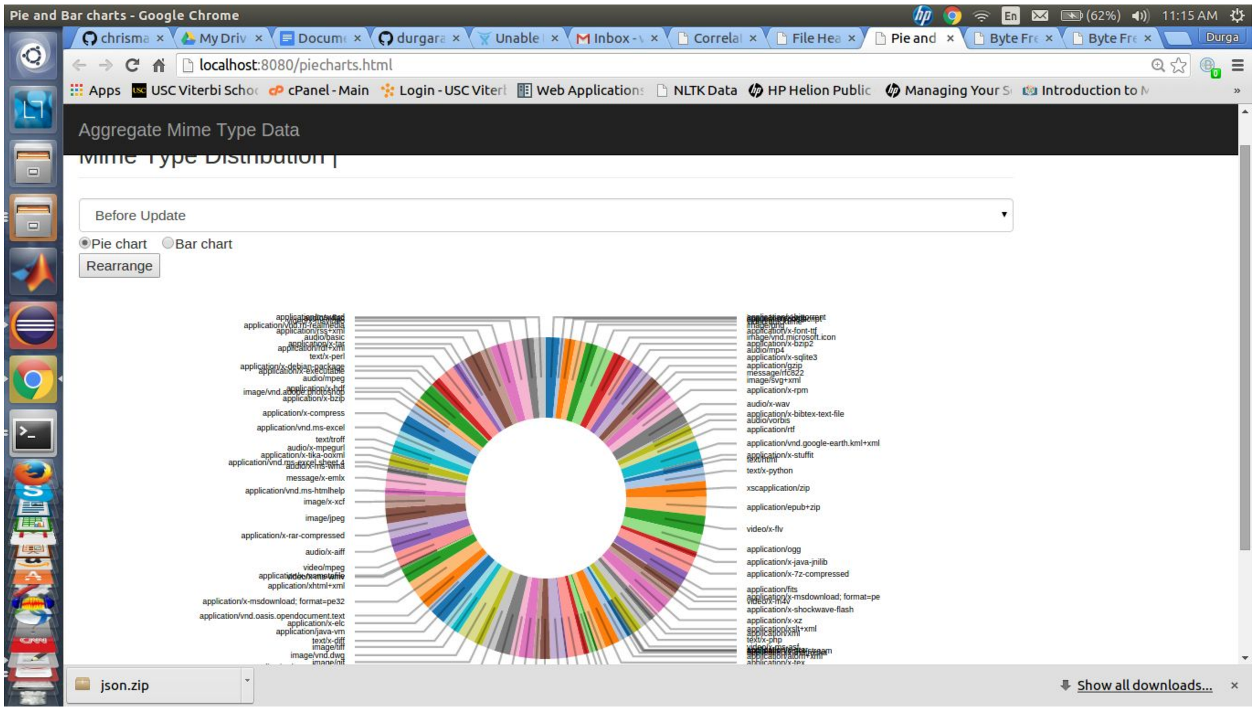
CSCI 599 Assignment 1

Team 9

1 March, 2016

INFERENCES ON DATASET



We used 70 GB of Polar dump data set. There are more than 50+ file formats which are identified by tika. The data distribution is mostly even. We observe more distribution of text and application file formats like application/vnd.google.earth.kml.xml , image/png, image/jpeg, application/rss-xml .

Part 4: Byte Frequency Analysis

In Byte Frequency Analysis, we read the file as an array of bytes and count the occurrences of each byte. Sometimes a certain byte count dominates over other byte counts, in which case we use a technique called compounding if the maximum byte count is greater than or equal to 70% of the total byte counts. This is done by taking the square root of the values obtained. We then normalize the byte counts to get the values between 0 and 1. This information helps to identify certain file formats where certain bytes are dominant.

For example, in the case of an html file the bytes corresponding to characters ‘<’ and ‘>’ are dominant and this information can guide in identifying the html files.
After performing BFA over 70% of the files for each of the file types we used this fingerprint file to identify the file format. We identified by taking the root means square values of differences in the byte value to for the files to be identified.

Inference:
Visualization for BFA is done using an interactive line graph with bytes on x-axis (0 to 255) and byte frequency on y-axis (0 to 1). The visualization helps see the byte count readings and byte value on hovering over graph points. On viewing the visualizations, types like text/html indicate < , > having high byte frequency count. We also see the there is uniform distribution of bytes for application/gzip that indicates that there are no prominent byte for this mimetype.

Part 5: Byte Frequency Correlation
Byte Frequency Correlation is an interesting algorithm that uses correlation strengths between bytes and the fingerprint byte counts as a feature to identify mimetypes. An important information that one can gather from correlation strengths is if a mime type is suited to be classified using the byte frequency method. Also, intuitively, this method checks for frequently occurring bytes in all files of that mime type and puts to use this information to identify mime type beyond BFA.
The formula we chose to calculate correlation strength is the normal distribution function with sigma value = 0.125. (Source: An Algorithm for Content-Based Automated File Type Recognition, Mason B. McDaniel). This measure was marked appropriate since the distribution was centered at 0 and didn’t need a choice of mean and that the peaks were well defined.
The test file values are also companded to ensure uniformity between training and test values. The companding function is the same as BFA where we see if the maximum count is greater than or equal to 70% of sum of other byte counts and only then compand the values.

Inferences
Visualization for BFC is done using an interactive line graph with bytes on x-axis (0 to 255) and correlation strength on y-axis (0 to 1). The visualization helps see the correlation strength readings and byte value on hovering over graph points.
Correlation matrix inference

Correlation matrix indicates how the frequency distribution is spread out. The visualization for this was done using matrix-based heatmap that is a heatmap with discrete blocks for comparing byte vs byte. Light blue indicates near 0 scores and red indicates low correlation and green to dark blue high correlation. In text/html matrix, the bytes 60 and 62 (< and >) indicate high correlation. Also, the distribution doesn't seem to be that uniform which shows that there are bytes that clearly indicate html types and byte frequency analysis is good for html. The application/octet-stream on the other hand has a uniform distribution that is not very indicative of how it might not be as effectively identified as compared to html on byte frequency analysis. On viewing the visualizations, types like text/html indicate a lot of high correlation points (marked in yellow in the visualization), which indicates that byte analysis is a suitable way to discern html mime type. Types like application/gzip and application/x-sh on the other hand had lesser high correlation strength bytes that indicate that it might not be useful to apply byte frequency methods for mime type detection.

Part 6: File Header/Trailer Analysis

In File Header/Trailer Analysis we consider first n header and trailer bytes of a file for analysis. Each fingerprint file for header and trailer is n * 255 matrix , where each row represent a byte considered. For each byte position byte value are scored independently upon the frequency with which byte value occurs at corresponding byte position.
For analysis we considered N to be 4 bytes , 8 bytes and 16 bytes.
If file size is less than number of bytes considered for analysis we won't be considering that file for Trailer Analysis. In header fingerprint for the missing byte position -1 value is assigned for whole row.
Different file formats have varying header lengths and this makes it difficult to decide the optimal number of bytes to be considered for analysis. But FHT does a really good task when it comes to large file sizes, if their corresponding file format has an identifying header as it saves time on reading through the whole file. More file formats can be identified with the header analysis rather than the trailer analysis because the identifying string of the file format is usually the header.

Part 7: Identifying new file formats

Much of the files which were categorized as octet stream were certain scientific file formats. Most of these files had certain unique headers which was easily identified through FHT analysis. After researching on these headers we updated this information in the tika-mimetypes.xml file to detect the corresponding file types. Of the many file formats which we identified we raised three JIRA tickets(for netcdf, fits and hfa) to contribute the changes to Apache Tika. Below is the list of the tika issues raised-

1) TIKA-1875
2) TIKA-1877
3) TIKA-1886

Part 8: TIKA SIMILARITY

For this task, we initially ran with 20 files from each filetype and the results were not as effective. We then collected 5000 files from each mimetype and placed in the same folder. To enable clarity on filetype, we appended the filetype string at the end of the filename since extensions are not present. On running tika-similarity on the data, promising results were obtained.
The Jaccard distance (similarity.py) metric when used produced 7 clusters which was lesser than the expected ideal 15. The results had video and audio types together, and most of the application/* types were placed together. Jaccard similarity

The edit distance and cosine similarity distance measures had 134 clusters and had more well defined clusters than Jaccard distance. The most well clustered type was application/pdf. The cosine similarity metric produced the best clusters with more related files placed together.
The resulting visulaizations for these 3 metrics are placed in the github repository: http://github.com/durgaravi/team9/documentation/tika-similarity

Part 9: CONTENT BASED DETECTION

We consider video/quicktime as the type to be explored as an example. We made a positive dataset consisting of 1000 video/quicktime files and around 200 negative file examples that had different file types that were not video/quicktime files.

A script to get 256 byte counts of each test file in positive and negative dataset was written in python. The dataset was divided into 3 equal sets for training, validation and testing. The python program invokes an R script to save the normalized byte count data as a .hist file. Then the fileTypeDetection R program is also invoked to train the model. This is done by http://github.com/durgaravi/fileTypeDetection/main.py. The results obtained on training video/quicktime files were:

```
[1] "Completed splitting files..."
[1] "Loading Dataset....."
```

```
[1] "Beginning Training Neural Networks"
[1] "the length of weights 517"
[1] "The time taken for training: 7.081000"
[1] "The training error cost: 0.000000"
[1] "The validation error cost: 1.557045"
[1] "The testing error cost: 2.589017"
[1] "Training Accuracy: 100.000000"
[1] "Validation Accuracy: 96.995708"
[1] "Testing Accuracy: 95.299145"
```

The obtained tika.model is then added under tika-core resources folder for detect along with tika-example.nnmodel. We then wrote a client program to detect video/quicktime files from octet-stream classified folders and we were able to detect all video/quicktime files. Out of the 59 files, one was .html and 58 were video files. The model could identify all 58 files as video/quicktime and classified the .text/html file as application/octet-stream.

CHOICE OF MIMETYPES

We chose the following mimetypes:
"application/pdf","text/plain","text/html","application/octet-stream","application/rss+xml"," application/rdf+xml","application/xhtml+xml","application/x-sh","application/gzip","application/msword","image/png","image/jpeg","audio/mpeg","video/mp4","video/quicktime"

We chose the mime types based on the file counts that were specified along with the TREC dataset description. We chose to use types that had large number of files so that the trained fingerprint files would be better.

Even on running mime type detection on the downloaded data, we noticed that a large number of files in the Polardump dataset had html and xhtml+xml files and we believe that this is because most of the files were obtained were webpages using web crawlers.

BFA vs FHT

- Training using BFA analysis takes more time than FHT since we read whole file to get the fingerprint.
- To infer a file format of a large file FHT is more effective since only header and trailer bytes are required to identify the file format.

- BFA analysis sometime may not be effective since it read whole file and that introduces noise in the BFA of the file and may not exactly match to the file formats fingerprint

WHY TIKA COULDN'T DETECT MIMETYPES FOR SOME FILES

1. Why Tika's detector was unable to discern the MIME types?
Tika's detector primarily used Magic number information present in tika-mimetypes.xml. Even though tika covered a lot of filetypes, many file types had different possible magic numbers which could be identified on doing FHT analysis. This information is not present in the current version of tika-mimetypes.xml.

2. Was it lack of byte patterns and specificity in the fingerprint?
Yes. For some file types, the magic numbers are not sufficient. For example: NetCDF, fits, ogg). We need to add another magic number attribute under these filetypes to enable tika to detect.
eg:

```
+    <magic priority="50">
+      <match value="CDF\001" type="string" offset="0" />
+      <match value="CDF\002" type="string" offset="0" />
+    </magic>
```

3. An error in MIME priority precedence?
There are possibilities of this happening. There were cases where application/octet-stream had high priorities as compared to some rare filetypes which made them classified as application/octet-stream. However, since the mimetype xml we had most of the priorities set as 50 and were equal, there was limited chances of this being the reason as compared to the magic number being missed out.

4. Lack of sensitivity in the ability to specify competing MIME magic priorities and bytes/offsets?
When we updated the new mime types, changing the magic priority didn't make any impact on the detection feature and this was probably because of inability to compare across types. The structure of tika-mimetypes.xml could be refined to make priority attribute more significant in the detection process.

ABOUT USING TIKA

For the assignment, we used both the tika java jar and tika in Python.

The tika-mimetypes.xml needs some restructuring. There is no proper documentation about the usage and importance of magic priority. When we updated the new mime types, changing the magic priority didn’t make any impact on the detection feature of tika.
Observed some minor issues while using tika, where identifying the fits file format from the command line returned a null string but when the same file was used to detect its type by calling the detect method on the Tika object it returned “application/fits”.