Efficient Linear Text Segmentation Based on Information Retrieval Techniques

Roman Kern
Know-Center
Inffeldgasse 21a/II
A-8010 Graz
rkern@know-center.at

Michael Granitzer
Graz University of Technology
Knowledge Management Institute
Inffeldgasse 21a/II
A-8010 Graz
mgranitzer@tugraz.at

ABSTRACT

The task of linear text segmentation is to split a large text document into shorter fragments, usually blocks of consecutive sentences. The algorithms that demonstrated the best performance for this task come at the price of high computational complexity. In our work we present an algorithm that has a computational complexity of $O(n)$ with n being the number of sentences in a document. The performance of our approach is evaluated against algorithms of higher complexity using standard benchmark data sets and we demonstrate that our approach provides comparable accuracy.

Categories and Subject Descriptors

I.2.7 [**Artificial Intelligence**]: Natural Language Processing; H.3.3 [**Information Storage and Retrieval**]: Information Search and Retrieval

Keywords

Text Segmentation, Sliding Window, Sentence Similarity

1. INTRODUCTION

The process of linear text segmentation splits a long text into chunks of consecutive text fragments, usually blocks of sentences. There also exist hierarchical text segmentation, where documents are iteratively split into finer grained topics segments. The output of such a algorithm closely resembles the typical structure of a document that consists of chapters and multiple levels of sub-chapters down to paragraph level. The motivation for authors to insert a paragraph break is not identical to a topic shift which is detected by a typical text segmentation algorithm, as reported in [17].

There exist many use cases for applying text segmentation on a document. The most obvious of these usages arises if the boundaries between individual documents are missing, which is often the case for transcripts of spoken text. Another motivation to apply text segmentation is the assumption that especially long documents may consist of multiple topics. If each of these separate topics is detected one could exploit this in various scenarios. One example where text segmentation has improved the performance is information retrieval, for example [15]. Another field in which text segmentation has proven to be helpful is text summarization [2].

Our algorithm, which we named *TSF*[1], uses a sliding window approach to identify topic boundaries. Topics are detected based on the assumption that the distribution of words can be used as indicator for the coherence of text fragments. This property of text is called lexical coherence, where a change in vocabulary correlates with a change of topics. The performance of our algorithm is compared with state of the art text segmentation algorithms using a standard test data set. Additionally we provide an evaluation based on long documents in different languages.

The next section presents various approaches to text segmentation. In section 3 our algorithm is described. The performance of our approach is evaluated in section 4. Finally section 5 concludes the results.

2. RELATED WORK

One of the most influential algorithms that detects changes in the lexical cohesion within text is *TextTiling* [8]. This algorithm uses a sliding window over the text and at each of the positions within the text two adjacent blocks of text are compared. By representing blocks as bag-of-words the between block similarity is calculated via the well known cosine similarity. In the original version of this algorithm blocks of fixed number of terms are used, but it has already been extended to use the sentence structure of the document. The sequence of similarities build a profile which is then smoothed and the valleys are regarded as boundary candidates. *TextTiling* offers a low runtime complexity, but suffers from lower accuracy than more complex approaches as demonstrated in [5].

Many of the best performing algorithms are based on the similarity matrix of all sentences within a document. Building such a matrix has a computational complexity of $O(n^2)$, with n being the number of sentences. In [5] they transformed the similarity matrix into a rank matrix and apply divisive clustering to find the best segments. This method, which is named *C99*, has demonstrated better performance than previous methods and has been further improved by us-

[1]A basic open-source implementation is available online at http://textsegfault.sourceforge.net/

MEDES 2009 October 27-30, 2009, Lyon, France

ing Latent Semantic Indexing to build the similarity matrix [4], named *CWM1*. Starting with the similarity-distance matrix in [11] the problem of text segmentation is transformed into a problem of image segmentation. Anisotropic diffusion is applied on the matrix and then a dynamic programming method is used to find the optimal segmentation. This algorithm is named *AniDiffDynProg03* in the evaluation section. The dynamic programming technique is shared with a number of approaches that have demonstrated the best performance in the area of text segmentation. Examples are [10], [12], [19], and [18]. The last two are referred as *U00* and *MI07* in the evaluation results.

3. ALGORITHM

Our approach is similar to the *TextTiling* algorithm in many ways, but also differs in some important aspects. They share the same intuition, that lexical cohesion is an indicator for topic changes within a document. Both algorithms also share the same sliding window technique. For each position i between two adjacent sentences within the document two blocks are build. One block contains a number of sentences that precede the position and one block of sentences that succeed the current position - B_i^{pre} and B_i^{post}. The size of the block is one of two parameters that has to be provided by the user and should reflect the minimal size of a text segment to be detected. The sentences are represented as term vectors and contain either the term frequency if no weighting is applied or the result of a weighting function for each term. Thus $w_{i,t}$ refers to the weight of the term t in sentence i.

For each of the two blocks the pairwise similarities of all sentences within the blocks are calculated. The arithmetic mean of the similarity values is then used to calculate the mean *inner similarity*, where μ denotes the average of pairwise similarities of two blocks. The inner similarity value can be interpreted as the similarity level of the sentences surrounding the current position of the sliding window. It is influenced by the overall term repetition characteristics of the text and reflects for example the writing style of the author:

$$sim_i^{inner} = \frac{\mu(B_i^{pre}, B_i^{pre}) + \mu(B_i^{post}, B_i^{post})}{2}$$

Next the pairwise similarities of the sentences from one block with the sentences from the other block are calculated. Again these similarity values are averaged to give the *outer similarity*, which is an indicator how similar one block is to the other:

$$sim_i^{outer} = \mu(B_i^{pre}, B_i^{post})$$

Finally these two similarity values are put into relation to build a measure for the *dissimilarity* of the two blocks around the current position:

$$dissimilarity_i = \frac{sim_i^{inner} - sim_i^{outer}}{sim_i^{inner}}$$

This value is positive if the mean similarity between the sentences of both blocks is lower than the average similarities within the blocks. This should be the case when the current position is a true topic boundary. The maximum of 1 is reached if the outer similarity is zero, which can only happen if the blocks have no term in common. If a dissimilarity value exceeds a threshold value the corresponding position is marked as boundary candidate. This threshold is the second parameter for the algorithm and should be supplied by the user. To prevent a series of boundaries between adjacent sentences for regions of high dissimilarity, a simple lookahead heuristic is used. A candidate is selected as boundary if there is no higher dissimilarity value for the next sentence positions. The block size parameter is reused as lookahead to prevent the introduction of another parameter.

The procedure to calculate the dissimilarity differs from the *TextTiling* algorithm, where only the outer similarity is exploited. The inner similarity is not calculated and thus the similarity level of the text is approximated by the similarity of the surrounding block similarities. These are smoothed and the relative difference between surrounding similarity values serve as criteria for boundary detection. In our approach the smoothing happens implicitly by using the average of the sentence similarities. Another difference is that in the *TextTiling* algorithm all terms within a blocks are merged into one term vector, while in our approach one term vector is build per sentence.

3.1 Sentence Similarity

Calculating the similarity between two sentences builds the core of the algorithm. There exists a number of highly sophisticated algorithms that integrate external knowledge into the similarity calculation, for example [13]. In the initial version of our text segmentation algorithm we opted for a simple and efficient method - a combination of term weighting and correlation computation as used in the field of information retrieval. The similarity of two sentences (i and j) which are represented as their weighted term vectors is calculated using the cosine similarity: $sim(i,j) = \sum_t w_{i,t} w_{j,t} / \|w_i\| / \|w_j\|$

3.2 Preprocessing

Before the similarities between the sentences are calculated the terms within the sentences are first filtered and weighted. For *TextTiling* algorithm the authors remark in [9] that using the term frequency alone appears to work better than using a weighting function.

To filter out function words we followed the intuition in [11] and instead of using a fixed set of stop words we use document specific stop words. Function words can be separated from content words by their distribution within documents. For each word we calculate its degree of dispersion as proposed in [7]. Words with an even distribution within documents are considered stop words[2].

3.2.1 Term Weighting

After all stop words within a sentence are filtered out, the remaining terms are weighted. Assigning a different weight to each term should reflect the differences in semantic content of each term and thus controls its influence in the similarity between sentences. This intuition has been the motivation in the research of term significance in the area of text segmentation, for example [16] and [6].

The weighting function we used is one of many variants of the *TF IDF*. The weight for a term i is a combination of its frequency within a sentence j and the number of documents

[2]For the evaluation we used 0.6 as dispersion threshold

from a corpus (of size $docCount$) the term occurs in[3]:

$$w_{i,j} = \frac{\sqrt{termFreq_{i,j}}}{\sqrt{tokenCount_j}} \left(log(\frac{docCount}{docFreq_i + 1}) + 1 \right)$$

3.3 Extensions

3.3.1 Number of Segments is Known

If the number of segments is known in advance, the algorithm triggers two additional functionalities. At first the block size parameter is calculated using a heuristic based on the number of sentences - n - in relation to the number of segments - m - including a parameter for the maximal expected variation of the segment sizes - v - which is set to $\frac{1}{3}$ in all our experiments: $blockSize = \lfloor (1 - v)\frac{n}{m} \rfloor$

After all segment boundaries are determined only the top m are selected for the final segmentation, which in combination with a high threshold can lead to the situation that the number of found segments is lower than the true segment size. An automatic adjustment of the threshold parameter could avoid this problem, but would also increase the complexity of our approach.

3.3.2 Variable Size of Blocks

Instead of using a fixed size for both blocks we modified the basic algorithm to use a dynamic block size for the block that precedes the current position of the sliding window. The size varies between the block size set by the user and twice the block size as upper limit. The actual size is reset to the lower limit at the beginning of the document and after a segment boundary has been detected and increases with each sentence until the upper limit is reached.

3.3.3 Weighted Mean

Based on the idea that sentences close to the sentence boundary that separates the two sentence block should contribute more to the inner and outer similarity we incorporated a weighted mean into the similarity calculations. This modification did improve the performance in some scenarios, but also had an detrimental effect in other scenarios. Therefore we decided not to include the weighted mean in the version of the algorithm we used for the final evaluation.

3.4 Computational Complexity

In contrast to many other text segmentation algorithms our approach does not compute the complete sentence to sentence similarity matrix of each document. Only the similarities alongside the main diagonal are computed with the maximal block size as upper limit. Thus our method has a computational complexity of $O(nk)$, with n as number of sentences and k as block size provided by the user. The block size is a constant and small in relation to the sentence count, resulting in an effective complexity of $O(n)$. Therefore this algorithm is feasible for even huge documents that might occur in some real-world scenarios.

4. EVALUATION

To evaluate the performance of *TSF* we follow the common method of creating artificial documents that are made from concatenating several distinct documents. The boundaries between these documents serve as ground truth.

Measures like precision and recall do not honor if the algorithm places a boundary near to the true segmentation. Therefore in [1] a measure is proposed - P_k - that calculates the disagreement of the segmentation made by the algorithm and a reference segmentation. A disagreement is caused when the proposed segmentation and the reference segmentation differ according to whether they place a pair of words within the same or different segments. The distance between these two words is chosen to be half the average true segment length (k). Applied to all terms yields P_k, which can be interpreted as the probability of the overall error. They also defined the probability of a missed segment boundary - P_{miss} - and the probability of a false alarm - $P_{falsealarm}$.

Some critique on the P_k was raised in [14] where the authors pointed out that the influence on the final error rate differs between a miss and a false alarm. They introduced another measure which they named $WindowDiff$. As most of the existing evaluations of text segmentation algorithms used the P_k measure we report the results for this measure, but we also include the $WindowDiff$ results for future reference.

	3-11	3-5	6-8	9-11
TextTiling	46%	44%	43%	48%
C99	13%	18%	10%	10%
U00	11%	13%	6%	6%
TopSeg02	10.74%	7.44%	7.95%	6.65%
AniDiffDynProg03	6.0%	7.1%	5.3%	4.3%
TSF	9.0%	9.3%	6.8%	9.2%

Table 1: Probability of error P_k for all four test data sets when the number of segments is not known.

4.1 Brown Corpus Evaluation

In [5] a test data set was introduced that since then has been used multiple times to compare the performance of different text segmentation algorithms. The test data is based on the Brown corpus and consists of four test sets that vary in the average sentence count per segment. Each test document is a concatenation of 10 corpus documents drawn from two of the categories within the Brown corpus - "news" and "learned". We used the remaining 13 categories of the Brown corpus to calculate the document frequencies for the *TF IDF* weighting. Unless otherwise stated 0.7 was used as *threshold parameter* and the *block size parameter* was set to 4 sentences for all evaluation runs. For being comparible with the results from [5] we used the software package provided by the author[4] for sentence splitting, tokenization, non-word removal (e.g. punctuation) and for stemming.

4.1.1 Comparison With Other Algorithms

Table 1 compares the result of state-of-the-art text segmentation algorithms published in recent years. With the exception of *TextTiling* all algorithms have a non linear runtime complexity. The performance numbers for *TextTiling* and *C99* were taken from [5]. The other algorithms are *U00*

[3]This is the default weighting of the open-source project Lucene, see http://lucene.apache.org/java/docs/

[4]http://myweb.tiscali.co.uk/freddyyychoi/software/C99-1.2-release.tgz

	3-11	3-5	6-8	9-11
C99	12%	11%	10%	9%
U00	10%	9%	7%	5%
CWM1	9%	10%	7%	5%
AniDiffDynProg03	6.0%	6.8%	5.2%	4.3%
MI07	4.7%	5.6%	2.6%	1.6%
TSF	8.7%	7.2%	6.7%	4.8%

Table 2: Error measure P_k for the four test data sets when the number of segments is known in advance.

[19], *TopSeg01* [3] *AniDiffDynProg03* [11] and our approach. For each of the four test sets the error probability P_k is reported for the configuration when the number of segments is not known and the default settings are used.

In table 2 the results are summarized if the number of segments is available for the algorithm. In this table the performance results for *CWM1* [4] and *MI07* [18] are also included. For both configurations the performance of our algorithm is comparable with most of the published algorithms, while effient and easy to implement. The $WindowDiff$ results for the four test data sets are 10.4% 9.5%, 7.8% and 12.6% if the number of segments is not known. If the true segment count is available to the algorithm the $WindowDiff$ values are 9.6%, 7.6%, 7.2% and 4.9%.

4.2 Reuters RCV1 & RCV2 Evaluation

Additionally to the evaluation based on the Brown corpus we also conducted a series of evaluations based on other corpora, the Reuters RCV1 and RCV2. The RCV1 corpus is a set of over 800000 English news stories from August 1996 to August 1997. The Reuters RCV2 corpus is built from the same timespan, but in thirteen different languages. This offers the possibility to assess the performance of the text segmentation algorithm in different languages. Another advantage of using these corpora is that the statistics used to calculate the *TF IDF* are build with documents that more similar to the test documents, as this is the case with the Brown corpus. For the Brown evaluation the documents used to generate the statistics differ in genre from the documents used for testing.

4.2.1 Test Document Generation

At first the documents from each language version are split into two parts, one to build the term statistics and one for testing. We used all documents from the year 1996 for the term statistics and all documents from 1997 as candidates for the test document generation. Following [8] only documents with at least 10 sentences are used for the test document generation, resulting in a average sentence length of about 17 per document. From these candidates 100 test documents where build by concatenating a varying number of randomly drawn documents. On average a test document consists of about 120 Reuters documents, with a minimum of 2 and a maximum of 437[5].

4.2.2 Parameters

[5]The ids of the exact split and a description of the procedure to remove boundary indicators, for example copyright messages, are available online: http://textsegfault.sourceforge.net/reuters-split.html

Algorithm	P_k	$WindowDiff$
C99	14.4%	18.3%
C99*	10.2%	11.4%
TSF	5.3%	6.2%
TSF*	4.1%	4.8%

Table 3: Performance comparison on the English test documents. The asterisks indicate that the number of segments is known in advance.

We used the open-source OpenNLP software package[6] for tokenization and sentence splitting. Unfortunately only the models for the English, German and Spanish language are available online. Therefore we were limited to these languages. Stemming is done using the Snowball stemmer[7].

All parameter settings were the same as in the Brown evaluation runs, with the exception of the number of sentences used to build one block. This setting was changed from 4 to 8 to reflect the increase of size of the test document length.

4.2.3 Performance Comparison

The first test run using the Reuters RCV1 corpus compares the performance of the C99 algorithm with our approach[8], see table 3. As in the Brown corpus evaluation our approach is able to outperform the C99 algorithm. The difference is even more pronounced for the Reuters documents. Because of the larger window size of 8 sentences the similarity measure between the sentences is more robust which leads to an improved performance.

4.2.4 Multilingual Performance

The final test run compares the performance of our text segmentation algorithm between different languages. Figure 1 depicts the P_k error measure for the language English, German and Spanish. One can observe a severe drop in performance between the different languages. Still the worst case performance of our approach yields performance figures similar to that of the C99 for the English documents. Closer inspection of the additional error measures P_{miss} and $P_{falsealarm}$ reveals that the used threshold is not optimal for the languages other than English. But even if an optimal threshold for each language is found, the performance will still be best for English documents, as both error measures are lower than in any the other languages.

The usage of noun-noun word-compounds in the German language could be one reason for the observed difference and splitting these word could increase the performance. In comparison with German and English which both stem from the Germanic family of languages, Spanish documents yield the worst performance. Whether this is the case for other Romance languages, like for example French, and what could be possible steps to close the gap between the different languages are still open questions.

5. CONCLUSIONS

Linear text segmentation of long documents can be helpful in a number of use cases. If the documents that are

[6]http://opennlp.sourceforge.net/

[7]http://snowball.tartarus.org/

[8]We selected the C99 as reference as the source code is publicly available

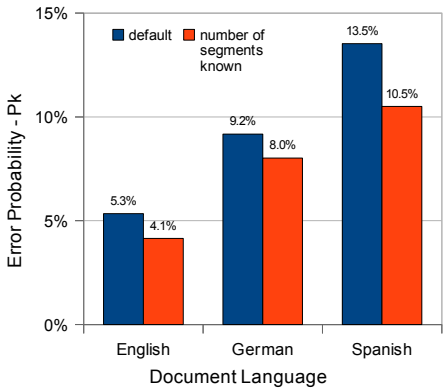

Figure 1: Performance of the text segmentation for different languages as measured by the error probability P_k.

to be splitted consist of a large number of sentences the runtime behavior of the algorithm becomes an increasingly important aspect. Another criteria for selecting a text segmentation algorithm is the quality of the segmentation. We presented an algorithm that combines a computational complexity of $O(n)$ with a performance that is similar to algorithms of higher complexity. We did our evaluations on a data set which has already been used in the past to compare many different approaches. Additionally we provided an evaluation that demonstrates the performance for longer documents in English and other languages.[9]

6. REFERENCES

[1] D. Beeferman, A. Berger, and J. Lafferty. Statistical models for text segmentation. In *Machine Learning*, pages 177–210, 1999.

[2] B. Boguraev, M. Neff, I. Center, and N. Yorktown Heights. Discourse segmentation in aid of document summarization. In *System Sciences, 2000. Proceedings of the 33rd Annual Hawaii International Conference on*, page 10, 2000.

[3] T. Brants, F. Chen, and I. Tsochantaridis. Topic-based document segmentation with probabilistic latent semantic analysis. In *Proceedings of the eleventh international conference on Information and knowledge management*, pages 211–218. ACM New York, NY, USA, 2002.

[4] F. Choi, P. Wiemer-Hastings, and J. Moore. Latent semantic analysis for text segmentation. In *In Proceedings of EMNLP*, 2001.

[5] F. Y. Y. Choi. Advances in domain independent linear text segmentation, 2000.

[6] G. Dias and E. Alves. Unsupervised topic segmentation based on word co-occurrence and multi-word units for text summarization. In *Proceedings of the ELECTRA Workshop associated to 28th ACM SIGIR Conference, Salvador, Brazil*, pages 41–48, 2005.

[7] S. T. Gries. Dispersions and adjusted frequencies in corpora. *International Journal of Corpus Linguistics*, 13(4):403–437.

[8] M. Hearst. TextTiling: Segmenting text into multi-paragraph subtopic passages. *Computational Linguistics*, 23(1):33–64, 1997.

[9] M. A. Hearst. Multi-paragraph segmentation of expository text, 1994.

[10] O. Heinonen. Optimal multi-paragraph text segmentation by dynamic programming. In *ANNUAL MEETING-ASSOCIATION FOR COMPUTATIONAL LINGUISTICS*, volume 36, pages 1484–1486. ASSOCIATION FOR COMPUTATIONAL LINGUISTICS, 1998.

[11] X. Ji and H. Zha. Domain-independent text segmentation using anisotropic diffusion and dynamic programming. In *In Proceedings of SIGIR*, pages 322–329. ACM Press, 2003.

[12] A. Kehagias, F. Pavlina, and V. Petridis. Linear text segmentation using a dynamic programming algorithm. In *Proceedings of the tenth conference on European chapter of the Association for Computational Linguistics, April*, pages 12–17, 2003.

[13] Y. Li, D. McLean, Z. Bandar, J. O'Shea, and K. Crockett. Sentence similarity based on semantic nets and corpus statistics. *IEEE Transactions on Knowledge and Data Engineering*, pages 1138–1150, 2006.

[14] L. Pevzner and M. A. Hearst. A critique and improvement of an evaluation metric for text segmentation. *Computational Linguistics*, 28:1–19, 2002.

[15] J. Reynar. Statistical models for topic segmentation. In *Proceedings of the 37th annual meeting of the Association for Computational Linguistics on Computational Linguistics*, pages 357–364. Association for Computational Linguistics Morristown, NJ, USA, 1999.

[16] K. Richmond and A. Smith. Detecting subject boundaries within text: A language independent statistical approach. In *Brown University, Providence, Rhode Island*, pages 47–54, 1997.

[17] C. Sporleder and M. Lapata. Broad coverage paragraph segmentation across languages and domains. *ACM Trans. Speech Lang. Process*, 3:1–35, 2006.

[18] B. Sun, P. Mitra, C. Giles, J. Yen, and H. Zha. Topic segmentation with shared topic detection and alignment of multiple documents. In *Proceedings of the 30th annual international ACM SIGIR conference on Research and development in information retrieval*, pages 199–206. ACM New York, NY, USA, 2007.

[19] M. Utiyama and H. Isahara. A statistical model for domain-independent text segmentation. In *ANNUAL MEETING-ASSOCIATION FOR COMPUTATIONAL LINGUISTICS*, volume 39, pages 491–498, 2001.

[9]The Know-Center is funded within the Austrian COMET Program - Competence Centers for Excellent Technologies - under the auspices of the Austrian Federal Ministry of Transport, Innovation and Technology, the Austrian Federal Ministry of Economy, Family and Youth and by the State of Styria. COMET is managed by the Austrian Research Promotion Agency FFG.