**MIME Diversity in the Text Retrieval Conference (TREC)**

**Polar Dynamic Domain Dataset**

1. **Download and install Apache Tika:**

Cloned the latest trunk and build tika from it.

1. **Download and install D3.js:**

Used d3 as a script in all the html pages and over cross-origin issue. We also tried a few examples to get a hang of the library.

1. **Download the Amazon S3-based TREC-DD-Polar data:**

We downloaded all diversity Json files from “latest-commoncrawl” bucket’s main directory. Based on number of different file types and size, we chose to download “572-team1” folder to try out before we go for full download. Challenge here was that we did not know which folder contains which file types as folder hierarchy was based on crawling and had no ordering with respect to MIME types.

We downloaded all .gif files from this folder. Wrote a pre-processing java code to evaluate the body of each response and save in a new file as this would be required by all the algorithms, but due to excess computing time we continued with the simpler polar-fulldump data. We equally divided 180 folders amongst all group members. Wrote a simple java code that would take a file and detect it using tika trunk and save move the file to its mime-type bucket. This code is supplied as the preprocessing.java file. We dumped the filename to a separate zerobucket folder so that these files can be observed from commoncrawl and detect issues that caused 0 bytes to be downloaded instead of the file.

b. Once, we had all the data sorted, we found that mime diversity reported on<http://github.com/chrismattmann/trec-dd-polar/> was not entirely matching with mime diversity that we observed. E.g. before looking at the data, we had decided to include “xscapplication/zip” mime type for our analysis. However, we did not get even a single file detected as that type from entire “polar-fulldump” even though there were supposed to be 85 files of that type. Mime-types selected over a range of different file formats.

|  |  |  |
| --- | --- | --- |
| Manali (~ 10 GB) | Kinara (~ 11 GB) | Nilay (~ 10 GB) |
| application\_xhtml+xml (9.5 GB)  image\_gif (451.2 MB)  text\_x-matlab (12.4 MB)  application\_diff+xml (8.8 MB)  message\_rfc822 (2.3 MB) | application\_pdf (8.9 GB)  image\_png (1.8 GB)  audio\_mpeg (431.8 MB)  audio\_x-ms-wma (7.1 MB) | video\_mp4 (4.2 GB)  image\_jpeg (4.0 GB)  video\_quicktime (2.1 GB)  application\_zip (433.6 MB)  audio\_x-wav (3.7 MB) |

application\_octet-stream left for all to observe together

b.We used an interactive pie chart since it clubs families together creating less cluster and greater visibility. We have included this chart in our submission. Figure 1 is created with mouse hover over octet stream data which one can see on the top left along with its relative percentage and number of files. On the left we are showing legend for top level mime types.

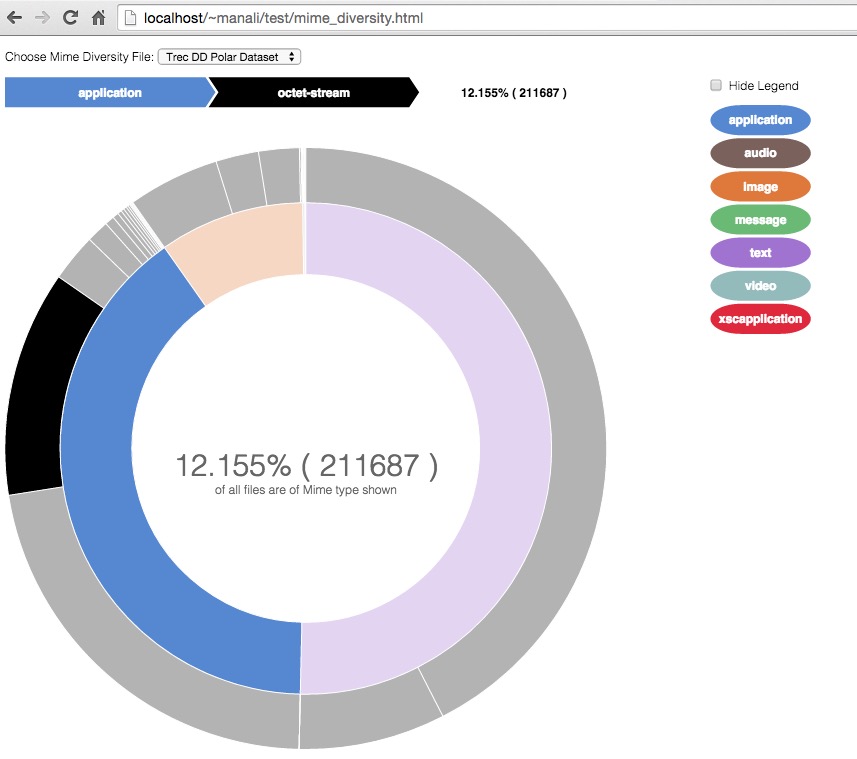


Figure 1: Trec DD Mime Diversity

4-a,5-a,6-a we created a maven project that runs a pipeline over all the data format.

Maindirectory contains all mimetypes folders.

Each mimetype folder contains 2 folders holding 75% and 25% files

mainDirectory/

├── image\_gif

│   ├── image\_gif

│   └── image\_gif\_25

├── message\_rcf822

└── text\_x-matlab

├── text\_x-matlab

└── text\_x-matlab\_25

use the jar file to run for each individual mimetype and supply it the complete path to the maindirectory as well as the name of the mimetype

java –jar mime-analyser-0.1-snapshot-jar-with-dependencies.jar “fullpath/to/mainDirectory” “image\_gif”

This pipeline is useful since it saves time and objects created between BFA and BFC can be used before they are dumped as json fingerprints.

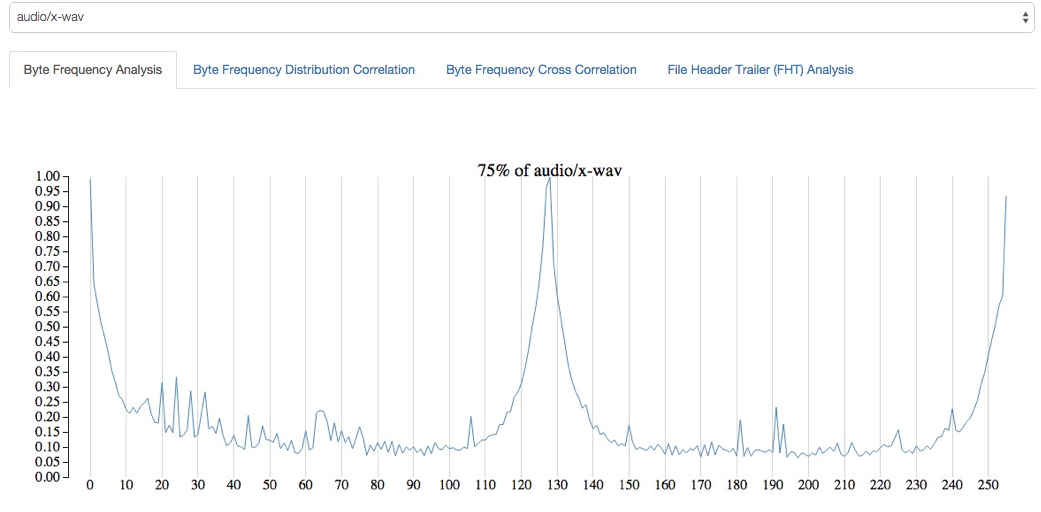
The jar first runs a BFA on 75% data and computes the fingerprint. It then uses remaining 25% data to compute fingerprint on 100% data for bfcc. It also computes the bfa strength for each file in test set and dumps into a json.

BFC is then performed for each file in 25% data. Each file in test set is mapped against BFA100 fingerprint to calculate correlation between two byte values for a given file type.

Finally FHT fingerprint is computed and scores for 4,8,16 bytes are dumped for each file in the test set.

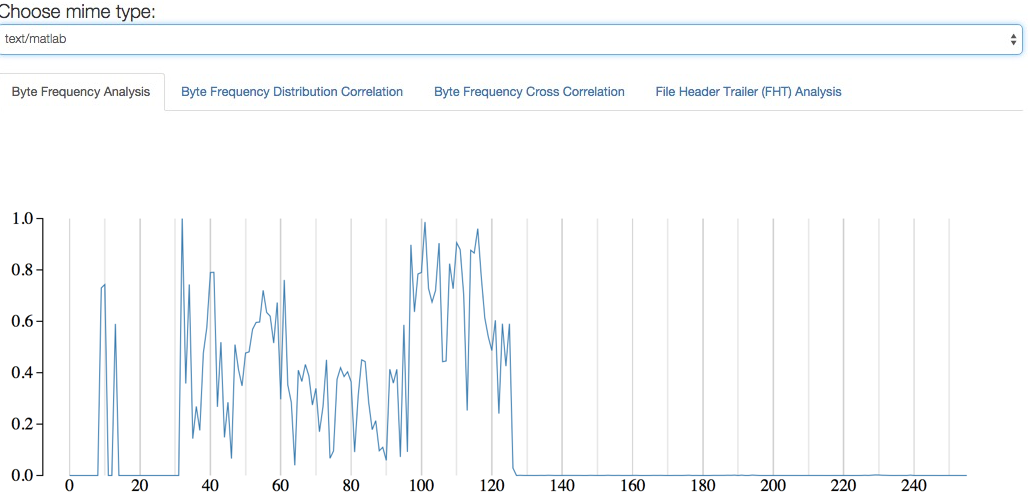
At each stage the necessary fingerprints and scores and file details are dumped into a json file for use in d3.

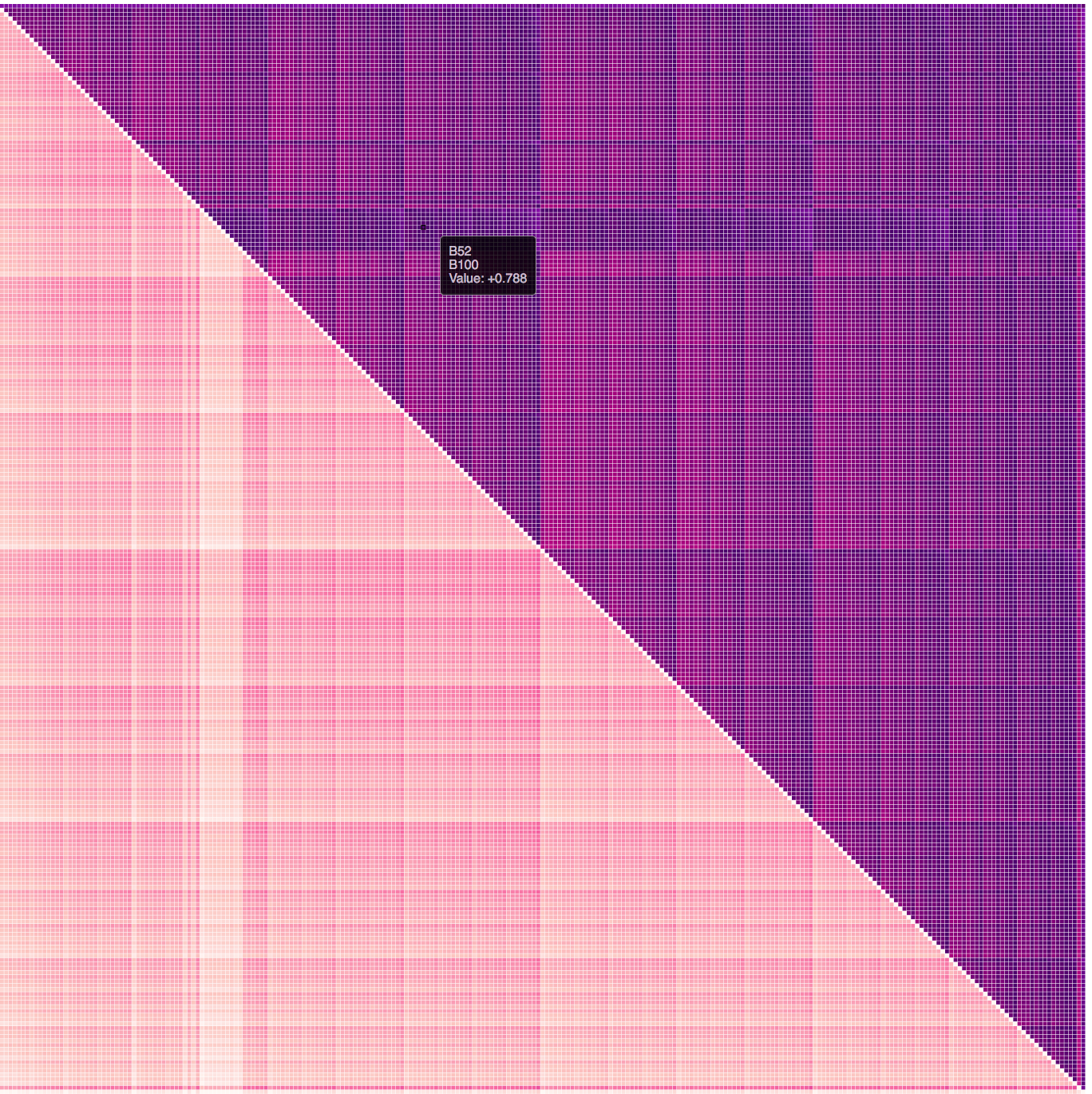
**4b.**



5**.    BFC and BFCC**

1. 2 high correlation, 2 low correlation
2. matlab  32-62 , 95-125 [low]
3. diff\_xml 32-59 [low]
4. rfc822  32-59[high], 59-63[low], 67-120[high]



**6.    FHT**

1. Wrote a java program FHT.java that computes the strength of byte values  at each position. Split the data into a set of 75 and 25. After calculating the fingerprint and comparing with each file in the test set we were able to set a threshold as to what scores of 4-byte, 8-byte and 16-byte headers would be classified as a file belonging to that type.
2. Created a Heat Arrays for each byte position depicting darker areas as possible bytes appearing at that position in a file.

7.    Final analysis

a. Use the knowledge gained from steps 4-6 to identify at least two new MIME magic byte fingerprints (including bytes, offsets and priorities) for all of the 15 types and adapt Tika’s mimetypes.xml file with this information



**Recompile Tika with this new MIME information or point to a new MIME types file via the classpath**

Netcdf files : these files start with String “CDF” followed by Start of Heading (SOH) character.

<mime-type type="application/x-netcdf">

<glob pattern="\*.nc"/>

<glob pattern="\*.cdf"/>

<magic priority="50">

<match value="0x43444601" type="string" offset="0"/>

</magic>

</mime-type>

Po files:

These files are used for translated language code etc. the follow a format of

msgid untranslated-string

msgstr translated-string

<mime-type type="txt/po ">

<glob pattern="\*.po"/>

<glob pattern="\*.pot"/>

<magic priority="50">

<match value="msgid \“.\*\”.\*\nmsgstr \“.\*\”" type="regex" offset="0:8000"/>

</magic>

</mime-type>

we used FHT to analyse the first 16 bytes of octet-stream files and to cluster them in one group based on correlation strength. Basically start with each octet-stream file being its own type and finally forming buckets. A brute force algorithm was used to match each file with every other file. If ye score fell within a threshold then it was merged into the fingerprint.

Using this information were able to get cdf clusters since there scores would be a straight 1 for 4 byte FHT.

It was difficult to analyse .po files using FHT since the characters were simple alphanumeric.

We also found these particular files having the same starting few bytes as distinct but couldn’t find any matching mimetype for it over the internet

c. Write a new program for MIME diversity analysis using your new Tika mimetypes.xml file, and compare the results with the initially classified MIME types from TREC-DD-Polar. Identify if any MIME type classifications changed.

files detected from octet-stream were added as new mimetypes to the chart.

d. Generate a D3 visualization (pie-chart and bar-chart) showing MIME diversity using your updated classifications.

Uses the same visualization as in 3b.

**8.    Tika Similarity**

**9.    Content Based MIME detector**

**s. Were you able to discern any new MIME types within the 200, 000 application/octet-stream (“unknown”) types?**

yes :.

po, cdf, readme files from ndics, grib file format,

gsb, rpm, mov,

**How well did BFA work, compared to FHT?**

**BFA only gives a frequency distribution of bytes occurring in a particular file type. It doesn’t specify**

**Did the D3 interactive visualizations help you understand the byte frequencies, and to identify patterns?**

Yes. D3 was very helpful because once you see a visualization, the file sits in your head and while observing test files one is unconsciously trying to compare another file with a similar looking output and that is really easy to do. Most important approach we took was analyzing the peak values in BFA and then calculated areas of high(dark color)

**Why Tika’s detector was unable to discern the MIME types?**

**Well tika didn’t have a mime magic of a file and hence wasn’t able to discern the mime type of certain files. Also the priorities play an important role in differentiating between extremely similar header files. Tika was not configured to read ‘po’ files which are language translation files. Certain files had characters that coul not be resolved hence we couldn’t even notice what went wrong. Some files had part garbage and part printable characters. Certain files were ideally supposed to be html files but they were stripped off almost all their tags and tika declared it as octet stream.**

**Was it lack of byte patterns and specificity in the fingerprint?**

**Yes. In case of ntcdf files, the missing byte patterns caused it to be detected as octet stream**

**Even still there are some file types which do not any magic bytes at all .**

**It could also be the case where thr priority values being default 50 for many files may have caused misclassification of certain files.**

**Also include your thoughts about Apache Tika – what was easy about using it? What wasn’t**?

Apache tika was not that difficult to use since it was in java and we would simply run it in debug mode to understand in depth of what exactly was happening. This assignment helped us understand open source code, how to modify it build it and use it like new and that was the best part about it. –by Kinara, Nilay