation's performance to that of human judgement, and it fared relatively well with an average precision score of 0.66 compared to the judge's 0.81, and average recall of 0.61 compared to the judges recall score of 0.71. Indeed, when run on non-test data, the segmentation of this system seems quite reasonable.

The tests were run with a variety of parameter specifications. The default parameters of the system were determined by taking the parameters that yielded the highest combined level of precision and recall. In the initial tests, the smoothing window sizes 10 and 20 were found to be too large and significantly hurt both the precision and performance. In the second round of tests, the parameters were kept much more moderate. The results of these tests are attached to this document. The best precision score was 0.77 when run on the New York Times texts, and it was accompanied by a recall score of 0.77 as well. While these scores may sound relatively impressive, it is important to note that they were only numerically evaluated on this one set of data, and so it is unlikely that those parameters would return such high scores in all circumstances.

## **FURTHER RESEARCH**

This implementation makes no use of structural cues in the text, and it would be interesting and most assuredly beneficial to consider this structure. This could be done by modifying the algorithm to assign the break only to the nearest paragraph boundary, rather than ignoring the white space as we have in this implementation. The choice was made to ignore white space information in order to allow for greater flexibility in the text data we wish to segment. However, if the system were operating within a narrower domain, it would be advantageous to tune the system to take advantage of available cues. For example, if the system was applied to html tagged web page texts, then it would probably be useful to weight the segmentation scheme to break at <P> paragraph boundary tags or <BR> break tags.

Another avenue of research is key word and sentence extraction from the sections obtained using this segmenting system, producing a summary or outline of the topics covered in the document. This might be done using a key sentence extraction technique such as those used in summarization systems. It would be an interesting

research topic to try to improve summarization systems by using a segmentation system to break the text into its subtopics, then find the key sentence summaries for each topic.

Other segmentation systems use a stemming routine in the preprocessing stage of the system. Hearst ignores stem values and uses the bare words, but it would certainly be worthwhile to see how using the stems in the similarity measure might affect the segments produced.

Finally, TextTiling is language independent, failing to use any semantic information in measuring cohesiveness. Rather than basing the similarity measure on the number occurrences of words in the sequence, it might be beneficial to base the similarity measure on the occurrences of semantic *classes* of words. This might be done using the synonyms provided by WordNet, perhaps in combination with a sense disambiguator to determine the intended sense.

## **SUMMARY**

This paper describes research in text segmentation, specifically Hearst's text segmentation algorithm TextTiling. The system presented in this paper uses the TextTiling algorithm to compute the cohesion between blocks of text and determine the most likely boundary locations. While this system fails to perform as well as many of the other segmentation systems that have recently been presented in the literature, it is certainly on the right path and can produce good results with the proper parameters.

## **REFERENCES**

Choi, F., 2000, Advances in domain independent linear text segmentation. To appear in Proceedings of NAACL'00, Seattle, USA.

Hearst, M. 1993. TextTiling: A quantitative approach to discourse segmentation. Technical Report 93/24, U. of California, Berkeley.

Hearst, M. 1994. Multi-paragraph segmentation of expository text. In *Proceedings of the 32nd Annual Meeting of the Association for Computational Linguistics (ACL-94)*, New Mexico, USA, 9-16.

Ponte, J. M., Croft, W.B. 1997, Text Segmentation by Topic. In *Proceedings of the First European Conference on Research and Advanced Technology for Digital Libraries*, pp. 120-129.

Reynar, J. C. (1994). An automatic method of finding topic boundaries. In *Proceedings of the 32nd Annual Meeting of the Association for Computational Linguistics (ACL-94)*, New Mexico, USA.

Richmond, K., Smith, A. and Amitay, E., 1997, Detecting subject boundaries within text: A language independent statistical approach. In *Proceedings of the Second Conference on Empirical Methods in Natural Language Processing (EMNLP--97)*, pages 47--54, Providence, Rhode Island, August 1—2.

## ABOUT THE SOFTWARE

The programs included are everything one needs to get started segmenting text! Several properly formatted documents are already included, but it is straightforward to make new ones as well.

To segment a text document using this segmentation system:

1. Run *sentencesnipper* on the text. *Sentencesnipper* is a quick and dirty sentence boundary detection system. It takes ASCII text as input along with an optional (but highly recommended) list of common abbreviations. The output of *sentencesnipper* is a printout of each sentence separated by two newline characters. An example is as follows:

```
%> sentencesnipper/sentencesnipper ../raw_data/basketball abbreviations

The players dispersed after a tense timeout, but a frantic Coach Jeff Van Gundy was
still standing on the court.

There were just 12.4 seconds on the clock, and all his team needed was one last
defensive stop to leave the Miami Heat in pieces once again.

...
```

Note that *sentencesnipper* is not a full-fledged sentence boundary detection program. It sometimes has problems with some abbreviations (even with the abbreviations file included), and commonly inserts two spaces instead of one, and there may well be other yet to be discovered quirks. Generally, though, it does a good job splitting the sentences apart, and is quite appropriate for this particular task.

2. Run *segment* on the snipped text. *Segment* is the actual text segmentation program. It requires only one command argument, the document to be segmented. It also takes four optional arguments: a list of stopwords, the threshold coefficient, the comparison size, and the smoothing window size. For example:

```
%> segment data/unmarked_data/nytimes.unmarked stopwords 1 10 6
```

This command runs *segment* on *nytimes.unmarked* data file, with the stopwords file, a threshold coefficient at 1 (higher number translates to increased tendency to break at less salient gaps), a

comparison size of 10 (10 sentences before gap compared to 10 after), and a smoothing window size of 6 (average of 6 surrounding gap scores plus the one to be replaced).

3. Evaluate using *evaluation.pl*. This is the third component of the package, and it is used to test the accuracy of the segment program's output against a marked version of the same text. The marked text file should be chopped into sentences using *sentencesnipper*, with each segment boundary marked with a "<--BREAK-->" statement with one newline character between the statement and both the preceding and next sentences. *evaluation.pl* takes the name of the data to be tested, the name of the previously marked data, and an integer indicating the leniency. Here is an example of how to run it:

```
%> evaluation/evaluation.pl ../../nytimes.results ../../nytimes.marked 2
```

This compares the *nytimes.results* file with the *nytimes.marked* file, and counts a successful boundary identification even if the break is two or less sentences from the actual break. Here is an example of the output of the program:

```

                |      Actual
System          | target  !target
-----
selected       |      8      25
!selected      |      1     444

Precision = 0.242424242424242
Recall = 0.888888888888889
```

4. If any of this doesn't work right or if you have questions, please email [brentf@stanford.edu](mailto:brentf@stanford.edu).

These are the results of the second set of tests, The left field is the name of the file, where the first number in the name is the threshold coefficient, the second is the comparison size, and the third is the smoothing window size. Notice that as we decrease our threshold, disallowing the less pronounced breaks, precision increases as recall decreases. Also, notice that for a smoothing window size of 4 we usually get better results than with the other window sizes, and we also seem to get better results with a comparison size of 7. According to this data, the magic numbers are  $-0.5$  threshold, a 7 sentence comparison size, and a 3 smoothing window of size 4, since these figures yield the highest precision score of 0.77, and a decent recall score of 0.77 as well. However, to maintain some degree of generality and ensure that these good results are not specific only to this data, the default values of the actual system will have a weaker threshold of 0 rather than  $-0.5$ , ensuring that some segmentation will occur in most texts.

<u>Output file</u>	<u>Precision</u>	<u>Recall</u>
nytimes_0.5_3_2	0.183673469387755	1
nytimes_0.5_3_4	0.214285714285714	1
nytimes_0.5_3_6	0.189189189189189	0.777777777777778
nytimes_0.5_5_2	0.257142857142857	1
nytimes_0.5_5_4	0.3	1
nytimes_0.5_5_6	0.269230769230769	0.777777777777778
nytimes_0.5_7_2	0.28125	1
nytimes_0.5_7_4	0.333333333333333	0.888888888888889
nytimes_0.5_7_6	0.333333333333333	0.777777777777778
nytimes_0.25_3_2	0.214285714285714	1
nytimes_0.25_3_4	0.272727272727273	1
nytimes_0.25_3_6	0.192307692307692	0.555555555555556
nytimes_0.25_5_2	0.321428571428571	1
nytimes_0.25_5_4	0.36	1
nytimes_0.25_5_6	0.4	0.888888888888889
nytimes_0.25_7_2	0.346153846153846	1
nytimes_0.25_7_4	0.4	0.888888888888889
nytimes_0.25_7_6	0.368421052631579	0.777777777777778
nytimes_0_3_2	0.310344827586207	1
nytimes_0_3_4	0.28	0.777777777777778

nytimes_0_3_6	0.2777777777777778	0.5555555555555556
nytimes_0_5_2	0.375	1
nytimes_0_5_4	0.315789473684211	0.6666666666666667
nytimes_0_5_6	0.5333333333333333	0.8888888888888889
nytimes_0_7_2	0.409090909090909	1
nytimes_0_7_4	0.5333333333333333	0.8888888888888889
nytimes_0_7_6	0.4666666666666667	0.7777777777777778
nytimes_-0.25_3_2	0.428571428571429	1
nytimes_-0.25_3_4	0.25	0.5555555555555556
nytimes_-0.25_3_6	0.3333333333333333	0.5555555555555556
nytimes_-0.25_5_2	0.3888888888888889	0.7777777777777778
nytimes_-0.25_5_4	0.4	0.6666666666666667
nytimes_-0.25_5_6	0.5	0.7777777777777778
nytimes_-0.25_7_2	0.5	0.8888888888888889
nytimes_-0.25_7_4	0.6666666666666667	0.8888888888888889
nytimes_-0.25_7_6	0.461538461538462	0.6666666666666667
nytimes_-0.5_3_2	0.375	0.6666666666666667
nytimes_-0.5_3_4	0.384615384615385	0.5555555555555556
nytimes_-0.5_3_6	0.4166666666666667	0.5555555555555556
nytimes_-0.5_5_2	0.428571428571429	0.6666666666666667
nytimes_-0.5_5_4	0.461538461538462	0.6666666666666667
nytimes_-0.5_5_6	0.461538461538462	0.6666666666666667
nytimes_-0.5_7_2	0.545454545454545	0.6666666666666667
<b>nytimes_-0.5_7_4</b>	<b>0.7777777777777778</b>	<b>0.7777777777777778</b>
nytimes_-0.5_7_6	0.6	0.6666666666666667