CommonCrawl3

Refreshing Apache Tika's Large-scale Regression Corpus

Since the last efforts to refresh the regression corpus (see ApacheTikaHtmlEncodingStudy and TIKA-2038), Common Crawl has added important metadata items in the indices, including: *mime-detected*, *languages* and *charset*. I opened TIKA-2750 to track progress on updating our corpus, and I describe the steps taken here.

We are enormously grateful to Sebastian Nagel and Common Crawl for using Tika to detect file types and running it on the entire crawl. The synergy of these two open source|data projects is phenomenal.

As always, we're enormously grateful to Rackspace for hosting our regression testing vm.

There are three primary goals of TIKA-2750: include more recent files, include more "interesting" files, and refetch some of the files that are truncated in Common Crawl. I don't have a definition of interesting, but the goal is to include broad coverage of file formats and languages. See below on Coverage Metrics.

While I recognize that the new metadata is automatically generated and may contain errors, this new metadata allows for more accurate oversampling of file formats/charsets that are of interest.

I started by downloading the 300 index files for September 2018's crawl: CC-MAIN-2018-39 (~226GB).

The top 10 'detected mimes' are:

mime	count
text/html	2,070,375,191
application/xhtml+xml	749,683,874
image/jpeg	6,207,029
application/pdf	4,128,740
application/rss+xml	3,495,173
application/atom+xml	2,868,625
application/xml	1,353,092
image/png	585,019
text/plain	492,429
text/calendar	470,624

Given the work on TIKA-2038 and the focus on country top level domains (TLDs), I also counted the number of *mimes* by TLD and the number of *charsets* by TLD (here).

Finally, I calculated the counts for pairs of 'mime' (as alleged by the http-header) and the 'detected-mime', and that is available here.

Step 1: Select and Retrieve the Files from Common Crawl

My sense from our JIRA and our user list is that people are primarily interested in office-ish files (PDF, MSOffice, RTF, eml, etc) and/or HTML. I therefore chose to break the sampling into three passes:

1. PDFs, MSOffice and other office-ish files 2. Other binaries 3. HTML/Text

I wanted to keep the corpus to below 1 TB and on the order of a few million files.

The sampling frame tables are available here; there's one sampling frame for each of the three file classes.

NOTE: I hesitate even to use the terms "sampling and "sampling frame" because I do not mean to imply that I used much rigor. I manually calculated the sampling frames based on the total counts so that we'd have roughly the desired number of files and file types. As I describe below, there some file types that I thought we should have more of (e.g. 'octet-stream').

The code for everything described here is available on github

Office formats

The top 10 file formats of this category include:

mime	count

application/pdf	4,128,740
application/vnd.openxmlformats-officedocument.wordprocessingml.document	53,579
application/msword	52,087
application/rtf	22,509
application/vnd.ms-excel	22,067
application/vnd.openxmlformats-officedocument.spreadsheetml.sheet	16,290
application/vnd.oasis.opendocument.text	8,314
application/vnd.openxmlformats-officedocument.presentationml.presentation	6,835
application/vnd.ms-powerpoint	5,799
application/vnd.openxmlformats-officedocument.presentationml.slideshow	2,465

```
select mime, sum(count) cnt
from detected_mimes
where
(mime ilike '%pdf%'
 OR
 mime similar to '%(word|doc|power|ppt|excel|xls|application.
*access|outlook|msg|visio|rtf|iwork|pages|numbers|keynot)%'
)
group by mime
order by cnt desc
```

Given how quickly the tail drops off, we could afford to take all of the non-PDFs. For PDFs, we created a sampling frame by TLD.

We used **`org.tallison.cc.index.mappers.DownSample`** to select files for downloading from Common Crawl.

Other Binaries

These are the top 10 other binaries:

mime	cnt
image/jpeg	6,207,029
application/rss+xml	3,495,173
application/atom+xml	2,868,625
application/xml	1,353,092
image/png	585,019
application/octet-stream	330,029
application/json	237,232
application/rdf+xml	229,766
image/gif	166,851
application/gzip	151,940

```
select mime, sum(count) cnt
from detected_mimes
where
(mime not ilike '%pdf%'
 and
 mime not similar to '%(word|doc|power|ppt|excel|xls|application.
*access|outlook|msg|visio|rtf|iwork|pages|numbers|keynot)%'
 and mime not ilike '%html%'
 and mime not ilike '%text%'
)
group by mime
order by cnt desc
```

I created the sampling ratios from for these by preferring non-xml, but likely text-containing file types. Further, I wanted to include a fairly large portion of *oct et-stream* so that we might be able to see how we can improve Tika's file detection.

We used **`org.tallison.cc.index.mappers.DownSample`** to select files for downloading from Common Crawl.

HTML/Text

For the HTML/text files, I wanted to oversample files that were not ASCII/UTF-8 English, and I wanted to oversample files that had no charset detected.

We used **`org.tallison.cc.index.mappers.DownSampleLangCharset`** to select the files for downloading from Common Crawl.

The Output

In addition to storing the files, I generated a table for each pull that included information stored in the WARC file, including information from the http-headers as archived in Common Crawl. The three table files are available here (116MB!).

Step 2: Refetch Likely Truncated Files

Common Crawl truncates files at 1MB. We've found it useful to have truncated files in our corpus, but this disproportionately affects some file formats, such as PDF and MSAccess files, and we wanted to have some recent, largish files in the corpus. We selected those files that were close to 1MB or were marked as truncated:

```
select url,cc_digest from crawled_files
where
(cc_mime_detected ilike '%tika%'
or cc_mime_detected ilike '%power%'
or cc_mime_detected ilike '%access%'
or cc_mime_detected ilike '%rtf%'
or cc_mime_detected ilike '%pdf%'
or cc_mime_detected ilike '%sqlite%'
 or cc_mime_detected ilike '%openxml%'
 or cc_mime_detected ilike '%word%'
 or cc_mime_detected ilike '%rfc822%'
 or cc_mime_detected ilike '%apple%'
 or cc_mime_detected ilike '%excel%'
 or cc_mime_detected ilike '%sheet%'
 or cc_mime_detected ilike '%onenote%'
or cc_mime_detected ilike '%outlook%')
and (actual_length > 990000 or warc_is_truncated='TRUE')
order by random()
```

A rollup of the files that were to be refetched by mime type is here:

mime	count
application/pdf	121,386
application/vnd.openxmlformats-officedocument.presentationml.presentation	3,929
application/x-tika-msoffice	3,830
application/vnd.ms-powerpoint	2,942
application/msword	2,783
application/vnd.openxmlformats-officedocument.wordprocessingml.document	2,722
application/x-tika-ooxml	2,612
application/vnd.openxmlformats-officedocument.presentationml.slideshow	1,663
application/rtf	1,569
application/vnd.ms-excel	1,186

The full table is here.

We used **`org.tallison.cc.WReGetter`**, a wrapper around 'wget' to re-fetch the files from the original URL. If the refetched file was > 50MB, we deleted it; and if the refetch took longer than 2 minutes, we killed the process and deleted whatever bytes had been retrieved.

We refetched these files to a new directory and stored them by their new digest. Each thread in WReGetter wrote to a table to record the mapping of the original digest to the new digest and whether the new file was successfully refetched and/or was too big. Because of limitations of disc space, we stopped the refetch procedure after refetching 98,000 documents, comprising 440GB of data.

We then randomly deleted 80% of the original truncated files and moved the other 20% to /commoncrawl3_truncated.

Finally, we moved the refetched files into the /commoncrawl3_refetched directory.

Step 3 – Areas for Improvements

We carried out this work on one of our TB drives. We have to figure out what to keep from our older commoncrawl2 collection and then merge the two collections. We may consider deleting some of the ISO-8859-1/Windows-1252/UTF-8, English text files. We could also identify truncated files based on parser exceptions and move those into /commoncrawl3_truncated.

Step 4 – Comparison of Contents

Top 20 "container" file mimes:

Mime	Count
application/pdf	528,617
text/plain; charset=ISO-8859-1	184,019
application/msword	78,210
application/vnd.openxmlformats-officedocument.wordprocessingml.document	75,739
text/html; charset=UTF-8	75,156
text/plain; charset=windows-1252	74,144
text/plain; charset=UTF-8	56,462
application/octet-stream	54,278
application/zip	44,989
application/rss+xml	34,213
image/jpeg	30,968
application/atom+xml	28,934
image/png	28,173
text/html; charset=windows-1252	26,232
application/xhtml+xml; charset=UTF-8	25,130
text/html; charset=ISO-8859-1	24,515
application/vnd.google-earth.kml+xml	23,391
application/xhtml+xml; charset=windows-1252	22,304
application/vnd.openxmlformats-officedocument.spreadsheetml.sheet	22,084
application/rtf	21,811

Top 20 Languages (including embedded files) as identified by language id:

Language	Number of Files
en	1,803,350
null	242,442
ru	155,934
de	109,953
fr	96,192
it	73,781
es	59,069
ja	50,941
pl	47,044
pt	35,490
ko	35,251
ca	30,717
fa	26,202

zh-cn	25,379
nl	23,554
ro	23,259
tr	23,111
da	21,967
br	21,420
vi	19,305

Code Coverage Metrics

Tobias Ospelt and Rohan Padhye (the author of https://github.com/rohanpadhye/jqf) both noted on our dev list that we could use coverage analysis to identify a minimal corpus that would cover as much of our code base as possible. Obviously, a minimal corpus designed for our current codebase would not be guaranteed to cover new features, and we'd want to leave plenty of extra files around in the hope that some of them would capture new code paths.

Nevertheless, if we could use jqf or another tool to reduce the corpus, that would help make our runs more efficient.

On TIKA-2750, Tobias reported that his experiment with *afl-cmin.py* showed that it would take roughly four months on our single VM just to create traces (~300 files per hour).

Other Resources

See ComparisonTikaAndPDFToText201811 for notes on a comparison of the output of pdftotext and Tika.