Malformed UTF-8 and Spam

Matt Crane
Department of Computer Science
University of Otago
Dunedin, New Zealand
mcrane@cs.otago.ac.nz

Andrew Trotman
Department of Computer Science
University of Otago
Dunedin, New Zealand
andrew@cs.otago.ac.nz

Richard O'Keefe
Department of Computer Science
University of Otago
Dunedin, New Zealand
ok@cs.otago.ac.nz

ABSTRACT

In this paper we discuss some of the document encoding errors that were found when scaling our indexer and search engine up to large collections crawled from the web, such as ClueWeb09. In this paper we describe the encoding errors, what effect they could have on indexing and searching, how they are processed within our indexer and search engine and how they relate to the quality of the page measured by another method.

Categories and Subject Descriptors

H.3.1 [**Information Search and Retrieval**]: Content Analysis and Indexing – Indexing methods; H.3.3 [**Information Search and Retrieval**]: Information Filtering; H.3.0 [**Information Search and Retrieval**]: General

Keywords

Information Retrieval, Web Documents, Errors, Procrastination

1. INTRODUCTION

Scaling up a search engine from smaller, controlled, collections such as the Wall Street Journal and Wikipedia collections to collections that are larger in size and drawn from an uncontrolled domain like the web, such as the ClueWeb09 collection, leads to a set of new problems as old assumptions no longer hold.

Documents from the general web contain issues not seen in pre-cleaned TREC-like collections, for instance invalid encoding of content and spam.

In this paper we will describe and discuss some of these encoding issues, the effect that they can have on indexing and search, and how they are processed within the ATIRE search engine [8]. We discuss the relationship between the class of errors and the spamminess of the page, and the potential combination of the detection of these errors with spam filtering.

ADCS'13, December 05–06, 2013, Brisbane, QLD, Australia

http://dx.doi.org/10.1145/2537734.2537746.

2. UNICODE & UTF-8

Unicode is a standard that defines a set of codepoints for characters, a set of encoding methods and rules for the normalization, decomposition, lower- and upper-casing etc. of these characters. Each unicode character has a unique codepoint, and a name. For instance, `WHITE SMILING FACE`, or `U+263A` where `263A` is the codepoint in hex. Currently the ATIRE search engine implements Unicode version 6.0 [3].

One of the most commonly used encodings for Unicode is UTF-8, a variable byte encoding that maintains backwards compatibility with ASCII, which is the most used encoding on the web[6].

In each UTF-8 encoded sequence, the high bits within the first byte determine how long the sequence is, while the remainder are identified as continuation bytes — identified by the bit pattern: `10xxxxxx`. Figure 1 shows some of these features for a set of UTF-8 encoded sequences.

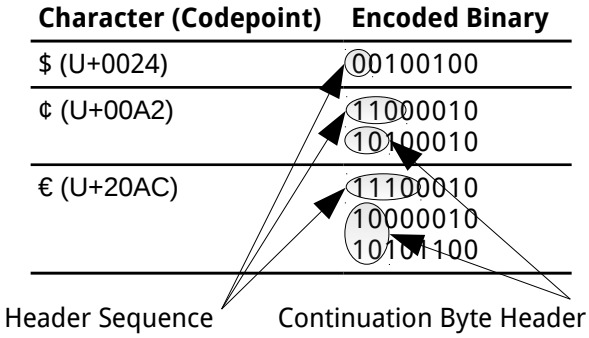

Figure 1: Diagram of encoding features of UTF-8

For instance, € (`EURO SIGN`, `U+20AC`), is encoded as follows. First, the codepoint (`20AC`) gets converted to binary: `10000010101100`. Because this is larger than 11 bits, but less than 17, the resultant encoding is a 3 byte sequence. The codepoint is padded with 0s to extend it to the necessary 16 bits for a 3 byte sequence.

Because the encoding is 3 bytes long, the first byte starts with `1110`, followed by the first four bits of the code point: `0010`. The second, and third, bytes then start with `10` as they are continuation bytes, followed by the next six bits of the code point: `10000010` and `10101100` respectively. The final, correct, encoding is then: `11100010 10000010 10101100`.

2.1 Web Data

Despite the assurances of the ClueWeb09 distributors — "English content is encoded in UTF-8 format (where proper UTF-8 character encodings apply)"[1] — the documents contained in the English section contain invalid UTF-8 encoded data. There are a number of issues we observe in the ClueWeb09 collection that needed to be handled by our UTF-8 parser:

Unexpected continuation bytes: continuation bytes are the non-first bytes that make up an encoded character. They are identified by having the high bit in the byte set to 1, and are unexpected when they do not follow the correct head-byte. Figure 2 shows an example of an unexpected continuation byte after a correctly encoded sequence (€).

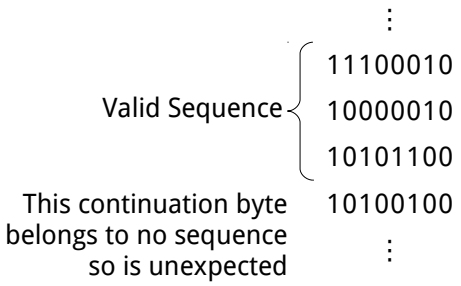

Figure 2: An unexpected continuation byte

Not enough continuation bytes: the first byte in a UTF-8 encoded sequence identifies how many bytes are contained in the sequence. This error occurs when the sequence is identified as containing a given number of bytes, but then this is not followed by the sufficient number continuation bytes. Figure 3 shows an example of a missing continuation byte, where the third byte for € is missing from the encoding.

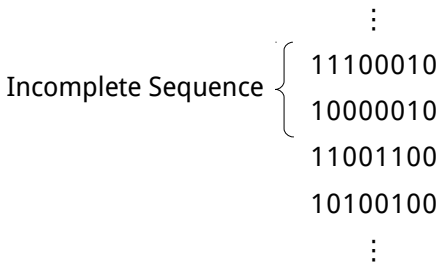

Figure 3: Not enough continuation bytes

This error is inherent in the ClueWeb09 corpus because pages were truncated at 10MB of content.

Invalid surrogate halves: the characters `U+D800`–`U+DFFF` have been defined as not legal unicode values, and so their respective UTF-8 encodings are invalid. These characters were originally used in UTF-16.

Invalid 4-, 5- and 6-byte sequences: the original specification allowed sequences of up to 6-bytes, allowing for codepoints up to 31 bits in length. However, this was later restricted by `RFC 3629` [9] to match the limits of UTF-16, with codepoints ending at `U+10FFFF`, removing 5-, 6- and about half of the 4-byte sequences.

Over-long encodings: by adding padding 0s at the beginning of the codepoint to be defined, the resultant encoding is longer than is necessary. The UTF-8 standard specifies that the smallest encoding is the correct, and only correct, encoding. Figure 4 shows an example over-long encoding of the € symbol, to result in a four byte encoding, whereas the valid encoding is only three bytes long.

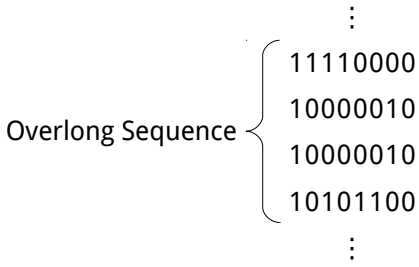

Figure 4: Over-long encoding

The over-long encoding of the NUL character was used in Modified UTF-8 [2], a variant used by the Java programming language, as a way to allow for a string containing the NUL character to be processed using traditional null-terminated string functions. These are the only class of errors that are non-trivial to correct, by replacing the over-long encoding with the correct encoding.

2.2 Error Frequencies

We investigated the frequency with which these different errors occurred within ClueWeb09 Category A, a collection of 500 million English documents crawled from the web. Table 1 provides the total number of times each error occurred, with the first column showing the error type and the second the number of times it occurred, for instance an over-long encoding of a non-ASCII character occurred 722,726 times.

Error	Occurrences
Unexpected continuation byte	1,062,303,975
Not enough continuation bytes	880,735,432
Invalid surrogate halves	78,378,657
Invalid 4-, 5- and 6-byte sequences	1,752,814,236
Over-long NUL encoding	20,374,475
Over-long ASCII (non-NUL) encoding	11,496,910
Non-ASCII over-long encoding	722,726

Table 1: Total occurrences of the various encoding errors in the ClueWeb09 corpus

We consider three cases of over-long encoding: an ASCII character, a NUL character and another unicode character. It is particularly interesting to see the large number of over-long NUL encodings. We suspect this is due to an uptake of Modified UTF-8, where the over-long encoding of NUL is permitted.

2.3 Spam

We next considered the various types of errors found in pages, and the spamminess of the page, we hypothesised that the more spammy a page was, then the more encoding errors it would contain. The spam scores generated for ClueWeb09 by Cormack *et al.* [4] were used.

Figure 5 shows the proportion of documents for each spamminess level that contain each of the different encoding errors. For most classes of error, the proportion of documents containing these errors increases as the spamminess

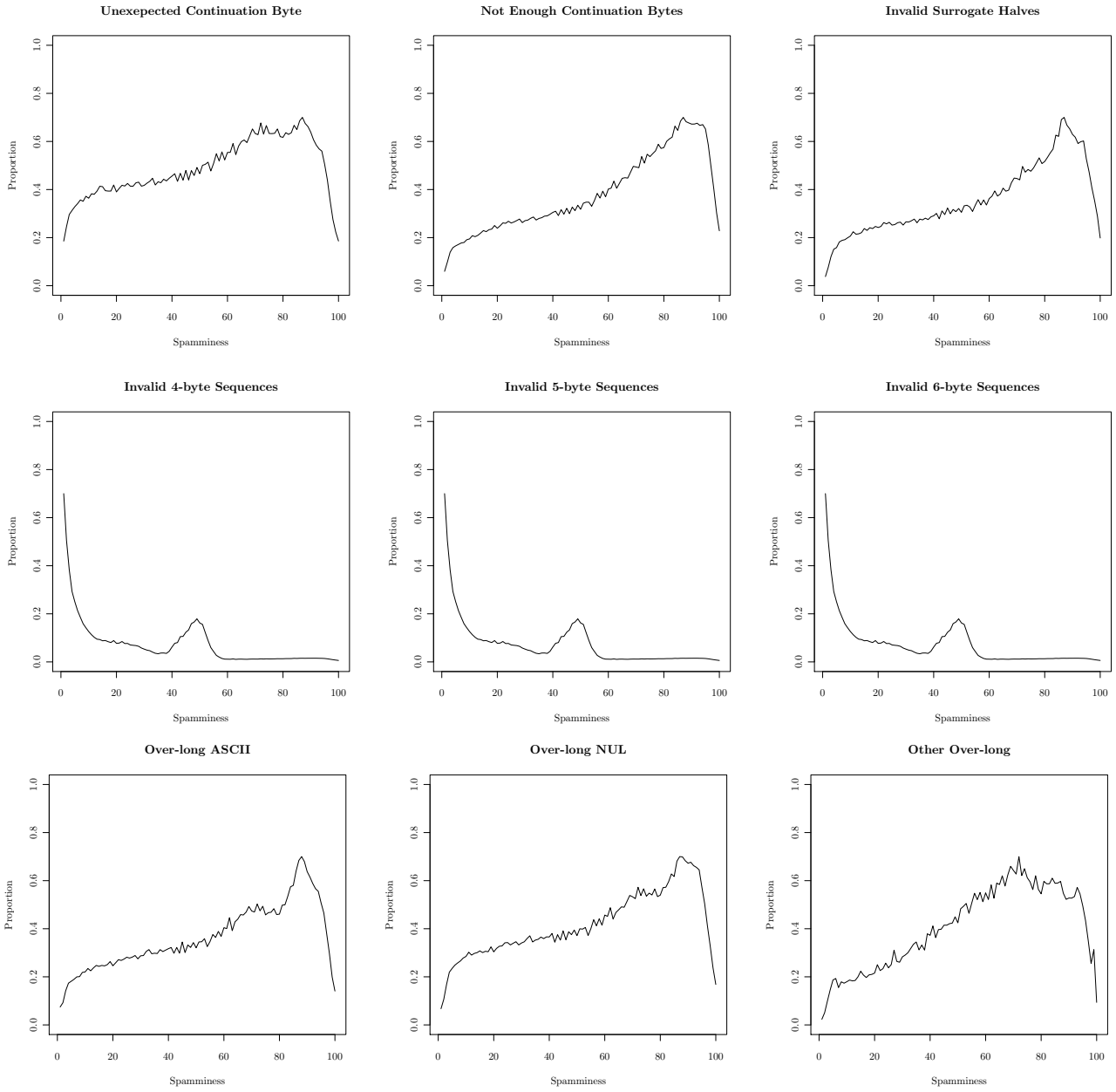

Figure 5: Cumulative percentage occurrences of various encoding errors as a function of the spamminess of the page

of the page increases, until the page is part of approximately the top 10% of spammy pages, when the proportion drops to the same levels as the least-spammy documents.

This drop in encoding errors is explainable to some extent by the fact that the spam page has to be presented to a user. It is likely that if a page contained a large number of encoding errors, then the page would be identified as spam by the user. Probably why the most spammy pages are constructed with the same care and attention to correct encodings as the least spammiest pages.

The prevalence of the invalid 4-byte sequences in the least spammy documents and then the proceeding bump in the middle-range documents is of particular interest. A preliminary investigation at some of the affected documents from within the Wikipedia suggests that these documents are English documents about non-English people and places, resulting in bad encodings of the non-English characters.

We performed t-tests for each error class to determine if there was a difference in distributions across spamminess levels for documents that contain each error, compared to those that didn't. The only class of errors that showed any significance at the $p < 0.01$ level was the invalid 4-, 5- and 6-byte sequences. This suggests that we can use this class of errors to determine whether a document is spam. If so then we can avoid indexing it. We leave for future work to determine the effect on precision and throughput.

2.4 Indexing

The unicode standard states that upon encountering an ill-formed sequence such as those described, that the decoder should consider this an error case, and that a valid parser should neither interpret or emit an ill-formed code unit. Ob-

viously, error-ing out and ceasing to process any further documents is not ideal for an indexer, and so we choose to clean the data when possible.

A common approach to dealing with these ill-formed sequences is to replace them with another appropriate character. The unicode specified `REPLACEMENT CHARACTER`, `U+FFFD`, is the suggested replacement by the unicode standard [7]. However, as the encoding of this replacement could be larger than the incorrectly encoded sequence (in fact from Table 1, we can conclude this is the most common case) and to avoid unnecessary copying (the character is not considered to be a letter or digit in unicode and would not be included in the terms) we did not try this approach.

Another approach is to interpret the invalid bytes as belonging to another encoding, such as ISO-8859-1 or CP1252. For example, the invalid UTF-8 sequence `0x80` would be interpreted as the euro sign if interpreted as being encoded using CP1252. However, we also did not try this approach as which other encoding the bytes should be interpreted as belonging to is a point of contention.

The approach taken inside the ATIRE search engine is to replace the bytes belonging to invalid sequences with their correct encodings in the case of an over-long encoding; or with the space character in all other cases. When the over-long sequence is replaced, the ending continuation bytes are left in place, which when encountered are then considered unexpected, which may inflate the count of unexpected continuation bytes in Table 1. It is not obvious how to count this kind of double error.

The space character was chosen because this would have minimal effect on the terms that would be considered for inclusion in the final index. The replacement of bytes like this does lead to the issue of terms being potentially split into multiple sections. However, we hypothesise that this is unlikely to result in a measurable retrieval effect.

3. UNWANTED CONTENT

When dealing with content from web collections, it can be beneficial to remove spam before indexing [5]. These spam documents include duplicate documents and low quality content. For instance, Cormack *et al.* [4] found that by removing 70% of the spammiest documents from the ClueWeb09 Category A corpus they were able to improve retrieval precision (estP@10) in 36 of 37 TREC submissions to the 2009 ad-hoc task by 1.4% to 425%. The one submission that did not improve was a Wikipedia only submission for which performance was degraded 4%.

The issue of deciding whether to index a document or not is an issue that needs to be addressed for larger collections. For example, given the list of the 70% of the documents to exclude, two additional string comparisons need to be made when performing a binary search, when compared to the list of the 30% of documents that would remain. Over the ClueWeb09 collection of 500 million documents using the smaller list would save 1 billion string comparisons, and this effect will only grow larger as the collections grow larger. The same saving of two string comparisons per document would save 1.4 billion comparisons on the ClueWeb12 corpus.

When the construction of these lists is done at run-time at the beginning of the indexing process, the overall indexing time increases by about 6%, but the indexing throughput is not affected by this filtering process [5].

The process of filtering at indexing time has a marginal improvement in retrieval effectiveness, increasing ERR-IA@20 from 0.1655 to 0.1931 ($p < 0.05$) on tuned BM25 runs. However, the index with fewer documents is also able to be searched to completion in 75% of the time [5].

If a document passes this spam filtering, but would fail to pass based on the rate, and type, of encoding errors present in the document, then we may wish to reject the document regardless, and vice versa. We leave this for future work.

4. CONCLUSION & DISCUSSION

There are a number of considerations when engineering an indexer and search engine to be able to scale to large collections sourced from the general web. These issues range from high level decisions about determining whether to include a document or not, to lower level issues such as incorrect character encodings.

We have shown that the errors in UTF-8 encoding exist in large numbers across web-pages regardless of how highly their content is rated in terms of spam. In fact, one class of errors is remarkably prevalent among those documents that are rated as being among the least spammiest documents. We showed that this class was the only class of errors that had a statistically significant distribution of documents containing that error when compared to documents not containing that error. We leave the implementation of this document filtering based on the encoding error type and rate, and the combination with the spam filter described in Section 3, to future work

The effect that cleaning the input, and thereby potentially splitting tokens, and the different replacement methods has on the retrieval performance is also left for future work.

5. REFERENCES

[1] ClueWeb09 Wiki. `http://boston.lti.cs.cmu.edu/clueweb09/wiki/tiki-index.php?page=Page+Encodings` accessed 3 Oct 2013.

[2] Modified UTF-8. `http://docs.oracle.com/javase/6/docs/api/java/io/DataInput.html#modified-utf-8` accessed 3 Oct 2013.

[3] Unicode 6.0.0. `http://www.unicode.org/versions/Unicode6.0.0/` accessed 3 Oct 2013.

[4] G. Cormack, M. Smucker, and C. Clarke. Efficient and effective spam filtering and re-ranking for large web datasets. *Information retrieval*, 14(5):441–465, 2011.

[5] M. Crane and A. Trotman. Effects of spam removal on search engine efficiency and effectiveness. In *Proceedings of the Seventeenth Australasian Document Computing Symposium*, ADCS '12, pages 1–8, 2012.

[6] M. Davis. Moving to Unicode 5.1. `http://googleblog.blogspot.com/2008/05/moving-to-unicode-51.html`.

[7] M. Davis and M. Suignard. Unicode technical report 36: Unicode security considerations. Technical report, 2012.

[8] A. Trotman, X. Jia, and M. Crane. Towards an efficient and effective search engine. In *Proceedings of the SIGIR 2012 Workshop on Open Source Information Retrieval*, pages 40–47, 2012.

[9] F. Yergeau. UTF-8, a transformation format of ISO 10646. RFC 3629 (INTERNET STANDARD), Nov. 2003.