TextTiling with ELMo

Benjamin Yang
UT Austin, Department of Computer Science
ben.yang@utexas.edu

Abstract

This paper describes an implementation of TextTiling from scratch and a modification of the algorithm that involves calculating cosine similarity between averages of ELMo word embeddings. We compare performance between the two on a standard synthetic data set (Choi, 2000).

1 Introduction

We experiment with two ideas: TextTiling and ELMo. TextTiling is an algorithm that partitions a document into subtopical segments (Hearst, 1994). ELMo is a word representation that is able to model the semantics of natural language (Peters et al., 2018). The original TextTiling algorithm described by Hearst computes similarity scores between adjacent blocks of text with a bag of words approach. Here, instead of using bag of words to represent a block of text, we use the average of the ELMo vectors for each word in the block.

We modify Hearst's original algorithm in several ways, which are inspired by Reynar (Reynar, 1998). For clarity, we call this version **TTY**. Hearst uses a construct called the depth score, while we use the minimum similarities to determine boundaries between adjacent blocks. Hearst recommends certain values for the parameters, and we experiment with how these affect the results of the modified algorithm. For clarity, we call the modified version with ELMo **TTE**.

The metric that Hearst uses is precision and recall. Some have come up with better metrics. Many others have evaluated the original TextTiling algorithm with them. Here, we use three metrics. P_k and WindowDiff appear in the literature often (Beeferman et al., 1999; Pevzner and Hearst, 2002). The more recent one is *boundary similarity* (Fournier, 2013b). We use the one-minus of it for consistency across the metrics. All of these metrics are error probabilities. To be clear, the lower the metric score a method receives, the more accurate the method.

2 TTY Implementation

Although others have already implemented the original TextTiling algorithm and its variants, we want to implement our own variation in order to understand it more in depth. Given a list of sentences, the algorithm will mark boundaries between sentences in order to segment the text. The goal is for each segment to contain a single topic.

Text Normalization: First, we split the document into tokens using NTLK's tokenzier. We convert each token to lowercase, remove all punctuation, and keep the tokens that consist only of letters. We apply the Porter Stemmer to the tokens and take the stems. Later on, we use a bag of words model to calculate similarity. The reason we apply this cleaning phase is to improve the results of the bag of words model.

We use NLTK's list of English stopwords and remove all stopwords from the list of stems. We call this resulting list of tokens the *normalized list* and write it as $T = [t_0, t_1, \ldots, t_n]$. This step also helps the bag of words model. Almost meaningless words (e.g. a, the) that appear very frequently will throw off the model by assigning high similarity to unrelated blocks of text that simply have high counts of the word *the*, for example.

Similarity Computation: Define a *pseudosentence* to be a list of w consecutive tokens. w is a free parameter of the algorithm. We partition the normalized list into contiguous pseudosentences and discard any leftover tokens. We call this list $P = [p_0, p_1, \ldots, p_m]$.

For example, if $T = [t_0, t_1, \ldots, t_{20}]$, and we

choose $w = 4$, we have

$$p_0 = [t_0, \ldots, t_3],\ p_1 = [t_4, \ldots, t_7],\ \ldots$$

and so on. We discard t_{20} because it does not fit into another whole pseudosentence.

Define a *block* to be a list of q consecutive psuedosentences. We slide two adjacent blocks along P. At every step, we move a single pseudosentence in distance and calculate a similarity score. It is important to note that we do not calculate the similarity unless both blocks fit entirely on P. Call this resulting list of similarities $S = [s_0, s_1, \ldots, s_r]$.

Specifically, let $B_i = [p_i, \ldots, p_{i+q-1}]$. Let $V = \{v : v \in T\}$ be the document's vocabulary. Then, $b_i : V \to \{0, 1, 2, \ldots\}$ is a mapping from a token to the count of the token in B_i. We use the similarity measure given in the original paper (Hearst, 1994). To calculate s_i, we calculate $sim(B_i, B_{i+q})$, which is

$$\frac{\sum_{v \in V} b_i(v) \times b_{i+q}(v)}{\sqrt{\sum_{v \in V} b_i(v)^2 \times \sum_{v \in V} b_{i+q}(v)^2}}$$

Boundary Decision: Hearst had decided on boundaries based off changes in sequences of similarity scores with a complex system involving depth scores (Hearst, 1994). However, Reynar experimented with the idea of simply using the minimum similarities to decide on boundaries, amongst other things (Reynar, 1998). Here, we use the in-between approach of taking all the local minimums (i.e. valleys) as boundaries. We disregard s_0 and s_r, even though they may be local minimums.

3 TTE Implementation

We describe the changes to TTY that we made in order to incorporate ELMo vectors. We use the Python module `allennlp` provided by the Allen Institute of AI (Gardner et al., 2017). The module provides AllenNLP's ELMo model pre-trained on the one billion word benchmark.

Text Normalization: We still convert the word tokens to lowercase and remove all punctuation. We do not apply stemming. In the original algorithm, we used a bag of words model, and stemming was important. If two blocks had different versions of the same term (e.g. playing, played, play), then we would have liked for them to contribute to the frequency of the root (e.g. play) and increase the similarity score between the blocks. For TTE, we leave this to the semantics that ELMo vectors capture.

We use the `allennlp` module's `ElmoEmbedder` class, which returns a list of three vectors for each word. Each vector represents the output of an ELMo layer. The ELMo paper recommends either the top layer vector or a specific weighting of the three vectors for downstream tasks (Peters et al., 2018). Here, for simplicity, we use the top layer vector.

We still remove stop words, except the process is a little more involved. For each ELMo vector, we check if its corresponding word is a stopword. If it is, we remove it. In the end, we have a list of ELMo vectors. The reason for this step is the same as the one for the bag of words model in TTY. The main hypothesis is that the final averaged vector of a block will point in some direction that characterizes the block in 1024 dimensional space. We do not want this direction to be close to the *a* or *the* vectors.

Similarity Computation: The term *token* generalizes well, and here, it is an ELMo vector. Specifically, in the context of TTE, T is a list of ELMo vectors. We still break up T into pseudosentences, but now P is slightly different. Before, P was a list of lists, each with w tokens. Now, we replace with psuedosentence with the average of its tokens. To be clear, P is a list of averages of w ELMo vectors.

We still slide the pair of adjacent blocks over P. But now, within each block, we take the average of its k averaged ELMo vectors in order to get the average vector of the entire block. The similarity score between two blocks is a cosine similarity between these two averaged ELMo vectors.

Boundary Decision: We use the same method as before.

4 Evaluation

4.1 Data

Structure: The data we evaluate our algorithms on is synthetic (Choi, 2000). There are 700 documents split into four categories of ranges. Each document consists of ten *segments*. Each segment is the first L sentences taken from a randomly chosen article in the Brown corpus. L is randomly picked from the range in Table 1 that the document is in.

We know that each segment is taken completely

Range of L	$[3, 11]$	$[3, 5]$	$[6, 8]$	$[9, 11]$
# docs	400	100	100	100

Table 1: Data Distribution

from a single article, so we are able to assume that with high probability, within a document, each segment has a distinct topic. Therefore, we assume that the topic boundaries are the segment boundaries.

Effect On Boundary Decisions: One big question was whether to force hypothesized boundaries to their nearest paragraph breaks, as Hearst did for TextTiling (Hearst, 1994). That is, given a document with many paragraphs, record the locations of the paragraph breaks. Whenever the algorithm hypothesizes a boundary, replace that hypothesis with the nearest recorded paragraph break.

Here, we do not do this because we evaluate on synthetic data. By construction, each document has ten paragraphs (i.e. segments). If we were to store the paragraph breaks, our algorithm would simply return the paragraph breaks to achieve the best results. Clearly, this is pointless.

4.2 Metrics

We use three different metrics to evaluate our results. The first is P_k (Beeferman et al., 1999). We include the evaluation results under P_k here so that we are able to compare them with results from other papers.

Pevzner and Hearst pointed out the flaws of P_k and created WindowDiff, which we will abbreviate as WD (Pevzner and Hearst, 2002). We include it here as it is the stricter and better metric with respect to P_k.

Finally, we use a version of the metric invented by Fournier called *boundary similarity*, which we will abbreviate as B (Fournier, 2013a). B is a number between 0 and 1. Here, we use 1 - B, which we will abbreviate as OMB. OMB is stricter than both P_k and WD. Note that P_k, WD, and OMB are all penalizing metrics. This is so that we can be consistent with our scoring. Fournier implemented each of these metrics in his Python module `segeval`, and that is what we use here (Fournier, 2013b).

$[3, 11]$	P_k	WD	OMB
TTY(10,6)	0.3862	0.4857	0.6689
TTE(10,6)	0.3204	0.3641	0.5915
TTY(20,3)	0.3093	0.3288	0.5921
TTE(20,3)	**0.2946**	**0.3074**	**0.5784**
TTY(20,6)	0.3279	0.3423	0.6494
TTE(20,6)	0.3194	0.3258	0.6717
TTY(30,3)	0.3118	0.3164	0.6572
TTE(30,3)	0.3165	0.3204	0.6777

Table 2: Average scores on $[3, 11]$

$[3, 5]$	P_k	WD	OMB
TTY(10,6)	0.3895	0.4061	0.5761
TTE(10,6)	**0.3581**	**0.3614**	**0.5403**
TTY(20,3)	0.3833	0.3835	0.6068
TTE(20,3)	0.3786	0.3786	0.6090
TTY(20,6)	0.4294	0.4296	0.7562
TTE(20,6)	0.4469	0.4472	0.8216
TTY(30,3)	0.4256	0.4256	0.7614
TTE(30,3)	0.4428	0.4428	0.8061

Table 3: Average scores on $[3, 5]$

$[6, 8]$	P_k	WD	OMB
TTY(10,6)	0.3573	0.4603	0.6503
TTE(10,6)	0.2716	0.3112	0.5332
TTY(20,3)	0.2765	0.2951	0.5441
TTE(20,3)	**0.2514**	**0.2584**	**0.5040**
TTY(20,6)	0.3233	0.3359	0.6289
TTE(20,6)	0.3394	0.3431	0.6781
TTY(30,3)	0.3199	0.3218	0.6429
TTE(30,3)	0.3189	0.3195	0.6482

Table 4: Average scores on $[6, 8]$

$[9, 11]$	P_k	WD	OMB
TTY(10,6)	0.4140	0.6103	0.7476
TTE(10,6)	0.3317	0.4366	0.6483
TTY(20,3)	0.2988	0.3511	0.6403
TTE(20,3)	0.2578	0.2931	0.5865
TTY(20,6)	0.2765	0.3063	0.6157
TTE(20,6)	0.2277	0.2375	**0.5685**
TTY(30,3)	0.2388	0.2446	0.5901
TTE(30,3)	**0.2159**	**0.2209**	0.5704

Table 5: Average scores on $[9, 11]$

5 Results

5.1 TTY vs TTE

Accuracy: First, we evaluate our algorithms, TTY and TTE, on different window sizes w and different block sizes q. The results are in Tables 2, 3, 4, and 5. The row with TTY(w, q) is the results of the TTY algorithm with parameters w and q. The row with TTE(w, q) is the results of the TTE algorithm with parameters w and q. **We report averages of metrics scores.**

The parameters that Hearst recommended in general for TextTiling are $w = 20$ and $k = 6$ (Hearst, 1994). Choi uses these same parameters when evaluating TextTiling on his data set (Choi, 2000). Table 6 shows Choi's results.

Range of L	$[3, 11]$	$[3, 5]$	$[6, 8]$	$[9, 11]$
P_k	0.46	0.44	0.43	0.48

Table 6: Choi's Evaluation of TextTiling

For both TTY and TTE, we see improvements. In general, the modifications done to TextTiling in TTY gives a higher jump in performance than the addition of ELMo vectors in TTE. However, TTE still recieves the best scores.

In all the categories, we see that our optimal parameters disagree with what Hearst recommended. This makes sense as Hearst's recommendation was based off naturally created texts.

The general trend seems to be that as segment size ranges get bigger, the optimal parameters need to get bigger to improve performance with respect to P_k and WD. For $[3, 5]$, the best score is with parameters $w = 10$ and $q = 6$. For $[6, 8]$, parameters of the best score are $w = 20$ and $q = 3$. For $[9, 11]$, we see the optimal parameters are $w = 30$ and $q = 3$.

Another trend is that the size of the segments is important with respect to P_k and WD. $[3, 5]$ has the lowest scores out of the latter three categories. We believe this is due to the lesser number of words in these documents. The similarity scores generated by the algorithm would not have been as accurate as those in the $[9, 11]$ group, which had more words per segment overall.

The OMB metric, the one that supposedly fixes many flaws of P_k and WD, disagrees a little with the other two. Why is the OMB score best in the category $[6, 8]$, when based on the analysis above, it should be best in $[9, 11]$? This question is left for future work.

Speed: The greatest strength of TTY and TextTiling variants in general is speed. TTY takes an average of roughly 0.07 seconds to hypothesize boundaries for a document in the synthetic data set. On the other hand, it takes TTE an average of 1.5 seconds to do the same task. This time is with GPU hardware acceleration on a Google machine. The comparison of 0.05 to 1.5 is not really fair, as the hardware used is different, but it is true that TTY is much faster than TTE. This makes sense as TTE involves operations on 1024 dimensional vectors many times.

5.2 In a Bigger Context

We insert our best TTE P_k result and compare it against other algorithms' results. We present part of a summary table provided by Li (Li et al., 2018) in Table 7. **The P_k scores here are averages across all category ranges**. We only provide a part of the table because the other algorithms evaluated were neural. They were not evaluated on the entire Choi data set, since the majority of data was reserved for training (Li et al., 2018). Although using ELMo did improve the base score of TTY, TTE did not beat a majority of non-neural methods.

	P_k
H94	0.4525
TTE(20,3)	0.2956
C99	0.1050
U00	0.0775
ADDP	0.0568
TSM	0.0092
GraphSeg	0.0664

Table 7: Li's Result Summary in P_k

6 Conclusions

Currently, neural approaches are dominating all fields of NLP, text/topic segmentation included. In the results from Li that I did not post, neural methods had the best scores (Li et al., 2018). I used one of the oldest ideas in topic segmentation, TextTiling, because I was curious and wanted to experiment with a classical idea in modern eyes. ELMo has clearly shown to improve existing approaches on many tasks, such as question answering and named entity recognition (Peters et al., 2018). I

had a big hunch that it would do the same for TextTiling. But what if it didn't? I had no evidence. Although TTE is not a state of the art approach, I hope we can appreciate this experiment as further confirmation of ELMo's ability to improve a system, even if only slightly.

References

Doug Beeferman, Adam L. Berger, and John D. Lafferty. 1999. Statistical models for text segmentation. *Machine Learning*, 34(1-3):177–210.

Freddy Y. Y. Choi. 2000. Advances in domain independent linear text segmentation. In *6th Applied Natural Language Processing Conference, ANLP 2000, Seattle, Washington, USA, April 29 - May 4, 2000*, pages 26–33.

Chris Fournier. 2013a. Evaluating text segmentation. Master's thesis, University of Ottawa.

Chris Fournier. 2013b. Evaluating Text Segmentation using Boundary Edit Distance. In *Proceedings of 51st Annual Meeting of the Association for Computational Linguistics*, page to appear, Stroudsburg, PA, USA. Association for Computational Linguistics.

Matt Gardner, Joel Grus, Mark Neumann, Oyvind Tafjord, Pradeep Dasigi, Nelson F. Liu, Matthew Peters, Michael Schmitz, and Luke S. Zettlemoyer. 2017. AllenNLP: A deep semantic natural language processing platform.

Marti A. Hearst. 1994. Multi-paragraph segmentation of expository text. In *32nd Annual Meeting of the Association for Computational Linguistics, 27-30 June 1994, New Mexico State University, Las Cruces, New Mexico, USA, Proceedings.*, pages 9–16.

Jing Li, Aixin Sun, and Shafiq R. Joty. 2018. Segbot: A generic neural text segmentation model with pointer network. In *Proceedings of the Twenty-Seventh International Joint Conference on Artificial Intelligence, IJCAI 2018, July 13-19, 2018, Stockholm, Sweden.*, pages 4166–4172.

Matthew E. Peters, Mark Neumann, Mohit Iyyer, Matt Gardner, Christopher Clark, Kenton Lee, and Luke Zettlemoyer. 2018. Deep contextualized word representations. In *Proceedings of the 2018 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, NAACL-HLT 2018, New Orleans, Louisiana, USA, June 1-6, 2018, Volume 1 (Long Papers)*, pages 2227–2237.

Lev Pevzner and Marti A. Hearst. 2002. A critique and improvement of an evaluation metric for text segmentation. *Computational Linguistics*, 28(1):19–36.

J.C. Reynar. 1998. *Topic Segmentation: Algorithms and Applications*. Ph.D. thesis, University of Pennyslvania, Department of Computer Science.