**Approved for Public Release; Distribution Unlimited. Case Number 18-3138-9**

“Initial Study of Charset Detection in Apache Tika”

Tim Allison

September 6, 2019

# Introduction

Detecting the character set (charset) of a text-based file is non-trivial. If the detection is incorrect, the extracted text can be completely corrupt (*mojibake*), and any downstream applications (search, entity extraction, machine translation) will fail catastrophically. Apache Tika runs three different detectors on text-based files in a cascade, accepting the first non-null result. This means that Tika ignores potentially conflicting “detections.” We wanted to carry out a study to determine if an ensemble detection scheme may help improve performance.

## Findings in Brief

We identified three areas for improvements in its html-based detectors. We found that on the 1.5 million text-based files in the Tika regression corpus, there was no major improvement to detection performance even with a brute force/“maximum” algorithm (run each detector and select the best). That said, we recognize that for some corpora with more homogenous documents and specific characteristics (e.g. mis-leading html-header declared charsets), there could be a major improvement in performance. This study offers the beginnings for follow on work if the community determines there is a need.

# Research Design

## Metrics

To evaluate the quality of charset detection, we rely on a simple heuristic of the sum of the “common tokens” per detector. For Tika 1.22, the Apache Tika team calculated the top 30,000 tokens in the Leipzig corpus for 121 languages. The tika-eval module runs language detection on the contents and then counts the “common tokens” for that language that were extracted from the text. This is an imperfect metric, but, based on review of a random sample of extracts with different numbers of common tokens for different charsets, it appears to be reliable enough to make actionable decisions.

## Document Types

For charset detection purposes, there are two primary classes of documents: those that have a header that declares the charset (e.g. HTML5: <meta charset="UTF-8"> or the older <meta http-equiv="Content-Type" content="text/html; charset=UTF-8">) and those that do not. In the following, we use the term “charset header” for those documents that have either version of a charset declaration.

## Charset Detection in Apache Tika

There are four charset detectors available in Apache Tika:

1. DefaultHTML detector – this is the legacy detector that is designed to read the charset header
2. StandardHTML detector – this is a recently added (optional) detector that is designed to follow the standards for HTML parsing of the charset headers
3. Universal detector – this is Tika’s wrapper around Mozillas’ chardet
4. ICU detector – this is Tika’s wrapper (with added charsets) of ICU4J’s charset detector

As the default, Tika applies three of these detectors in a specified order (DefaultHTML, Universal, ICU), and it relies on the first non-null result. For example, if the DefaultHTML detector finds a valid charset, Tika will apply that charset and not run any of the other detectors; if DefaultHTML fails to find a charset, Tika will run the Universal detector. The order of the detectors and which detectors are run are all configurable via a tika-config.xml file. Users are also able to add their own detectors.

## Methodology

For this study, we configured Tika to use a single detector on each run against the 1.5 million files. The output of this was 4 directories (one per detector), each of which contained 1.5 million .json extracts, for a total of 6 million .json extracts. We then ran the tika-eval module against each of the four directories and then merged the results into a single Postgresql table.

This allows us to simulate different algorithms via SQL. For example, the cascading methodology used by Tika can be simulated with:

case

when html\_default\_charset is not null

then html\_default\_charset

when universal\_charset is not null

then universal\_charset

when icu\_charset is not null

then icu\_charset

else null

end

We also simulated a “maximum” (brute force) algorithm, which simulated running each individual configuration and then selecting the one with the maximum number of common tokens. For example:

sum(greatest(icu\_num\_common, universal\_num\_common, html\_default\_num\_common, html\_standard\_num\_common))

In the following analysis we rely on the raw number of “common tokens” as a potential sign of improved or degraded performance. For the tika-eval module, we created word lists of the top 30,000 most common words for each of 118 languages in the Leipzig corpus. Across a corpus of natural language documents, we’ve found that this can be a reasonable indicator of improved performance. There are some exceptions, but this is a reasonable approximation/indicator of text fidelity.

## Data

The Apache Tika team focused on 1.5 million text-based files in their large scale regression corpus. The majority of the files are plain text, html or xhtml, as shown in Table 1.

|  |  |
| --- | --- |
| **Mime** | **Number of Files** |
| text/plain | 661,662 |
| text/html | 431,652 |
| application/xhtml+xml | 170,778 |
| text/calendar | 91,488 |
| text/csv | 39,065 |
| text/x-vcard | 21,309 |
| application/x-bibtex-text-file | 20,793 |
| text/tsv | 17,086 |
| text/x-matlab | 10,121 |
| text/x-php | 8,507 |

Table : Top 10 Mime Types

For the sake of simplicity, we focused on primary/container documents only. That is, there are numerous text-based files embedded as attachments in other files. This study is limited to non-embedded text-based files.

# Results

## Detection when there is an HTML Charset Header

For this section, we included documents if either the StandardHTML detector or the DefaultHTML detector was able to detect the charset.

At the highest level, simply the sum of the number of common tokens, the Universal encoding detector has the best performance across all individual encoding methods in this corpus, and, in fact, it is slightly better than Tika’s current cascading method. When we substituted the StandardHTML detector for the DefaultHTML detector, there was a modest increase in performance. If the “maximum” algorithm is applied (run each detector and select the one that yields the most common tokens), there is an increase in common tokens of 1.5% over the default cascading methodology.

|  |  |
| --- | --- |
| **Encoding Detector** | **Number of Common Tokens** |
| HTML Default | 356,336,795 |
| HTML Standard | 349,016,345 |
| Universal | 357,682,248 |
| ICU | 356,796,692 |
| Tika Cascade | 357,522,827 |
| Alternative Cascade (Standard, Universal, ICU) | 358,468,533 |
| Alternative Cascade (Default, ICU, Universal) | 357,506,357 |
| Maximum | 362,925,156 |

Table : Number of Common Tokens by Detector

For the corpus, globally, we do not see significant improvements available with the above algorithms. In the next two sections, we report on two finer grained analyses:

1. Error analysis of the HTMLStandard detector vs. HTMLDefault detector
2. Error analysis of the Universal detector vs. the HTMLDefault detectors

The first analysis reveals two easy area for improvement in the HTMLStandard detector that will make its performance slightly better than the HTMLDefault detector; and the first analysis also reveals that there are some cases where, puzzlingly, the stricter detector is able to extract a charset that HTMLDefault is not able to extract. The second analysis identifies two other areas for improvement and highlights some of the challenges in misleading charset headers.

**Error Analysis on HTMLStandard Detector vs the HTMLDefault Detector**

As a standards based detector, the HTMLStandard detector is stricter than the HTMLDefault detector. The following table presents the most common cases where the HTMLDefault detector was able to extract an encoding, but the HTMLStandard detector was not.

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **HTMLDefault** | **HTMLStandard** | **HTMLDefault Sum Common Tokens** | **HTMLStandard Sum Common Tokens** | **Difference in Sums** |
| ISO-8859-1 | NULL | 6,374,139 | 0 | 6,374,139 |
| x-MacRoman | NULL | 632,939 | 0 | 632,939 |
| windows-1254 | NULL | 385,049 | 0 | 385,049 |
| UTF-8 | NULL | 327,508 | 0 | 327,508 |
| windows-1252 | NULL | 327,418 | 0 | 327,418 |
| x-IBM949 | NULL | 286,356 | 0 | 286,356 |
| x-windows-949 | NULL | 218,745 | 0 | 218,745 |
| Shift\_JIS | NULL | 201,627 | 0 | 201,627 |
| IBM850 | NULL | 193,851 | 0 | 193,851 |
| IBM437 | NULL | 185,564 | 0 | 185,564 |

Table : HTMLDefault vs HTMLStandard when HTMLStandard Fails to Detect

The following table shows the cases where the HTMLStandard detector performed better than the HTMLDefault. We opened a public ticket to drill down on the cases where the more lenient extractor (HTMLDefault) is not able to extract a charset, but the stricter one is (<https://issues.apache.org/jira/browse/TIKA-2936>). We also opened a ticket to handle “UTF-16” as the HTMLStandard detector does (<https://issues.apache.org/jira/browse/TIKA-2937>).

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **HTMLDefault** | **HTMLStandard** | **HTMLDefault Sum Common Tokens** | **HTMLStandard Sum Common Tokens** | **Difference in Sums** |
| UTF-16 | UTF-8 | 6,099 | 748,024 | 741,925 |
| NULL | UTF-16LE | 0 | 359,988 | 359,988 |
| NULL | UTF-8 | 0 | 308,847 | 308,847 |
| NULL | x-MacRoman | 0 | 213,256 | 213,256 |
| NULL | windows-1252 | 0 | 127,931 | 127,931 |
| NULL | windows-1251 | 0 | 112,417 | 112,417 |
| GB2312 | GBK | 6,094,276 | 6,162,373 | 68,097 |
| windows-1251 | UTF-8 | 100,539 | 149,643 | 49,104 |
| UTF-16BE | UTF-8 | 14 | 43,565 | 43,551 |
| NULL | windows-1250 | 0 | 37,659 | 37,659 |

Table : HTMLDefault vs HTMLStandard when HTMLDefault Fails to Detect Correctly

Finally, we observed that the HTMLStandardDetector returned “replacement” for "ISO-2022-KR" and "ISO-2022-CN", where, it appears, the files were actual encoded as ISO-2022-KR and ISO-2022-CN. The Tika team opened <https://issues.apache.org/jira/browse/TIKA-2933> to determine if that decision was standards based.

**Error Analysis on the HTMLDefault Detector vs the UniversalDetector**

As noted above, on the corpus, generally, the Universal detector appears to perform slightly better than the HTMLDefault detector even on documents that have a charset header. For example, when the charset header is incorrect, the Universal detector by itself would dramatically improve performance. See for example, the following table, where we show the charset pairs that would have shown improved performance if the Universal detector had been trusted instead of the HTMLDefault detector.

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **HTMLDefault** | **Universal** | **HTMLDefault Sum Common Tokens** | **Universal Sum Common Tokens** | **Difference in Sums** |
| UTF-8 | EUC-JP | 4,437 | 481,919 | 477,482 |
| EUC-JP | Shift\_JIS | 1,512 | 391,126 | 389,614 |
| UTF-16 | windows-1252 | 1,240 | 368,496 | 367,256 |
| UTF-16 | UTF-8 | 2,563 | 321,717 | 319,154 |
| EUC-JP | UTF-8 | 764,957 | 1,047,029 | 282,072 |
| windows-1255 | UTF-8 | 17,450 | 246,271 | 228,821 |
| windows-1256 | UTF-8 | 52,185 | 249,105 | 196,920 |
| EUC-KR | UTF-8 | 1,081,986 | 1,274,249 | 192,263 |
| UTF-8 | Shift\_JIS | 2,040 | 191,757 | 189,717 |
| windows-1252 | UTF-8 | 427,997 | 554,311 | 126,314 |

Table : Top 10 Increase in Common Tokens when Relying on Universal instead of HTMLDefault

If we reverse the sort order, we see cases where the DefaultHTML detector was correct and the Universal detector likely mis-identified the charset. These would not cause problems for Tika’s cascading algorithm. However, these would pose problems if relying on the Universal detector alone.

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **HTMLDefault** | **Universal** | **HTMLDefault Sum Common Tokens** | **Universal Sum Common Tokens** | **Difference in Sums** |
| windows-1256 | x-MacCyrillic | 836,439 | 122,963 | 713,476 |
| windows-1256 | windows-1252 | 1,416,972 | 747,434 | 669,538 |
| ISO-8859-3 | windows-1252 | 866,028 | 509,403 | 356,625 |
| ISO-8859-4 | windows-1252 | 700,149 | 474,725 | 225,424 |
| UTF-8 | windows-1252 | 23,734,682 | 23,538,742 | 195,940 |
| windows-1254 | windows-1252 | 822,735 | 640,069 | 182,666 |
| ISO-8859-2 | windows-1252 | 1,768,784 | 1,661,027 | 107,757 |
| ISO-8859-9 | windows-1252 | 660,658 | 553,961 | 106,697 |
| windows-1254 | ISO-8859-1 | 1,020,002 | 933,650 | 86,352 |
| GB2312 | windows-1252 | 330,887 | 245,021 | 85,866 |

Table : Top 10 Increase in Common Tokens when Relying on HTMLDefault instead of Universal

The tables above illustrate the point that relying on a single detector or even a cascade of detectors can fail.

## Detection on all Text-based files (without reference to an HTML Charset Header, whether or not it exists)

**Error Analysis on the UniversalDetector vs. the ICUDetector**

In the following table, we show the top 10 differences where ICU appears to be correct. For this run, we included all 1.5 million files, not just those in which there was a charset header. When Universal returns ‘NULL’, the ICU detector would be triggered by the cascading approach in Tika – so, for practical purposes, the reader may ignore those rows. As above, it appears that Universal’s ‘windows-1252’ and ‘x-MacCyrrillic’ is not exceedingly reliable.

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Universal** | **ICU** | **Universal Sum Common Tokens** | **ICU Sum Common Tokens** | **Difference in Sums** |
| NULL | windows-1252 | 0 | 958,757 | 958,757 |
| NULL | ISO-8859-1 | 0 | 575,731 | 575,731 |
| x-MacCyrillic | windows-1256 | 21,368 | 575,343 | 553,975 |
| windows-1252 | windows-1250 | 1,243,288 | 1,444,453 | 201,165 |
| NULL | Shift\_JIS | 0 | 188,239 | 188,239 |
| windows-1252 | ISO-8859-9 | 1,095,140 | 1,231,895 | 136,755 |
| windows-1252 | GB18030 | 125,662 | 244,553 | 118,891 |
| KOI8-R | GB18030 | 9,660 | 119,107 | 109,447 |
| NULL | GB18030 | 0 | 105,877 | 105,877 |
| ISO-8859-1 | ISO-8859-2 | 8,672,941 | 8,762,191 | 89,250 |

Table : Top 10 Increase in Common Tokens when Relying on ICU instead of Universal

# Conclusion and Next Steps

We identified three immediate areas to improve specific detectors (TIKA-2933, TIKA-2936 and TIKA-2937). We found that on the *study corpus* there could be a small increase in accuracy when using the “maximum” algorithm. Given that the overall increase in detection accuracy for the corpus was limited, we would not recommend implementing a detector for the “maximum” algorithm or other ensemble detectors just yet.

However, we recommend that the community evaluate the performance on their data to determine if there is a broader need for such an ensemble detector in the community. If there is such a need, we’d recommend running conventional machine learning – naïve bayes, J48 (decision trees), random forests or similar – with a cross-fold validation framework to build a model for prediction for an ensemble detector based on the three detectors. One could also experiment with the costs of brute force (maximum algorithm): run all the detectors and run the parsers on each unique charset and pick the one with the lowest OOV%.