CSCI–572 ASSIGNMENT 1

Crawling using Apache Nutch

Dark and deep web are two potential places to sell guns illegally. We used Apache Nutch to crawl for images of guns, Apache Tika for content and metadata extraction, and Selenium to automate Ajax interaction and more.

Team #32

Sanjay Jagadeesh

Manohar Thagadur Nataraju

Soumya Gowda Basvana

Pramod P. Setlur

CSCI–572 Assignment 1

Question 1: Download and configure Nutch to crawl Weapons images as identified in the seed list.

a. The list of files we modified to deal with politeness are as follows:

   Nutch-Site.xml:

   1. We set http.agent.rotate to 'true' and added 10 agent names in agents.txt.
   2. We increased the http.timeout from 10,000 to 80,000.
   3. We changed the http.content.limit to -1 to remove the limitation on the size of the crawling file.
   4. We increased the fetcher.threads.fetch from 10 to 20.
      Our Nutch-Site.xml contains the other changes.

b.

   i. Regex-urlfilter.txt and Automaton-urlfilter.txt
      1. We skipped domains like facebook.com, tumblr.com, twitter.com, etc. to filter majority of images unwanted to us.
      2. We removed the image extensions to avoid skipping images.

   ii. Mimetype-filter.txt:
      1. We added image/* mimetype to allow crawling of documents with images/* and text/html mimetype.

   iii. Suffix-urlfilter.txt:
      1. We commented all the image suffixes to prevent them from exclusion.

c. We enabled rotating bots in agents.txt.
   i. Spider#32Crawler, MasosapaCrawler, etc

Question 2: Perform crawls of Weapon's images sites

a. We used the NutchPy library
b. We wrote a python script to generate all the image mimetypes that were found in our crawled data.
c. We found the following mime types:
   1. image/png
   2. image/jpg
   3. image/jpeg
   4. image/gif
   5. image/bmp
   6. image/x-icon
   7. image/tiff
   8. image/x-xbitmap
   9. image/pjpeg

Many of the URLs that weren't fetched were mainly because of java.net.SocketTimeoutException, java.lag.IllegalArguementExeption, HTTP 403.

d. The list of 100 Urls that we found difficulty in fetching has been added in "*URLs not crawled.xlsx*"
e. We crawled all the 85 urls with 30 rounds
   The crawl stats at this stage is tabulated as follows:-

```
$ bin/crawl urls/ ~/CSCI-572/guns_crawl_Step2 30
$ bin/nutch readdb ~/CSCI-572/guns_crawl_Step5/crawldb -stats
```

Total URLS	4710184
retry 0:	4625184
retry 1:	64302
retry 2:	20698
Min Score	0.0
Avg Score	2.9849576E-5
Max Score	1.173
Status 1 (db_unfetched)	4348772
Status 2 (db_fetched)	314453
Status 3 (db_gone)	28243
Status 4 (db_redir_temp)	5114
Status 5 (db_redir_perm)	6146
Status 6 (db_notmodified)	7
Status 7 (db_duplicate)	7449

Question 3: Use the information you learn in question 2 to extend the Nutch Selenium Protocol Plugin

a. We used the protocol-interactive-plugin for Nutch and enabled it to get past the pages that required Ajax interaction such as paginating and form login.
b. We wrote *Custom Handlers* trying to generalize a pattern for the websites that required a login or those which were paginated.
c. Upon building and testing Nutch with the plugin enabled we could see Firefox pop open and Nutch executing our *Custom Handlers*.

Question 4: Build the latest version of Tika and install Tesseract

a, b, c. We installed Tesseract and upgraded Tika to latest version.

Question 5: Re-run your Weapons Crawls with enhanced Tika and Nutch Selenium you have built in step 3

a. The list of 100 Urls that we found difficulty in fetching has been added in "*URLs Not Crawled Selenium.xlsx*"
b. About 60% of the Urls from 2d are still present in the result of this step. The exact urls can be found in the above excel sheet.

c. Some URLs were not fetched because the corresponding mime types couldn't be parsed by Nutch. Tika parsing enabled to overcome this hindrance and parse additional mime type related to images.
d. The crawl stats at this stage is tabulated as follows:-

```
$ bin/crawl urls/ ~/CSCI-572/guns_crawl_Step5 30
$ bin/nutch readdb ~/CSCI-572/guns_crawl_Step5/crawldb -stats
```

Total URLS	4067987
retry 0:	3898790
retry 1:	35875
retry 2:	9765
Min Score	0.0
Avg Score	2.3457E-3
Max Score	1.173
Status 1 (db_unfetched)	3267867
Status 2 (db_fetched)	340989
Status 3 (db_gone)	34075
Status 4 (db_redir_temp)	4226
Status 5 (db_redir_perm)	8632
Status 6 (db_notmodified)	11
Status 7 (db_duplicate)	6508

Question 6: Develop two deduplications to use on the extracted text and meta data in the parsed content from Nutch

a. We developed an algorithm to find exact duplicates. Here is the gist of the algorithm:
   - Read the dump file generated after reading the segments.
   - Filter this file with URLs containing only JPGs, PNGs, GIFs and other image mimetypes.
   - Using *tika-parser,* we extracted the metadata for the above image URLs.
   - Extract every fifth character in the metadata to create a fingerprint.
   - Hash the fingerprint using MD5 and store all the found duplicates in a list.

b. We developed an algorithm to find near duplicates using Simhash:

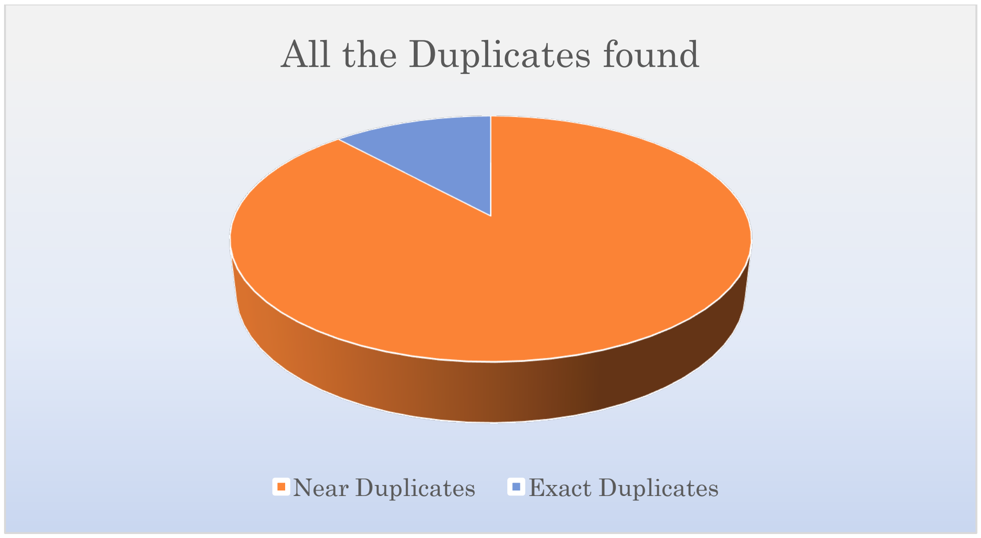
The algorithm we employ to detect near duplicate is a simple implementation of the simhash technique explained in Manku et al., WWW 2007, **"Detecting Near-Duplicates for Web Crawling"**.

We used 128 bit md5 hash to generate Simhash for a given string. The string here is the metadata of the image extracted using tika parser. The metadata mainly includes the properties of the image such as file size, image width & height, X & Y resolution, compression type, content type, etc.

The Algorithm is described below:

- Define a global fingerprint size (of 128 bits). This can be changed to use 32 bits as well.
- Create a global fingerprint array (global_fingerprint[ ] filled with size of zeros)
- For the given metadata of the image, generate shingles of size 4
- For each shingle, generate its md5 hash and examine the bits of the hash.
    - If the bit is 0, add 1 to global_fingerprint[i]
    - If the bit is 1, remove 1 from global_fingerprint[i]
- For each bit j of the global_fingerprint
    - If global_fingerprint[j] >=0, set global_fingerprint[j] = 1
    - If global_fingerprint[j] < 0, set global_fingerprint[j] = 0
- Repeat the above steps to generate the simhash for metadata of all images.
- Compare Simhash to determine the hamming distance. Hamming distance between any two Simhash is obtained by 'xor'ing the bits in two Simhash. If the hamming distance is about 10 or less it indicates the two Simhash differ in over 10 bits and hence similarity is around 90%
- If the two Simhashes are similar by about 90% or more we classify them as near duplicates.

Thus the Simhash Algorithm is successfully able to classify the image urls as near duplicates by comparing the hamming distance of the Simhashes which is obtained using the metadata of the image.



Question 7: Enable the Nutch similarity scoring filter focused crawling plugin

a. We went through the documentation of Similarity Scoring Filter and understood the main purpose of the scoring filter.
b. We enabled this plugin and re-ran the weapon crawls.
c. We added the goldstandard.txt with all words that were relevant to weapons. And, we added the stop words that are meant for inverse document frequency inside the stopwords.txt. We could see improvements in the results of near deduplication algorithms.
d. The crawl stats at this stage is tabulated as follows:-

```
$ bin/crawl urls/ ~/CSCI-572/guns_crawl_Step7 30
$ bin/nutch readdb ~/CSCI-572/guns_crawl_Step7/crawldb -stats
```

Total URLS	3568756
retry 0:	25687
retry 1:	8643
retry 2:	6489
Min Score	0
Avg Score	2.36387E-3
Max Score	1.932
Status 1 (db_unfetched)	2878690
Status 2 (db_fetched)	45876
Status 3 (db_gone)	5435
Status 4 (db_redir_temp)	7456
Status 5 (db_redir_perm)	4563
Status 6 (db_notmodified)	86
Status 7 (db_duplicate)	12564

EXTRA CREDITS

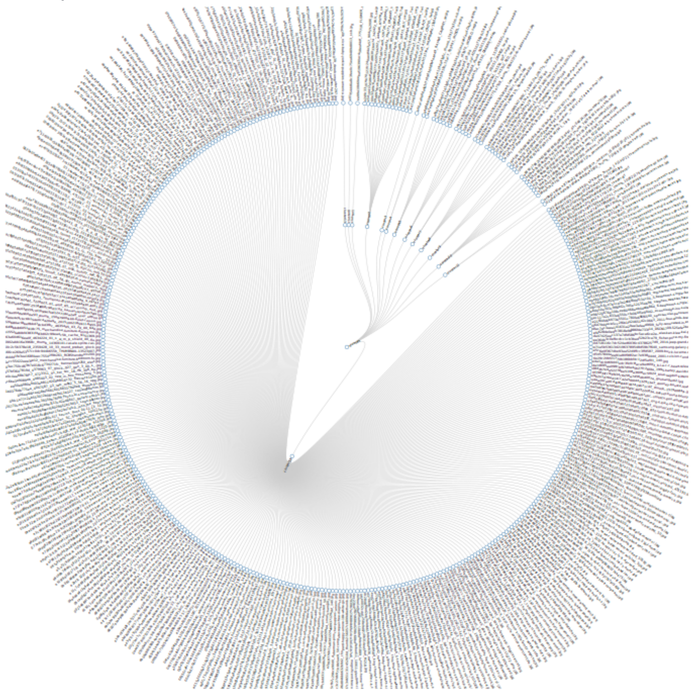
Question 8: Download and configure Nutch python which you will use to control your crawls

a. We went over the documentation and wrote a python program to crawl with Nutch using Nutch rest server and Nutch python.
b. The python program *Nutch_python_crawl.py* is attached.
c. We went through the documentation and implemented the best practices for crawling using Nutch rest server.
d. NutchPy is a python library for working with Apache Nutch. It has functionality to work with Nutch data structures – to read Sequence files, LinkDb, etc. We feel these

functionalities can be added inside the *Nutch REST server* as an API. It can accept a link to the the segments generated and further processing of can be performed by the *Nutch REST server*.

Question 9: Dump the crawl data out of your Nutch content and then run tika-simlarity over it

a. Here is what tika-similarity inferred with all the images that Nutch was able to crawl successfully.



b. We could see an interesting pattern in the D3 that was generated by *tika-similarity*. Many images that were actually guns followed a noticeable pattern.
c. Cluster 0 consisted of about 30,000 images many of which were guns. Cluster 1 contains images of that can be excluded from guns.

Question 10: Re-run your crawls using Memex Explorer, a domain specific search tool.

a. We didn't encounter any bugs while crawling using Memex Explorer.
b. Yes we were able to run our crawls successfully. Here are the steps we followed to crawl using Memex:
    - We installed anaconda from https://www.continuum.io/downloads.
    - We forked the Memex explorer project from Gthub.
    - We created a Memex environment in anaconda and crawled using *crawl urls/ ~/CSCI-572/guns_crawl_Step10 3.*

- The above output directory was created inside *anaconda/envs/memex/lib/nutch*
- All the segments were created as expected inside the output directory under the Nutch directory.

c. We didn't observe any capabilities that were missing from Memex Explorer's Nutch.

CONCLUSION

The goal of the assignment was to collect as many unique images of guns and weapons on the provided seed lists. We could get around 107.000 images.

We faced multiple challenges throughout this project. Here are the major challenges:

- Developing an interactive selenium to interact with dynamic Ajax based content which enabled us to get into deep web and collect more images of weapons. We had to downgrade the Firefox version to 33.0 to successfully enable the protocol-interactive-selenium plugin.
- Identifying the duplicate images was one another major challenge. The project led us to a thought process as to why the images were duplicated. Most of the times it happens that there may exist the same content in different regions (example, we may have same page in .com domain, .co.uk, .co.in, etc.) which may result in duplication of web content. We developed algorithms to identify exact and near duplicates. The challenge was in extracting the metadata for the images based on which they could be categorized into exact or near duplicate images. In addition, the duplicated URLS were filtered in the crawling phase itself.
- Initially we faced one of the challenge in increasing the nutch heap memory size where in the crawling aborted after running for 2 days due to insufficient heap memory. We figured out the configuration parameter JAVA_HEAP_MAX in bin/nutch script to increase the size of the heap memory.
- Configuring the nutch crawler to deal with politeness and completeness. This enabled the crawler to increase its sensitivity to the way it appeared in web logs for servers providing weapons oriented content on the Internet.
- Getting to work with the various open source technologies. Most of the times we couldn't get much documentation in installing or integrating the technologies with apache nutch. This led us to actively engage in the discussions in apache nutch forums, post our queries in the forums and to even contribute back to the forum by answering other's queries.