Removing Manually Generated Boilerplate from Electronic Texts: Experiments with Project Gutenberg e-Books

Owen Kaser
University of New Brunswick

Daniel Lemire
Université du Québec à Montréal

July 12, 2007

Abstract

Collaborative work on unstructured or semi-structured documents, such as in literature corpora or source code, often involves agreed upon templates containing metadata. These templates are not consistent across users and over time. Rule-based parsing of these templates is expensive to maintain and tends to fail as new documents are added. Statistical techniques based on frequent occurrences have the potential to identify automatically a large fraction of the templates, thus reducing the burden on the programmers. We investigate the case of the Project Gutenberg^{TM} corpus, where most documents are in ASCII format with preambles and epilogues that are often copied and pasted or manually typed. We show that a statistical approach can solve most cases though some documents require knowledge of English. We also survey various technical solutions that make our approach applicable to large data sets.

1 Introduction

The Web has encouraged the wide distribution of collaboratively edited collections of text documents. An example is Project Gutenberg[1] [14] (hereafter PG), the oldest digital library, containing over 20,000 digitized books. Meanwhile, automated text analysis is becoming more common. In any corpus of unstructured text files, including source code [2], we may find that some uninteresting "boilerplate" text coexists with interesting text that we wish to process. This problem also exists when trying to "scrape" information from Web pages [8]. We are particularly interested in cases where no single template generates all text files — rather, there is an undetermined number and we do not initially know which template was used for a particular file. Some templates may differ only in trivial ways, such as in the use of white space, while other differences can be substantial — as is expected when distributed teams edit the files over several years.

Ikeda and Yamada [10] propose "substring amplification" to cluster files according to the templates used to generate them. The key observations are that chunks of text belonging to the template appear repeatedly in the set of files, and that a suffix tree can help detect the long and frequent strings.

Using this approach with PG is undesirable since the suffix array would consume much memory and require much processing: the total size of the files is large and growing. Instead, we should use our domain knowledge: the boilerplate in PG is naturally organized in lines and only appears at the beginning or end of a document. We expect to find similar patterns in other hand-edited boilerplate.

1.1 Related Work

Stripping unwanted and often repeated content is a common task. Frequent patterns in text documents have been used for plagiarism detection [17], for document fingerprinting [15], for removing templates in HTML documents [6], and for spam detection [16]. Template detection in HTML pages has been shown to improve document retrieval [4].

[1]Project Gutenberg is a registered trademark of the Project Gutenberg Literary Archive Foundation.

The specific problem of detecting preamble/epilogue templates in the PG corpus has been tackled by several hand-crafted rule-based systems [1, 3, 9].

2 Stripping PG

In PG e-books, there is a preamble that provides various standard metadata. Following the transcribed body of the book, there is frequently an epilogue. We want an automated solution to remove the preamble and epilogue.

The desired preambles and epilogues used in PG e-book files have changed several times over the years, and they may change again in future. This makes fully hand-crafted PG parsers [1, 3, 9] an unsatisfactory solution. The best way to obtain a robust solution is to use methods that automatically adjust to changes in data.

3 Algorithm

Our solution identifies frequent lines of text in the first and last sections of each file. These frequent lines are recorded in a common data structure. Then, each file is processed and a sequence of GAP_MAX infrequent lines is used to detect a transition from a preamble to the main text, and one from the main text to an epilogue. A technical report [12] gives details.

3.1 Classification-Error Effects

Two types of errors may occur when trying to identify the preamble from line frequencies. If a sequence of false negatives occurs within the preamble, then we may cut the preamble short. In the simplistic analytic model where false negatives occur with probability σ, the expected number of lines before GAP_MAX false negatives are encountered is $\sum_{k=1}^{\mathrm{GAP\_MAX}} \sigma^{-k}$ which is 1.4 million lines for $\mathrm{GAP\_MAX} = 10$ and $\sigma = 0.25$.

If some false positives occur shortly after the preamble, then we may overestimate the size of the preamble. Let p denote the probability of a false positive. The expected number of misclassified lines following the preamble is $\sum_{k=1}^{\mathrm{GAP\_MAX}} (1-p)^{-k} - \mathrm{GAP\_MAX}$. With $\mathrm{MAX\_GAP} = 10$, this is small for $p \leq 20\%$.

3.2 Data Structures

The algorithm's first pass builds a data structure to identify the frequent lines in the corpus. Several data structures are possible, depending whether we require exact results and how much memory we can use. One approach that we do *not* consider in detail is taking a random sample of the data. If the frequent-item threshold is low (say $K = 5$), too small a sample will lead to many new false negatives. However, when K is large, sampling might be used with any of the techniques below.

3.2.1 Exact Counts Using Internal Memory

For exact results, we could build a hash table that maps each line seen to an occurrence counter. We need about 700 MiB for our data structure.

3.2.2 Exact Counts Using External Memory

To know exactly which lines occur frequently, if we have inadequate main memory, an external-memory solution is to sort the lines. Then a pass over the sorted data can record the frequent lines, presumably in main memory.

3.2.3 Checksumming

For nearly exact results, we can hash lines to large integers, assuming that commonly used hashing algorithms are unlikely to generate many collisions. We chose the standard CRC-64 checksum, and a routine calculation [18] shows that with a 64-bit hash, we can expect to hash roughly $2^{64/2}$ distinct lines before getting a collision.

3.2.4 Hashing to Millions of Counters

To use even less memory than CRC-64 hashing, one solution is to use a smaller hash (e.g., a 23-bit hash) and accept some collisions. Once the range of the hash function is small enough, it can directly index into an array of counters, rather than requiring a lookup in a secondary structure mapping checksums to counters.

In our experiments on the first PG DVD, we process only files' tops and bottoms, and we use a 23-bit hash with the 3.4 million distinct lines. Assume that hashing distributes lines uniformly and independently across the counters. Then the probability that a randomly selected infrequent line will share a counter with one of the ≈ 3000 frequent lines is estimated as $\approx 3000 \times 2^{-23} = 3.6 \times 10^{-4}$. These few additional false positives should not be harmful.

It is more difficult to assess the additional false positives arising when a collection of infrequent lines share a counter and together have an aggregate frequency exceeding the frequent-item threshold, K. By assuming that the line frequency distribution is very skewed and lines are frequent with a small probability p, we have derived [12] that the probability of a false positive is less than $p^{(n-1)/c}$ where n is the number of distinct lines and c is the number of counters ($c = 2^{23}$). This was verified experimentally.

3.2.5 Tracking Hot Items

Many algorithms have been developed for detecting "frequent items" in streams. In such a context, we are not interested in counting how many times a given item occur, we only want to retrieve frequent items. Cormode and Muthukrishnan survey some of them [5].

A particularly simple and fast deterministic method, *Generalized Majority* (GM), has been developed independently by several authors [7, 11, 13]. GM uses c counters, where $c \geq 1/f - 1$ and f is the minimum (relative) frequency of a frequent item; in the special case where $f = 1/2$, a single counter is sufficient. In the case where the distribution is very skewed, the algorithm already provides a good approximation of the frequent items: it suffices to keep only the items with the largest count values. We believe this observation is novel.

3.3 Heuristic Improvements

A large majority of PG e-books can have their preambles and epilogues detected by a few heuristic tricks. Our heuristics [12] were implemented using regular expressions.

4 Experimental Results

We implemented the data structures discussed in § 3.2 in Java 1.5 (using Sun's JDK 1.5.0) and tested them on a older machine with Pentium 3 Xeon processors (700 MHz with 2 MiB cache) and 2 GiB of main memory.

4.1 Errors

Looking at epilogues, the choice of data structure did not have much effect on accuracy. For preambles, the GM approach had moderately higher errors in about 30% of the cases. However, this always involved 10 or fewer lines.

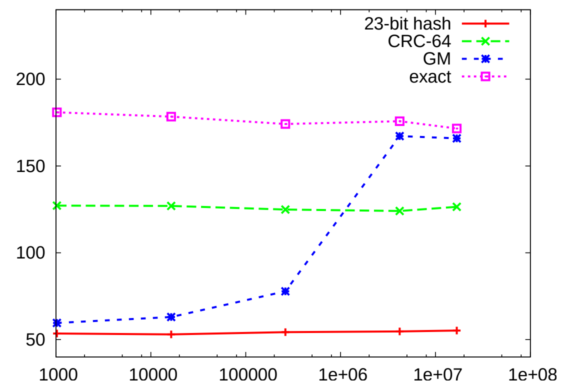

Figure 1: Wall-clock times (s) vs. the number of counters, c.

Comparing results on preamble detection, the heuristics were somewhat helpful, but GM still had difficulties compared to using exact counts in about 30% of the cases.

4.2 Run Times

We had our data structures process all tops and bottoms of the files on the first PG DVD. Experiments considered a range of values for c, the number of counters used. For each data point, ten trials were made and their average is shown in Fig. 1.

GNU/Linux shell utilities, presumably highly optimized, could sort and build the list of frequent lines in under 100 s.

4.3 Comparison to GutenMark

Of those software tools that reformat PG e-books, it appears only GutenMark [3] formally attempts to detect the preamble, so it can be stripped. We used its most recent production release, dated 2002, when PG e-books did not have a long epilogue. Thus we can only test it on preamble detection.

Despite several large errors compared to our approach, in many cases the GutenMark approach worked reasonably well.

5 Conclusion

Detecting the PG-specific preambles and epilogues is maybe surprisingly difficult. There are instances where a human without knowledge of English probably could not accurately determine where the preamble ends. Nevertheless, our approach based on line frequency can approximately (within 10%) detect the boilerplate in more than 90% of the documents.

Line frequency follows a very skewed distribution and thus, as we have demonstrated, hashing to small number of bits will not lead to a large number of lines falsely reported as frequent. Indeed, using 23-bit line hashing, we can approximately find the frequent lines, with an accuracy sufficient so that preamble/epilogue detection is not noticeably affected. Simple rule-based heuristic can improve accuracy in some cases, as observed with epilogues.

About the Authors

Owen Kaser holds a BCSS from Acadia U. and an MS and Ph.D. from SUNY Stony Brook.

Daniel Lemire received his B.Sc. and M.Sc. from the U. of Toronto and a Ph.D. from the École Polytechnique de Montréal.

References

[1] T. Atkins. Newgut program. online: `http://rumkin.com/reference/gutenberg/newgut`, 2004. last checked 18-01-2007.

[2] D. C. Atkinson and W. G. Griswold. Effective pattern matching of source code using abstract syntax patterns. *Softw., Pract. Exper.*, 36(4):413–447, 2006.

[3] R. S. Burkey. GutenMark download page. online: `http://www.sandroid.org/GutenMark/download.html`, 2005. last checked 18-01-2007.

[4] L. Chen, S. Ye, and X. Li. Template detection for large scale search engines. In *SAC '06*, pages 1094–1098, 2006.

[5] G. Cormode and S. Muthukrishnan. What's hot and what's not: tracking most frequent items dynamically. *ACM Trans. Database Syst.*, 30(1):249–278, 2005.

[6] S. Debnath, P. Mitra, and C. L. Giles. Automatic extraction of informative blocks from webpages. In *SAC '05*, pages 1722–1726, 2005.

[7] E. D. Demaine, A. López-Ortiz, and J. I. Munro. Frequency estimation of internet packet streams with limited space. In *Proceedings of ESA-2002, LNCS 2461*, pages 348–360. Springer-Verlag, 2002.

[8] D. Gibson, K. Punera, and A. Tomkins. The volume and evolution of web page templates. In *WWW '05*, pages 830–839, 2005.

[9] J. Grunenfelder. Weasel reader: Free reading. online: `http://gutenpalm.sourceforge.net/`, 2006. last checked 18-01-2007.

[10] D. Ideda and Y. Yamada. Gathering text files generated from templates. In *IIWeb Workshop, VLDB-2004*, 2004.

[11] R. M. Karp, S. Shenker, and C. H. Papadimitriou. A simple algorithm for finding frequent elements in streams and bags. *ACM Trans. Database Syst.*, 28(1):51–55, 2003.

[12] O. Kaser and D. Lemire. Removing manually generated boilerplate from electronic texts: Experiments with project gutenberg e-books. Technical Report TR-07-001, Dept. of CSAS, UNBSJ, 2007. available from `http://arxiv.org/abs/0707.1913`.

[13] J. Misra and D. Gries. Finding repeated elements. *Sci. Comput. Program.*, 2(2):143–152, 1982.

[14] Project Gutenberg Literary Archive Foundation. Project Gutenberg. `http://www.gutenberg.org/`, 2007. checked 2007-05-30.

[15] S. Schleimer, D. Wilkerson, and A. Aiken. Winnowing: local algorithms for document fingerprinting. In *SIGMOD'2003*, pages 76–85, 2003.

[16] R. Segal, J. Crawford, J. Kephart, and B. Leiba. SpamGuru: An enterprise anti-spam filtering system. In *Proceedings of the First Conference on E-mail and Anti-Spam*, 2004.

[17] D. Sorokina, J. Gehrke, S. Warner, and P. Ginsparg. Plagiarism detection in arxiv. In *ICDM '06: Proceedings of the Sixth International Conference on Data Mining*, pages 1070–1075, Washington, DC, USA, 2006. IEEE Computer Society.

[18] Wikipedia. Birthday paradox — Wikipedia, the free encyclopedia, 2007. [Online; accessed 18-01-2007].