CSCI-599 TEAM-10
MIME Diversity in the Text Retrieval Conference (TREC) Polar Dynamic Domain Dataset

This report provides a brief summary of the tasks completed for Assignment 1.

TASK 1 & 2 - PRE-REQUISITES:

We downloaded the required software for this project – Apache Tika 1.11, Tika Python and D3js.

TASK 3 – DOWNLOAD TREC DD POLAR DATASET AND REPRESENTATION USING D3JS:

The polar data was downloaded from Amazon S3 bucket and the files were organized under separate folders according to their mime types after running Tika for mime detection. Both Tika Python and Tika-app (Tika jar file) were used for classifying the files. During the classification, the following errors were encountered:

a) *Image/png parse error*: Caused by invalid chunk size. These images were not completely downloaded by the crawler due to the content size restriction.
b) *RSS parse error*: Caused by semantically malformed XML files.
c) *Microsoft office parser error*: Caused by the parser trying to access bytes beyond the file length.

INITIAL OBSERVATION OF THE DATA SET:

- We collected a total of **580,500 files** classified under various top-level media types.
- Among the files that were classified as application/octet-stream, most of them were empty. We found close to 90k files which had file lengths of 0 bytes.

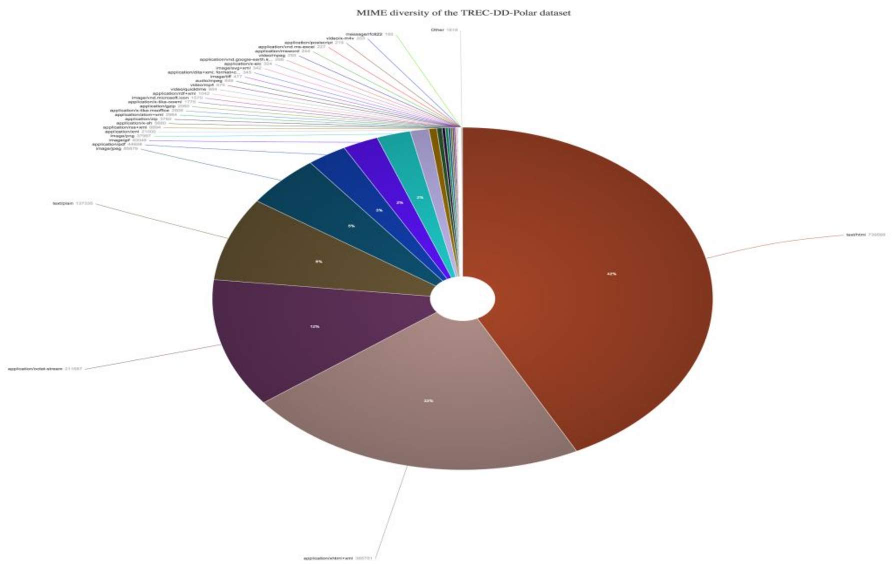
Initially, we selected the following mime types for our analysis –

- text/plain
- text/html
- image/gif
- image/jpeg
- audio/mpeg
- application/pdf
- application/ms-word
- application/vnd-ms-excel
- application/zip
- image/tiff
- application/java-vm
- application/vnd-ms-powerpoint
- video/quicktime
- audio/x-ms-wma
- application/octet-stream

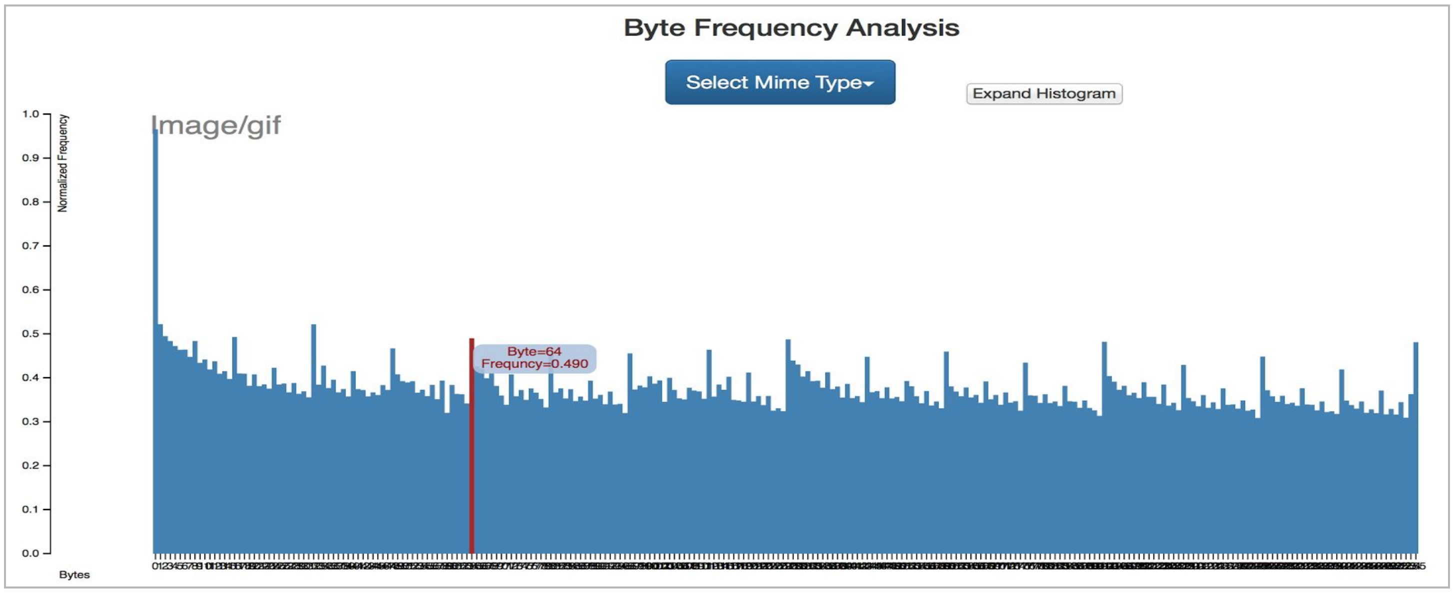
However, towards the middle of the project, we chose the following additional mime types to uncover new magic patterns in the data since for popular types like html, gif etc, no new magic patterns were revealed during analysis.

- application/postscript
- application/atom+xml
- application/rdf+xml
- application/rss+xml

Finally, an interactive D3 based Pie chart was generated from the existing mime diversity of the TREC dataset. The D3 library d3pie.js was used for the visualization.



Task 4 – Byte Frequency Analysis and representation using D3js:

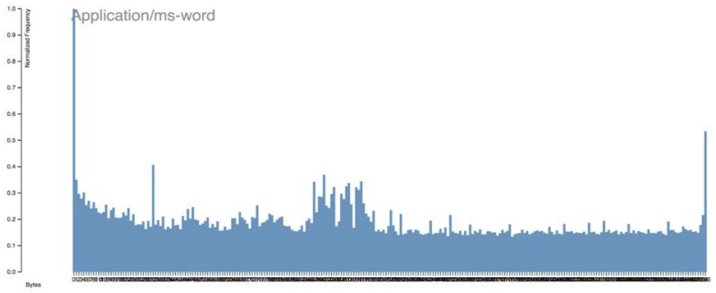


75% of the files in each mime type were used in generating the byte fingerprints.
The byte frequency was normalized by the count of occurrence of the most frequent byte. Normalization was done to prevent one very large file from skewing the file type fingerprint. For most of the file types there were certain byte values which occurred very frequently and were masking the other bytes with lower frequencies. So to overcome this we used the companding function (Mu-law), Where each byte strength is calculated using the formulae: $y=x^{(1/\beta)}$ with $\beta = 2$.
Below are some interesting observations that were made:

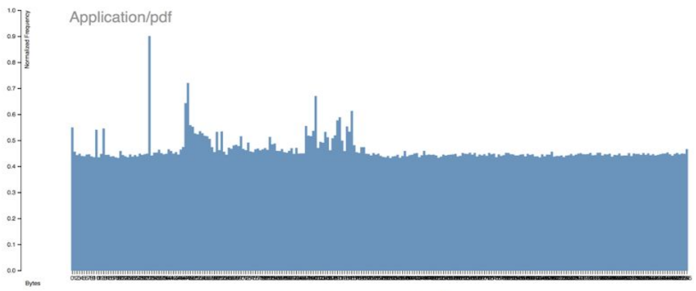
- Although most of the mime fingerprints were unique, there was a high degree of similarity between fingerprints of text/plain and text/html. Byte 32 (ASCII: space) is the most frequently occurring byte in these 2 file types.
- We also observed that for binary files there was nothing significantly eye catching. Byte patterns were plain and flat.
- For application/html files byte 60(ASCII : <) and 62(ASCII : >) had a high frequency strength.

- For Application/rss+xml byte 114(ASCII:r) and 115(ASCII:s) had a high frequency strength so we considered this to be a potential magic byte. Similarly for application/atom+xml we found bytes 100(ASCII:d), 101(ASCII:e) and 102(ASCII:f) having a high frequency strength.

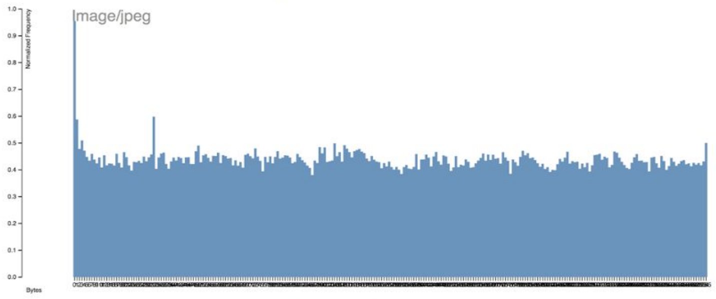
Below are the Byte Signatures for few of the commonly occurring mime types :



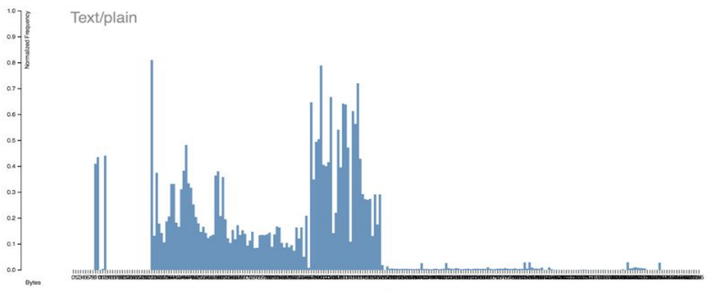
Application/ms-word



Application/pdf



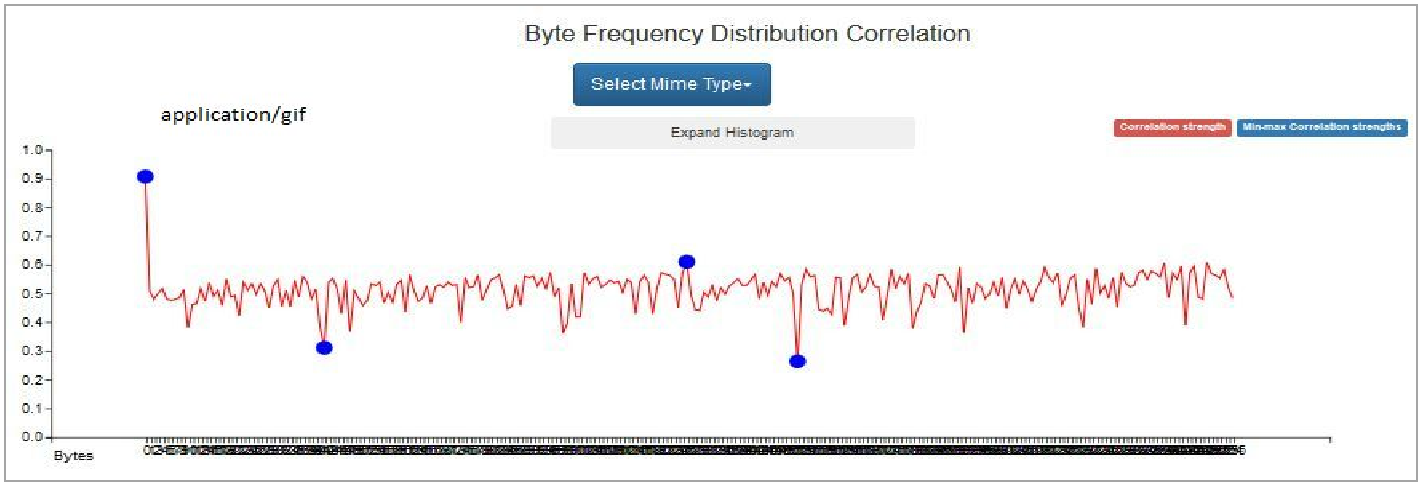
Image/jpeg



Text/plain

Task 5 – Byte Frequency Correlation and representation using D3js:

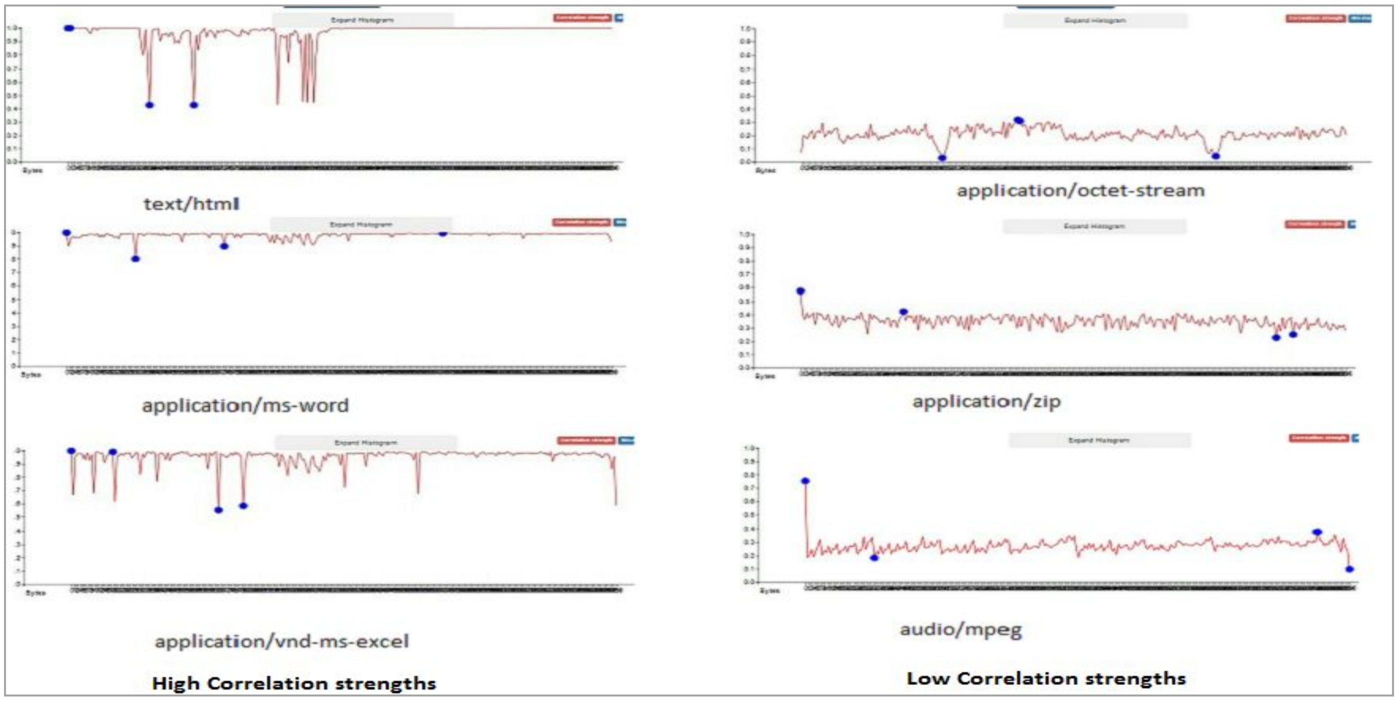
Byte Frequency Distribution - Correlation:



The BFD correlation was done against 25% of the remaining files for each mime type. The "correlation strength" between the byte frequencies of the same byte values is calculated for these files against the fingerprint generated from 75% of the files. The correlation strength was calculated by the given formula:

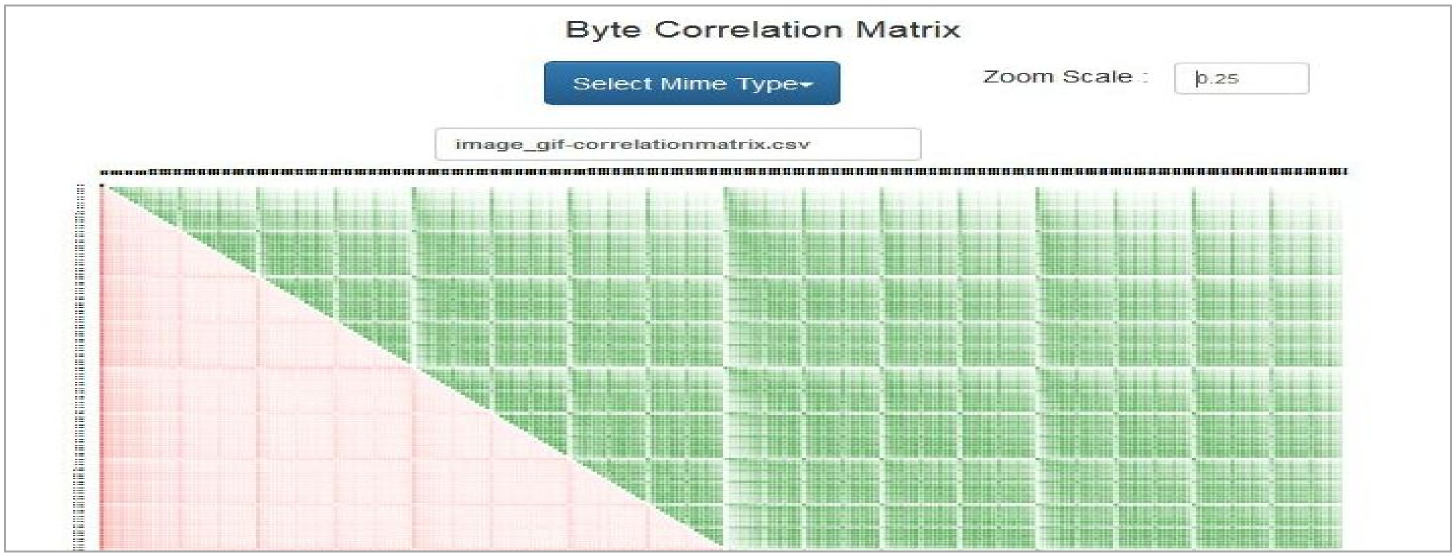
$F(x) = e^{(-x2/2\sigma2)}$ where σ=0.125. This function was used since it is known to provide high tolerance for small variations and less tolerance for large variations and gives a bell curve like graph.

Below are some mime types where very high and low correlations were observed:



As expected, the application-octet stream had very low correlation strengths since the underlying data was a mix of different mime types.

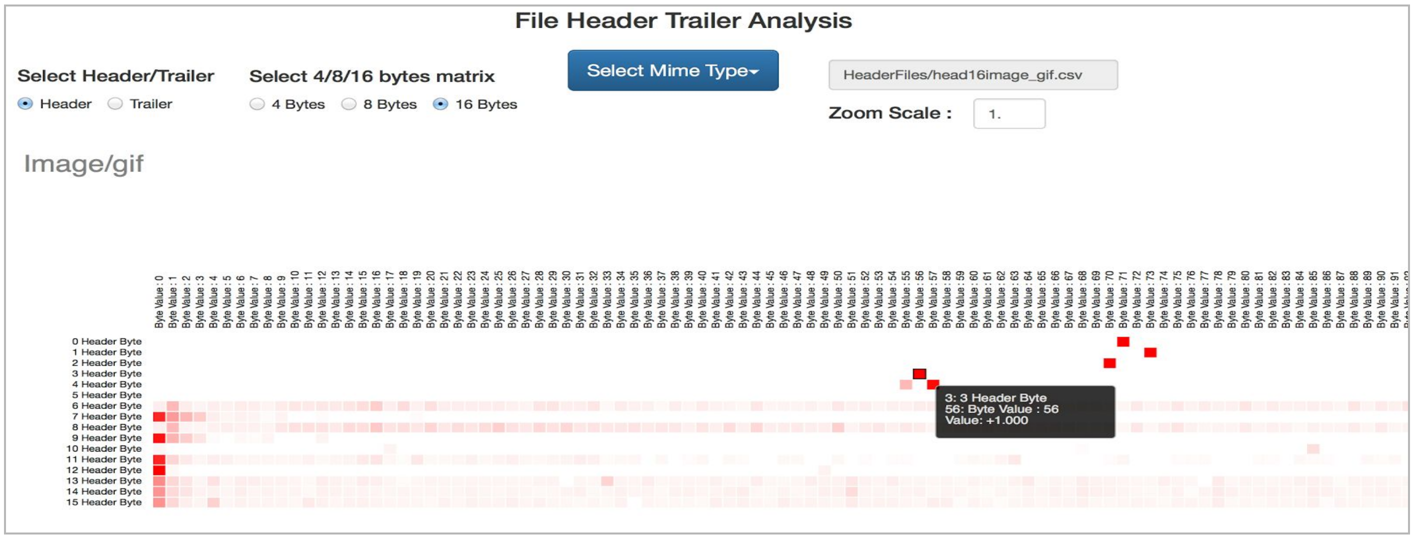
Byte Frequency Cross - Correlation:



The byte cross-correlation was done for 25% of the remaining files for each mime type. A cross-correlation matrix was calculated where the lower half of the matrix holds the average frequency difference between pairs of bytes in these files and the upper half stores the correlation strengths between the previous byte frequency difference and the byte frequency difference of each incoming file.

The correlation strengths are calculated using the same formula mentioned in the previous section and the same normalization and companding functions are applied while calculating frequencies. As before, all values are averaged by dividing them by the total number of incoming files.

TASK 6 – FILE HEADER TRAILER ANALYSIS AND REPRESENTATION USING D3JS:



File Header Trailer Analysis

Select Header/Trailer: [● Header] ○ Trailer

Select 4/8/16 bytes matrix: ○ 4 Bytes ○ 8 Bytes [● 16 Bytes]

Select Mime Type

HeaderFiles/head16image_gif.csv

Zoom Scale : 1.

Image/gif

3: 3 Header Byte
56: Byte Value : 56
Value: +1.000

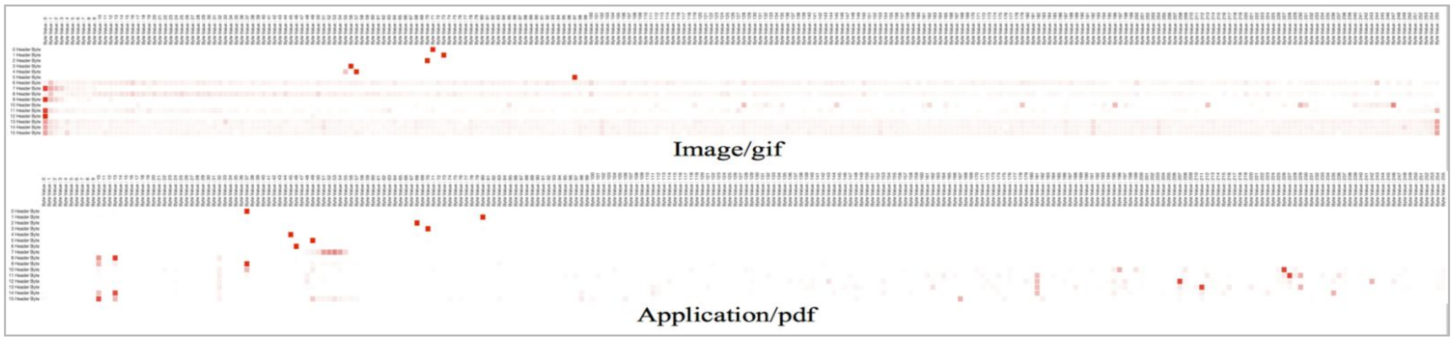
Byte Value : 0 … Byte Value : 91
0 Header Byte … 15 Header Byte

The header and trailer analysis was very effective in identifying new magic bytes. From the above scatter matrix we could clearly identify the bytes that were frequently occurring as part of the file header and trailer. The above d3 allows you to interactively select different header and trailer formats. It also allows you to zoom through for inspection.

While cross verifying the header bytes from the above scatter matrix with the byte order marks present in tika-mimetypes.xml file, we realized that most of the common file types like image/gif, application/pdf etc had their magic bytes already present. So we decided to analyze the header/trailer patterns for the rarely occurring mime types. Below were some of the new magic bytes that were added as part of this process.

a) Application/java-vm :
   i) <match value=”0x03002D” type=”string” offset=”5” />
   ii) <match value=”0x015F” type=”string” offset=”8” />
b) Application/msword :
   i) <match value="0x0D444F43" type="string" offset="0"/>
   ii) <match value="0xDBA52D00" type="string" offset="0"/>
c) Application/postscript :
   i) <match value="0x252150532D41646F" type="string" offset="0"/> etc.

Below are the fingerprints generated for few common mime types:



Task 7 – Update Tika's Mime Repository:

We were able to find new magic byte patterns in **17 mime-types and 2 new mime-types** which were not present in TIKA's mime repository. The details of these mime types are captured in the table below:

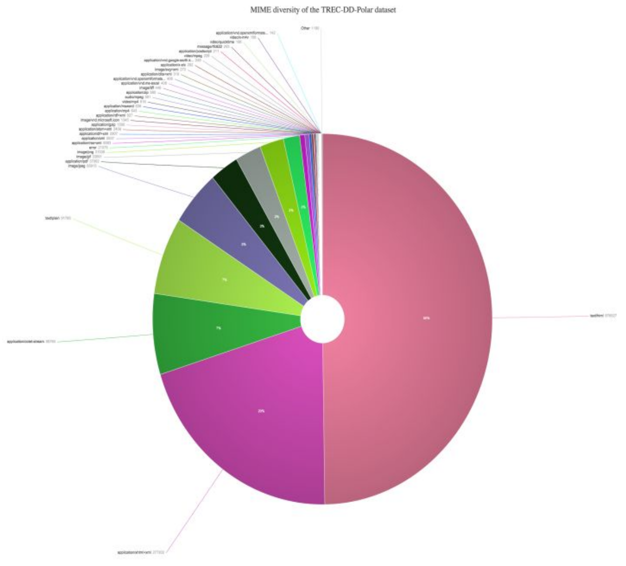
#	New/ Existing	Mimetype	New Mime magic	Priority	Offset	Pattern/Bytes	New Files
1	Existing	Application/postscript	<match value="0x252150532D41646F" type="string" offset="0"/>	50	0	%!PS-Adobe-3 .0 EPSF-3.0	2
2	Existing	Application/rdf+xml	<match value="0x726466" type="string" offset="5:400"/>	50	5 - 400	rdf	4
3	Existing	Application/atom+xml	<match value="0x66656564" type="string" offset="5:50"/>	50	5:50	feed	2
4	Existing	Application/rss+xml	<match value="0x727373" type="string" offset="5:50"/>	50	5:50	rss	3
5	Existing	Image/tiff	<match value="MM\x00\x2b" type="string" offset="0"/>	50	0	MM.+	0
6	Existing	Application/java-vm	<match value="0x015F" type="string" offset="8" /> <match value="0x03002D" type="string" offset="5" />	50	8 , 5	._ , -	0
7	Existing	Application/zip	<match value="0x504B4C495445" type="string" offset="30"/> <match value="\x50\x4B\x53\x70\x58" type="string" offset="526"/>	50	30, 526	PKLITE , PKSFX	0
8	Existing	Application/msword	<match value="0x0D444F43" type="string" offset="0"/> <match value="0xDBA52D00" type="string" offset="0"/> <match value="0xECA5C100" type="string" offset="512"/>	50	0,0,512	.DOC , Û¥-. , ì¥Á.	2
9	Existing	Application/vnd.ms-excel	<match value="0x0908100000060500" type="string" offset="512"/> <match value="0xFDFFFFFF10" type="string" offset="512"/>	50	512,512	.... , ýÿÿÿ..	5
10	Existing	Application/vnd.ms-powerpoint	<match value="0x0F00E803" type="string" offset="512"> <match value="0xA0461DF0" type="string" offset="512">	50	512,512	..è. , F.ð	0
11	Existing	Audio/x-ms-wma	<match value="0x3026B2758E66CF11" type="unicodeLE" offset="0" />	50	0	!-- 0&²u.fÏ.¦Ù.ª.bÎ l	1

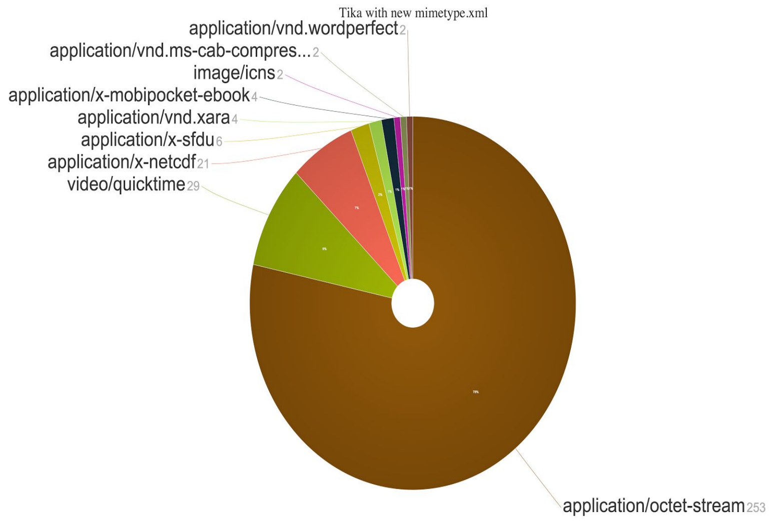
12	Existing	Video/quicktime	<match value="0x66726565" type="string" offset="4"/> <match value="0x77696465" type="string" offset="4"/> <match value="0x706E6F74" type="string" offset="4"/>	50	4,4,4	wide, free	29
13	Existing	Application/vnd.wordperfect	<match value="0xff575043" type="string" offset="0"/>	50	0	ÿWPC	2
14	Existing	Application/x-mobipocket-ebook	<match value=" 0x0516074d4f4249424f4f4b" type="string" offset="60"/>	50	60	BOOKMOBI	3
15	Existing	Application/vnd.xara	<match value="xar!" type="string" offset="0">	50	0	xar!	8
16	Existing	Application/vnd.ms-cab-compressed	<match value="MSCF" type="string" offset="0">	50	0	MCSF	2
17	Existing	Application/x-netcdf	<match value="CDF" type="string" offset="0" />	50	0	CDF	22
18	New	Image/icns	<match value="icns" type="string" offset="0">	50	0	icns	3
19	New	application/x-sfdu		50	0:512	Standard Formatted Data Units	10

We also updated the tika-mimetypes.xml for these new patterns and re-ran the tool on the application/octet stream to get the following results.

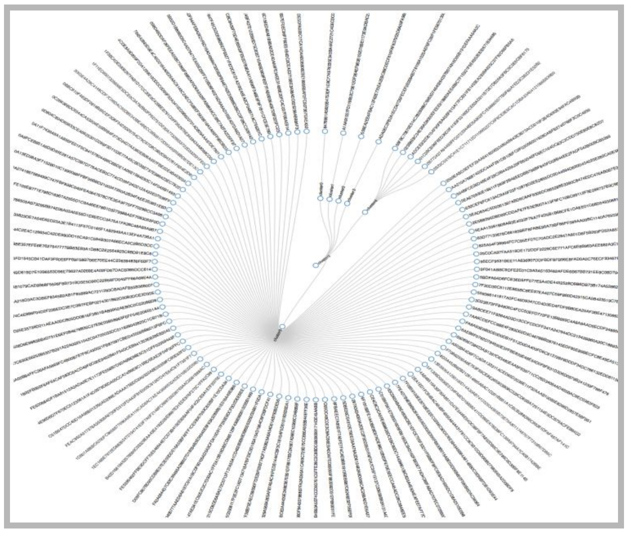
The below pie charts represent the mime data diversity before and after TIKA was updated. Out of the data we downloaded from the Amazon S3 instance, we found 95,780 files belonging to the application/octet-stream (Graph 1). On updating TIKA for the new magic patterns, we regenerated the graph as can be seen below (Graph 2).

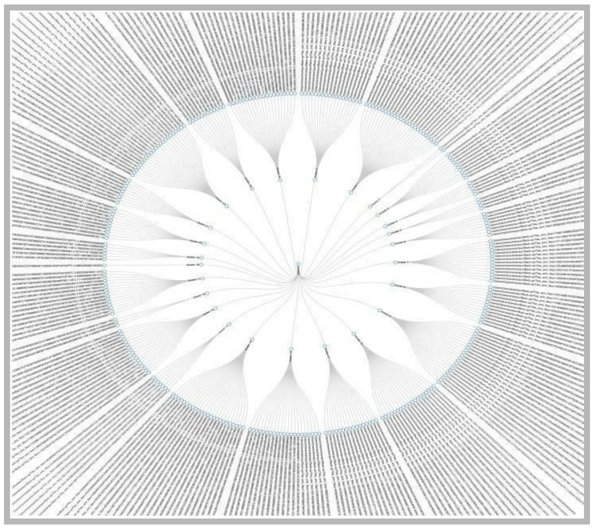
22% of the application/octet-stream data was re-classified after we added the new magic byte patterns in TIKA.

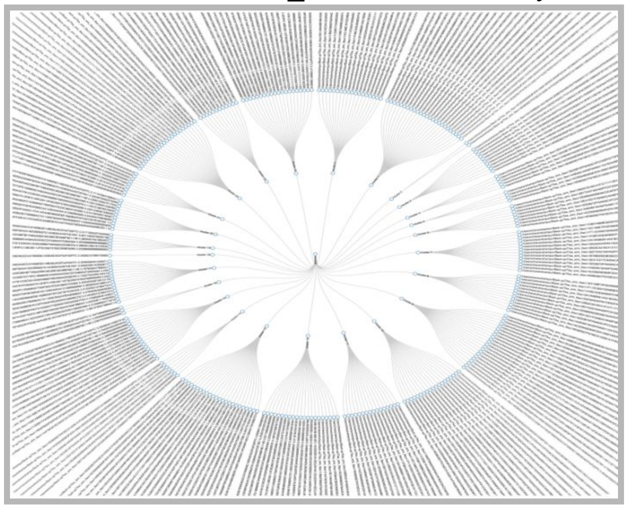




Task 8 – Tika Similarity on TREC Data MIME Types:


Jaccard_Distance Similarity


Cosine_Distance Similarity

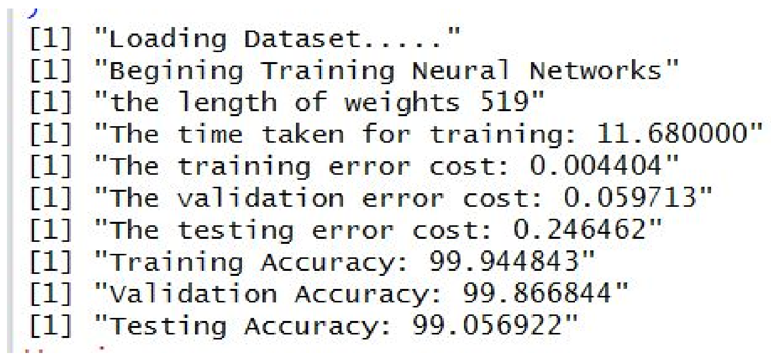

Edit_Distance Similarity

The above graphs are the generated, cluster-based visualization of the three distance measure metrics.
The method how documents are grouped into clusters using Jaccard distance is different from the method used for Edit and Cosine Distance.
For Jaccard Distance we create a Golden Meta dataset by taking the union of all the metadata keys of the input files present in the directory and then Compare each file metadata keys with this Golden set and calculate the Jaccard Distance which is nothing but the resemblance score which is used to cluster the documents.
For Cosine and Edit Distance we compare the metadata key and values of each file with every other file in the directory and generate a cluster as shown above.
Jaccard Distance was the most effective out of the lot, since it allowed for visualizing the metadata/content-type that is associated with each file. Around 80% of the files were correctly classified into clusters based on their content-type, However there were few deviations for e.g. a file belonging to type: image/vnd.microsoft.icon was classified under the same cluster as that of video/quicktime. This misclassification is because Tika extracts the same set of metadata features for both the files and when these are compared with the golden standard they evaluate to a common resemblance score.

Cosine and Edit Distance were not very useful in classifying the files into their mime-types but they provided an accurate measure of similarity between any two files, Using this data we could chain across the results and get a and again run Tika on all these files and identify their file types.

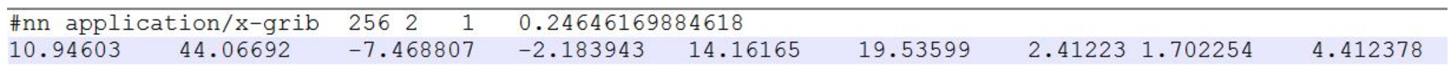
Task 9 – Content Based Mime Detection on TREC Data:

We selected Image/jpeg for applying the content based MIME detection in Tika to our data set. We created a trained model for the identified Mime-type by using around 5000 files each for training, validation and testing including negative training examples to train the model. A byte histogram was created in the form of .csv and was fed into the R Model taken from GitHub project source repository of filetypeDetection (LukeLuish). The following is the output of training , validating and testing our neural network model:



```
[1] "Loading Dataset....."
[1] "Begining Training Neural Networks"
[1] "the length of weights 519"
[1] "The time taken for training: 11.680000"
[1] "The training error cost: 0.004404"
[1] "The validation error cost: 0.059713"
[1] "The testing error cost: 0.246462"
[1] "Training Accuracy: 99.944843"
[1] "Validation Accuracy: 99.866844"
[1] "Testing Accuracy: 99.056922"
```

In the end model parameters and the configurations are written in a text file called 'tika-example.nnmodel'.



```
#nn application/x-grib   256 2   1   0.24646169884618
10.94603    44.06692    -7.468807    -2.183943    14.16165    19.53599    2.41223 1.702254    4.412378
```

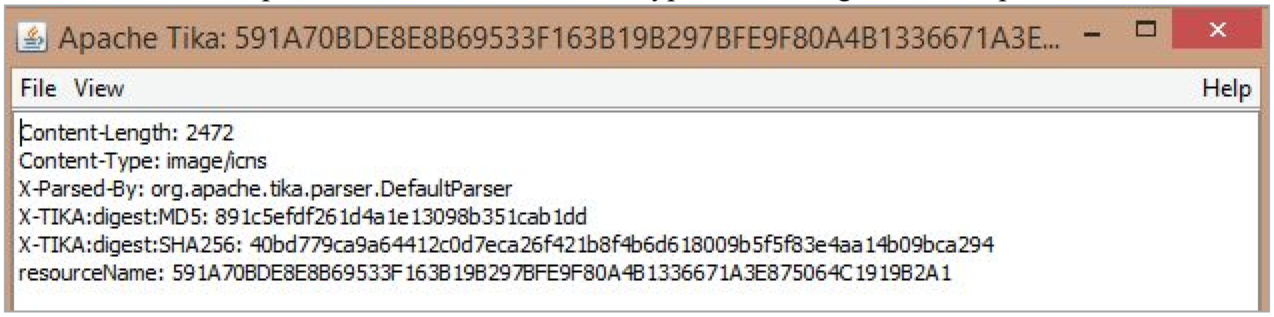
Here , 256 is the number of input (bytes), 2 is the number of hidden input, 1 output units and test set error cost being 8.526 e $^{-13.}$The model parameters are represented in the 2nd line delimited by tabs.
We then imported this file to Tika. ExampleNNModelDetector (subclass of TrainedModelDetector) recreates the trained model , to predict unseen files better.
To test the model we used the unit test "MimeDetectionWithNNTest" in: tika-1.11-src\tika-1.11\tika-core\src\test\java\org\apache\tika
The assertEquals function in the unit test fails as all the files gets classified as octet-stream data. The detect function in TrainedModelDetector.java always returns the Maxtype as octet-stream as the total probability of the tested files never crosses 0.5f. We suspect that the number of files taken for training, testing and validation is really less in number, to do any significant analysis, which we would expect in any neural network with training model formed from anything less than huge chunk of data.

HIGHLIGHTS:

1. We were able to find new magic byte patterns in **17 mime-types**.
   Note: We created issues in Apache TIKA's JIRA issue tracker along with relevant pull requests to the github repository. These issues are currently being followed-up (TIKA-1883, TIKA-1882).

2. We found two new mime types in the data: **image/icns** and **SFDU** (Standard Formatted Data Unit)
   ICNS: For the ICNS mime type, we updated the tika-mimetypes.xml file with the appropriate file extension and magic byte pattern information and also updated the "**tika-parser**" project to add a new

custom parser – ICNSParser.java and relevant Test classes to parse this new mime type. As a result of this, TIKA can now parse files for this new mime type. Also being followed up as TIKA-1893.


Apache Tika: 591A70BDE8E8B69533F163B19B297BFE9F80A4B1336671A3E...
File View
Help

```
Content-Length: 2472
Content-Type: image/icns
X-Parsed-By: org.apache.tika.parser.DefaultParser
X-TIKA:digest:MD5: 891c5efdf261d4a1e13098b351cab1dd
X-TIKA:digest:SHA256: 40bd779ca9a64412c0d7eca26f421b8f4b6d618009b5f5f83e4aa14b09bca294
resourceName: 591A70BDE8E8B69533F163B19B297BFE9F80A4B1336671A3E875064C1919B2A1
```

SFDU: Since these file types do not have a pre-existing mime type; we created an extension mime type "application/x-sfdu" as follows. This adds additional capability in TIKA to classify files with the new mime type.

```
   <mime-type type="application/x-sfdu">
    <_comment>Standard Formatted Data Units</_comment>
        <magic priority="50">
          <match value="Standard Formatted Data Units" type="string" offset="0:512">
          </match>
        </magic>
      <glob pattern="*.sfdu"/>
 </mime-type>
```

CONCLUSION:

1. Were you able to discern any new MIME types within the 200,000 application/octet-stream ("unknown") types?
   Yes, we were able to detect two new mime types. The relevant information can be found in the "Highlights" section, point 2.

2. How well did BFA work, compared to FHT?
   BFA helped us in analyzing byte patterns for mime types like Application/rss+xml, Application/atom+xml etc where we could clearly figure out from the signatures the frequently occurring bytes and then find out where exactly they were occurring in files. On the other hand Most of the new byte patterns were identified using FHT Analysis as mostly the unique identifiers for a file were present in header section of the file.

3. Did the D3 interactive visualizations help you understand the byte frequencies, and to identify patterns?
   Yes, D3 was very helpful in visualizing byte frequency patterns for a given mime type. It also helped bringing out the differences between different mime types in a very intuitive manner.

4. Describe your results from BFA, BFC, and FHT. What MIME types did you pick, and why?
   We initially picked up 14 mime types based on two things: a) popularity of the mime types b) The mime types which had a sizable amount of files for us to analyze.
   However, when we started analyzing some of the more popular formats like plain text, HTML etc, we found that almost all the mime magic had already been captured in TIKA. So, we included few additional mime types in our analysis (See section 1 notes).

Out of the three methods, FHT helped us identify most of the new mime magic we found. However, there were mime types like - application/x-mobipocket-ebook and others which FHT could not have revealed since very strong patterns were found in places other than the header. The BFA analysis and BFC cross-correlation matrix helped us identify such patterns.

5. Why Tika's detector was unable to discern the MIME types?

    a. Was it lack of byte patterns and specificity in the fingerprint?
        - A lot of mime-types did not have any magic patterns configured in tika-mimetypes.xml. The detection for these types was being done based on file extensions only. Hence we were able to add magic byte patterns for such files.
        - Also, for a few mime types, though magic bytes were configured, new magic patterns appeared due to our byte analysis, which were then added to the repository.

    b. An error in MIME priority precedence?
        - Did not come across any incorrect mime type classification due to errors in priority precedence.

    c. Most of the files that Tika classified as application/octet-stream were empty and since there is no data present in the file to figure out the mime type, Tika used the default fall back parser and classified them as Octet-Stream.

    d. Lack of sensitivity in the ability to specify competing MIME magic priorities and bytes/offsets?
        - A little more flexibility in configuring magic byte patterns in TIKA, for example,
            a) Being able to configure the frequency of occurrence of a byte pattern instead of just the offset.
            b) Being able to configure magic patterns without the use of offset. For example, a range of 0 to file size

    As a result of these limitations, mime magic for popular formats like JavaScript and CSS is very difficult to add and these files end up being classified as plain text.

6. Feedback about Tika usage
Easy to install and use. Code is well structured and it was easy to find relevant pieces of code wherever required. However proper code commenting was missing in multiple places.

YES. We would like to contribute our updated MIME classifications code and visualizations to http://polar.usc.edu/ and also to refine the TREC dataset, and also be disseminatd to DARPA and NSF.