Identifying File Types in the TREC-Polar-DD Dataset Using Byte-Signature Methods

Nithin Krishna, Ajay Kumar, Zack Winoker

March 4, 2016

Abstract

We use byte-frequency based statistical approaches to identify file types in the TREC-Polar-DD Dataset. We study the efficacy of these techniques and combine them with clustering methods to enhance their utility. We also study the effect that file size has on these approaches. The limitations of magic byte classification are explored for the text-plain type and a method for automated new file type detection is discussed.

1 Introduction

File type identification is an essential step in many file forensics and security applications [10]. Traditional methods for identifying file types include reading file extensions (ex: .doc) and magic byte approaches. However, these approaches can be maliciously spoofed or otherwise fail [1]. In response to these shortcomings, more robust statistical approaches have been developed [11].

In this paper, we investigate the utility of byte frequency signature methods developed by McDaniel and Heydari [1] in classifying the types of files in the TREC Polar Data set [2]. This large data set is comprised of information pertaining to climate change in polar regions and is largely unstructured. Previously, Apache Tika has been used to identify a large number of the file types in this set [2]. However, many of them remain classified as octect-stream or text-plain, which are effectively catch-all MIME types. In this study, we implement McDaniel's BFA, BFC, BFCC, and FHT methods [1] and use them to classify types in the TREC set. We compare the efficacy of each method for classifying types and for identifying new file types. Furthermore, we address how byte frequency signatures may vary across different file sizes. We also use k-means clustering to attempt to identify file types within the catch-all MIME types octet-stream and text-plain [12].

In analyzing the results, we discuss shortcomings of the magic byte approach and suggest a more robust solution for new file type identification. Finally, our results are presented using D3 visualizations hosted at [6].

2 Methods

2.1 The TREC Dataset

The TREC-Polar-DD data set is about 65GB and 1,741,530 files in size. In order to speed up the download, the data set was downloaded over 20 threads which took approximately 18 hours for the download to complete. The Amazon S3 bucket containing the data was mounted using RioFS which is a user space file system. The common crawl folder structure was traversed in a Depth First manner to retrieve files which were then inputted to Tika version 1.11 in order to determine the MIME type of the file. The files with same MIME types were grouped into folders and the resulting type distribution is displayed at [8]. In the same link, we also show the MIME diversity before classifying using Tika 1.11 (see figure 1)

Figure 1: Charts of the MIME Diversity with the octet-stream type highlighted. Left, from the previous classification efforts. Right, after running Tika 1.11

Type	Meta	D1-Count	D2-Count	Difference
application-atom+xml		2984	2913	-71
application-dif+xml	New	0	5813	5813
application-dita+xml; format=concept		345	319	-26
application-epub+zip		36	30	-6
application-fits	Retained	24	24	0
application-gzip		2060	1732	-328
application-java-vm	Retained	1	1	0
application-msword	Deprecated	244	2	-242
application-octet-stream		211687	147023	-64664
application-ogg	Deprecated	26	0	-26
application-pdf		44658	44556	-102

Figure 2: A subset of our MIME Diversity table at [8]. Data includes whether more or fewer files have been classified as a given type after using Tika 1.11. Also included is whether a type is new, deprecated, or unchanged.

After the initial classification was made, we removed empty files from the octet-stream folder. We found that 97.5% of these files were empty. Then we computed an average byte frequency signature for each of the types identified by Tika (see figure 3 for an example). For types with greater than 5 files, we used 75% of the available files. Otherwise, all files were used to create the signature.

In computing each signature, we used the companding function $x^{1/b}$ with b=1.5. This was used instead of either A-law or mu-law companding functions since it is much faster to compute and is a relatively accurate approximation of both[3].

During this step, we also investigated the possibility that byte signatures varied over file sizes within any given file type. Traditional approaches to file

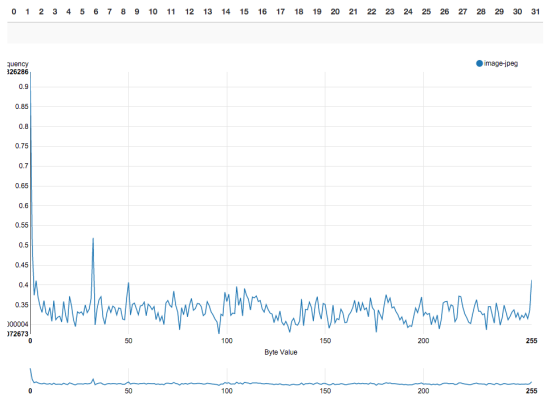

Figure 3: The average byte frequency signature computed for the MIME type image-jpeg.

type detection do not consider file size because they assume that this variation does not exist. Varying file sizes are typically handled by normalizing signatures by the largest byte frequency [1]. We tested the validity of this method as follows. First, we used k-means clustering with a Euclidean distance metric to sort each file type into 5 size-based buckets. Then an average byte frequency signature was generated for each bucket in each type. The size-specific signatures were then compared visually. Results are displayed at [9] (see figure 4 for an example).

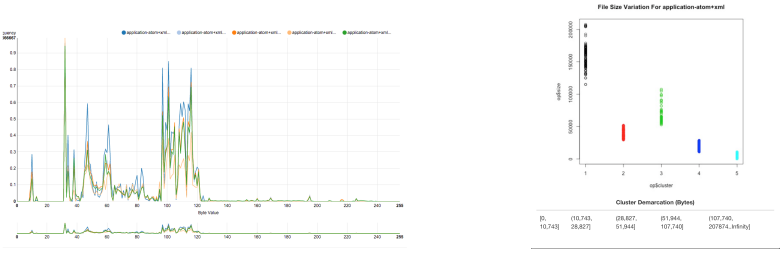

Figure 4: Left: Byte frequency signatures for application-atom+xml files. Each curve is for a different range of file sizes, where the ranges were determined using k-means clustering. Right: Clusters found by k-means for this type.

We computed the BFC for all file types originally identified in the TREC dataset. The average correlation was computed for each type using 75% of the available files for that type. A newly computed signature was generated from the remaining 25%. We also computed BFCC on the BFA signatures computed for every given type. The results are available at [15,16].

We computed FHT matrices with headers and trailers of size 8, 16, and 32 bytes. Our 4 byte results are found by taking the first 4 bytes of the 8 byte results.

We attempted to identify further file types within the catch-all MIME types octet-stream and text-plain using k-means clustering. 5 clusters were generated

and a Euclidean distance metric was used. Average byte frequency fingerprints for each cluster were then generated and compared as in our BFA step. We also re-classified octet-stream and text-plain using version 1.12 of Tika in order to compare our multi-cluster BFA results for these catch-all MIME types.

During our analysis, we also paid special attention to the byte range 97-122, which represents English-language characters. Files that are text-based should have a relatively higher frequency of these bytes in their signatures (see figure 5). We used this observation to help identify text-heavy files in catch-all types.

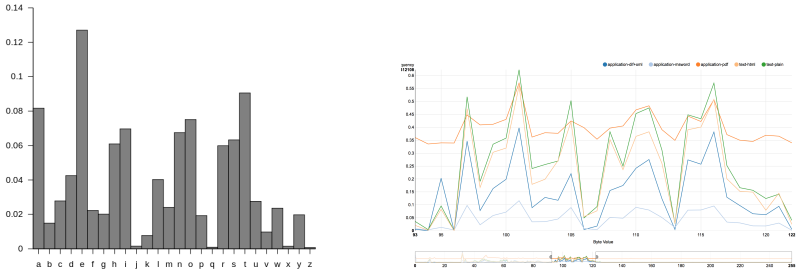

Figure 5: Left: Letter frequency for the English language. Right: Byte frequencies in the range 97-122 for a variety of text-based files. Note the similarity in both plots.

3 Results

We first discovered 5 new types just by running Tika 1.11 on the dataset. These were: text-x-matlab, text-x-csrc, application-zlib, application-x-grib, and application-dif+xml. A large number of files were also reclassified. In particular, 64,664 files were removed from the octet-stream type, 27,540 from text-plain, and 11,720 from application-xml. 10,660 files were added to text-html and 1,362 to text-matlab. The remainder of the results are presented in the D3 visualization at [8]. Tika was able to make these new classifications due to updates in its MIME-type knowledge base. For example, magic bytes were added for text-x-matlab, which allowed Tika to identify matlab files. Changes to magic-priorities of other types resulted in further reclassification.

3.1 File Sizes

For the majority of file types, we found that file size had little to no effect on the character of the corresponding fingerprint (see figure 4). In a small number of cases, there appeared to be a size-specific variation. However, in most of these cases it appears that these variations were due to small sample sizes within the given size range. Therefore, we conclude it is sufficient to normalize byte frequencies to account for file size. Note that this assumption was used without testing in previous studies or was discarded as incorrect [1,12]. Note also that our conclusion agrees with similar conclusions that were made in [13] using significantly less data.

3.2 FHT

For most file types, the FHT sparse matrices were well defined and would be useful for comparisons against files of unknown type. For example, the FHT visualizations for types application-msword and application-zip are quite distinctive (see figure 6 for an example).

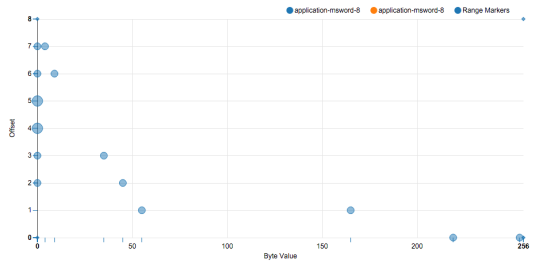

Figure 6: The 8-byte FHT matrix for the file type application-msword. Note that this matrix is sparse, which is necessary if we are to use it for file type and magic byte identification.

Using the results from the 4-byte analysis, we were able to identify magic bytes for 4 file types: CAB, NETCDF, QUICKTIME, and FITS. We added these magic bytes to Tika (see [4]).

However, the sparse matrices for types spanning multiple subtypes showcased virtually no defining characteristics. For example, consider the FHT visualizations for text-plain (figure 7) . There is clearly large variability in the FHT matrix for the files in this type, rendering it useless for figuring out magic bytes.

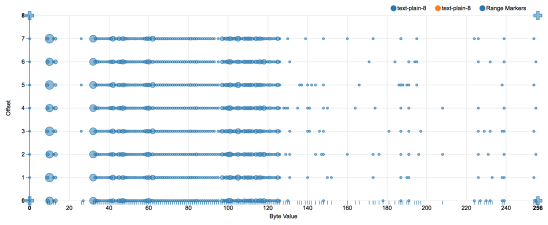

Figure 7: The 8-byte FHT matrix for the file type text-plain. Note that this matrix is *not* sparse, so we can't use it for identifying magic bytes or file types.

3.3 Multiple clusters

Our multi-cluster analysis of octet-stream revealed that we had classified a number of text and ms-word files under this category. This is evident from the peak in byte values ranging from 97 to 122, which are letters in the English alphabet. We see this peak in the signature for clusters c1 and c4, indicating that there are at least two text-based types or collections of types within our octet-stream classification. We compared the octet-stream signature to application-msword's and found further evidence of similarity.

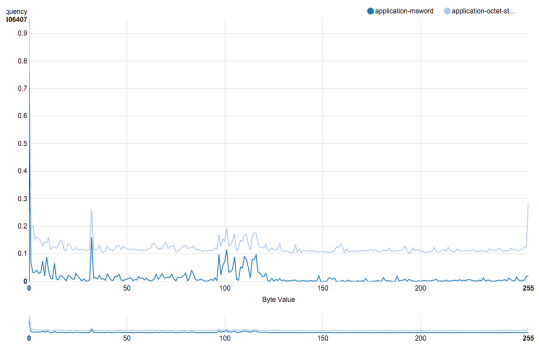

Figure 8: The application-octet-stream and application-msword types have very similar byte frequency signatures.

There also was a significant similarity between cluster c5's signature and application-quicktime's. We therefore predicted that the octet stream category also contained quicktime or some other audio/video files. To verify this, we ran Tika version 1.12 on our octet-stream files. Of the 3792 octet-stream files, 2833 were ms-office files of some type and 6 were audio-vorbis (the breakdown is specified in table 1), indicating that our prediction was correct.

Octet-stream subtype	Number of files
application-msword	714
application-vnd.ms-excel	504
application-vnd.ms-powerpoint	49
application-x-mspublisher	2
application-x-tika-msoffice	1564
audio-vorbis	6
application-x-tika-ooxml-protected	1
application-octet-stream	952
Total MS-Office	2833
Total	3792

Table 1: Tika 1.12 classified our octet-stream files into the above types. Note the large percentage of MS office files and the presence of audio files, both of which match our predictions from clustering and signature methods.

We then re-ran the clustering algorithm on the remaining 952 application-octet-stream files. The results revealed an apparently new cluster with a unique byte frequency signature, with a notable peak at byte number 136 (see the curve "aon5" at [14]). The fact that repeated application of k-means and signature generation revealed this indicates that these methods are very useful for exploratory analysis of file types.

Our multi-cluster analysis of text-plain indiciated that there are a large number of Javascript, CSS, and malformed HTML files. This was confirmed via manual examination of the files. This examination also made it clear that text-plain subtype classification is actually a much richer problem than the corresponding problem in octet-stream. While we can use FHT analysis to

identify magic bytes of octet-stream subtypes, that approach is not possible for text-plain (see figure 7). Note that by nature, Javascript, CSS, and HTML files that do not start with "DOCTYPE..." or "HTML..." do not have any magic bytes or distinguishing features. Tika will therefore never be able to identify these files using its current magic byte based methods. However, our multi-cluster results give us a definitive classification of these types. See figures 9, 10, and 11 for more information. These findings are corroborated by our BFC and BFCC results [15,16].

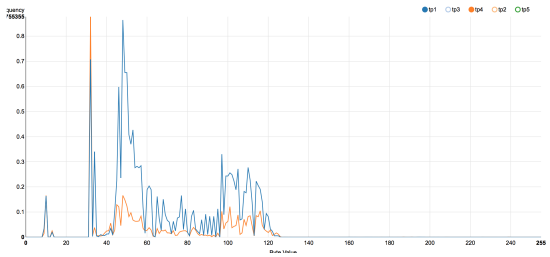

Figure 9: Text-plain clusters tp1 and tp4 have a high occurrence of numbers and characters common to the CSS vocabulary.

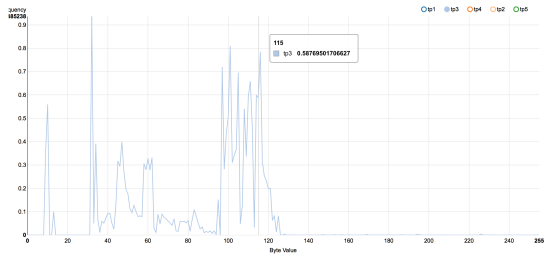

Figure 10: Text-plain cluster tp3 has a high occurrence of the characters '<' and '>' (bytes 60 and 62), which are overwhelmingly common in HTML.

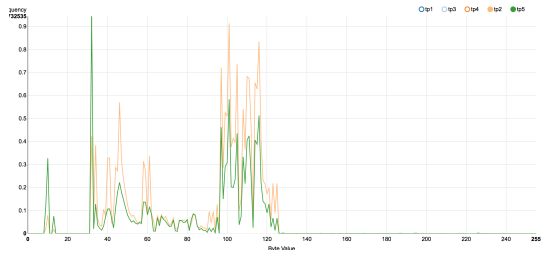

Figure 11: Text-plain clusters tp2 and tp5 have a high occurrence of the characters '(', ')', '{', and '}' (bytes 40,41,123 and 125), which are very common in Javascript when defining functions.

Based on these clustering/signature findings, we suggest that new file types can be systematically identified by the following process. After classifying files

using existing signatures or magic byte methods, use k-means to cluster and generate signatures for catch-all types such as octet-stream and text-plain. Then manually investigate these clusters and assign each of them a type. Alternatively, one could automatically assign each of them a type 'tika-type-n', where n is an index. Reclassify all files using these additional signatures. Repeat until no new significant clusters are found or until distinct magic-byte patterns appear in the catch-all subtypes.

It is also worth noting that our BFA step takes about 2 hours to run on a typical laptop computer for all 1.7 million files. We were therefore able to create signatures for all identified MIME types. We believe that this efficiency is due to our use of the approximate companding function $x^{1/b}$ and a parallelized 2-phase Map-Reduce-like computation. Our extensions to Tika should therefore successfully scale to larger data sets (see [5]).

4 Conclusions and Future Work

We found that FHT analysis is useful for identifying magic bytes of previously-identified file types. We also confirmed that file sizes have little to no effect on their corresponding byte frequency signatures.

Our work also indicated that there are two central issues that can use further work here: the "text-plain" problem and the "octet-stream" problem.

The text-plain problem, namely that magic byte and FHT approaches cannot effectively classify many subtypes of text-plain, is very important for future classification efforts. A significant portion of files on the internet are code and other text-plain subtypes. Hence there is a need for a much more robust mechanism for content-detection in both information-retrieval and security applications. Signature based approaches like ours and that in [10] should be used to supplement existing magic byte approaches in Tika. We believe [17] is a step in the right direction.

The "octet-stream" question asks how we can automate identification of magic byte or other patterns in a class of unclassified types like octet-stream. We suggested a simple recursive solution using a combination of signature generation and k-means clustering. Once this process stops producing new signatures and clusters or once most clusters have distinctive magic bytes, a user can manually examine the results, look for magic bytes and new file types, and then add them to an existing MIME-types database. We believe this solution would be a highly useful addition to Tika.

5 Appendix and Further Notes

1. Our code can be found at [5].
2. Code and a readme for the visualizations can be found at [18].
3. The homepage for the visualization is at [6].
4. We implemented pairwise Jaccard Similarity for Tika Similarity. The pull request can be found at [7].

5. We found that the Python interface made Apache Tika very easy to use. Our code was easy to scale and parallelize.

6. Note that any figures omitted from this report can be found on the visualization's web app [6].

7. We would like to contribute our results to polar.usc.edu.

8. We have also opened the following issues for Tika:

   (a) https://issues.apache.org/jira/browse/TIKA-1891

   (b) https://issues.apache.org/jira/browse/TIKA-1888

   (c) https://issues.apache.org/jira/browse/TIKA-1889

   (d) https://issues.apache.org/jira/browse/TIKA-1891

6 Bibliography

1. Mcdaniel, M., and M.h. Heydari. "Content Based File Type Detection Algorithms." 36th Annual Hawaii International Conference on System Sciences, 2003. Proceedings of the (2003). Web.

2. https://github.com/chrismattmann/trec-dd-polar/

3. McDaniel, Mason. "An Algorithm for Content-Based Automated File Type Recognition." Http://www.forensicswiki.org/. Dec. 2001. Web. ¡http://www.forensicswiki.org/w/images/f/f9/Mcdaniel01.pdf¿.

4. https://github.com/nithinkrishna/tika/commit/86579ec5644ca5fac47d945423b824d8ca7b0c57

5. Readme is at https://github.com/nithinkrishna/file-content-analyzer

6. D3 visualization is at http://104.236.190.155/#/

7. https://github.com/chrismattmann/tika-similarity/pull/56

8. http://104.236.190.155/#/compare/before/now.

9. http://104.236.190.155/#/size

10. "A New Approach to Content-based File Type Detection." Web. http://arxiv.org/pdf/1002.3174.pdf

11. Ahmed, Irfan, Kyung-Suk Lhee, Hyunjung Shin, and Manpyo Hong. "Content-based File-type Identification Using Cosine Similarity and a Divide-and-Conquer Approach." IETE Tech Rev IETE Technical Review 27.6 (2010): 465. Web.

12. Li, Wei-Jen, Ke Wang, S.j. Stolfo, and B. Herzog. "Fileprints: Identifying File Types by N-gram Analysis." Proceedings from the Sixth Annual IEEE Systems, Man and Cybernetics (SMC) Information Assurance Workshop, 2005. Web.

13. Zhang, Like, and Gregory B. White. "An Approach to Detect Executable Content for Anomaly Based Network Intrusion Detection." 2007 IEEE International Parallel and Distributed Processing Symposium (2007). Web.

14. http://104.236.190.155/#/visualize/bfa/aon1:aon2:aon3:aon4:aon5
15. http://104.236.190.155/#/signatures/bfc
16. http://104.236.190.155/#/signatures/bfcc
17. https://issues.apache.org/jira/browse/TIKA-1582
18. https://github.com/nithinkrishna/content-visualization