TextTiling - The Hacker's View

Karan Praharaj

January 21st

As hackers, we need to implement the approach proposed in the paper [1]. To do that, we first need to check the reproducibility of the paper. We will break down the algorithm into three parts, and discuss the reproducibility of each part individually.

The TextTiling algorithm for discovering subtopic structure using term repetition has three main parts:

1. Tokenization
2. Lexical Score Determination
3. Boundary Identification

Tokenization

This can be done in a straightforward manner using any of the off-the-shelf tokenizers available, e.g. - SpaCy and NLTK. If the document contains markup information or HTML tags, this can be filtered out using Regex. After tokenization, the text is subdivided into "pseudo-sentences", with a predefined size based on number of words. This can be done with a relatively simple string-splitting script.

Lexical Score Determination

For the block-comparison algorithm, the author provides concrete steps on how to determine block size, and then compute similarity values. The lexical score for the similarity between two text blocks is calculated by taking a normalized inner product between them. One can code this computation in NumPy, which offers commonly used scientific computing functions in its package. For the plotting of the token-sequence gap numbers against similarity scores, Matplotlib, which is a Python library to create data visualizations, can be used.

The vocabulary introduction algorithm is also easily reproducible. Here we need to compute the lexical score by creating a text interval of length $w * 2$ (w is the length of the token-sequences) centered around every token-sequence gap. Next, we are to subdivide this text interval into two parts divided equally by tokens. Finally, a lexical score is determined by computing the ratio of new words in an interval divided by the length of that interval ($w * 2$). This score function can also be written in NumPy with one line of code.

Boundary Identification

In this step, we determine segment boundaries and assign depth scores, i.e. the depth of the valley (if there is one), to each token-sequence gap. This is done identically for all lexical scoring methods. The method to calculate the depth score has been delineated clearly, and is simple to implement.

The author also provides an alternative to this method of computing depth scores, which is to use the slope of the valley's sides, or the "sharpness" of the change in vocabulary. She does add that this alternative approach comes with its own caveats, especially in the cases that there are short digressions. Having said that, implementing this alternative approach for depth is not complicated from the "hacker's" vantage point.

Smoothing the Plot

The depth scoring algorithm is vulnerable to the unwanted perturbations due to small valleys. A remedy has been proposed by the author to remove these small dips, using average smoothing with a fixed window size. The specific function provided in the paper is easily implementable using a for loop, but the author does mention that the choice of the function is arbitrary, and so we may use another function in-lieu of the one given.

References

[1] Marti A. Hearst. Text tiling: Segmenting text into multi-paragraph subtopic passages. *Computational Linguistics*, 23(1):33–64, 1997.