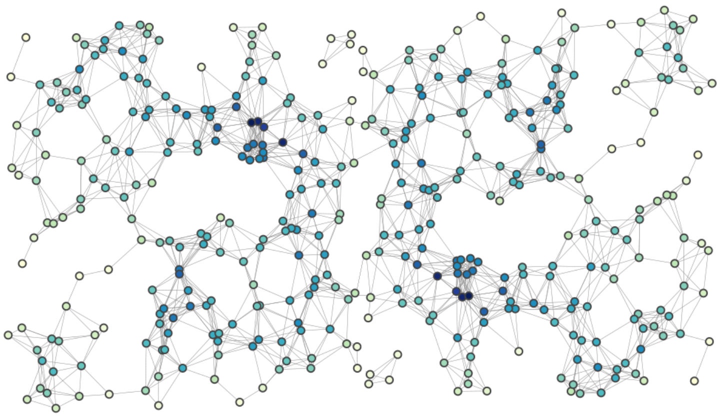

OLD DOMINION UNIVERSITY

CS 432 WEB SCIENCE

Assignment Three

Derek Goddeau

Professor

Michael L. Nelson

February 23, 2017

1 Download the HTML for the 1000 URIs

To download the HTML for the URIs the bash script `get_html.sh` is used. The `fetch()` function does all the work, downloading the HTML using `wget` and creating a `SHA-1` hash of the URI to store locally.

```
fetch() {
    while read uri; do
        local hash=$(echo -n "$uri" | sha1sum | cut -d ' ' -f 1)
        local hash+=".html"
        wget -O data/raw_html/"$hash" "$uri"
    done < "$FILE"
}
```

Then to remove the HTML and leave only words the `word_soup` program was used. The program is multithreaded and accepts an `-t` parameter to specify the number of threads to use. Because BeautifulSoup is used, and it uses recursion internally to find child elements the `-r` flag is available to change the default Python recursion limit 1000 to prevent `RuntimeError` from being thrown on very large documents, but the system must still have the memory to handle the maximum runtime recursion level. Finally input and output files must be specified.

After spawning the number of threads specified, before parsing the file the MIME type is checked using the `magic` package, if it is not "`text/html`" then it is ignored. Then `BeautifulSoup` is used to parse the HTML using the `lxml` parser and the unwanted portions of the document are removed using list comprehensions over the elements and the `extract()` method. The result for each is written to a file named after the hash concatenated with "`.txt`" in the specified output directory.

2 Calculate TFIDF

Mixing bash with Python using bash magic in a Jupyter notebook a list of all files with the term TERM is created. Then from the list using the Python random package a random sample of 10 URIs is chosen.

```
TERM = 'hacker'
term = ' ' + TERM + ' '
files = ! grep -H -r "{term}" ../data/words/* | cut -d ':' -f 1 | sort -u

import random
files = random.sample(files, 10)
```

From there the data needed to preform the calculations and build the table are collected into a pandas dataframe and written to a CSV file for safe keeping and import to R. While Python does have many capable utilities now for dealing data sets, its natural syntax does not lend itself to vectorized calculations like other mathematically specialized languages such as R and GNU Octave. The formulas used to compute the values are as follows,

$$\begin{aligned}\text{Let } t &= \text{Term}\\ \text{Let } IDF &= \text{Inverse Document Frequency}\\ \text{Let } TF &= \text{Term Frequency}\end{aligned}$$

$$\begin{aligned}TF &= \frac{\text{term frequency in document}}{\text{total words in document}}\\ IDF(t) &= \log_2\left(\frac{\text{total documents in corpus}}{\text{documents with term}}\right)\end{aligned}$$

The value for `corpus_size` from `http://www.worldwidewebsize.com/` February 23, 2017and `docs_with_term` was obtained from google on February 23, 2017. The equations which are performed in the following code are the same but vectorized so that the division becomes elementwise division of vectors,

$$\begin{bmatrix} x_1 \\ \vdots \\ x_n \end{bmatrix} \odot \begin{bmatrix} \frac{1}{y_1} \\ \vdots \\ \frac{1}{y_n} \end{bmatrix} = \begin{bmatrix} \frac{x_1}{y_1} \\ \vdots \\ \frac{x_n}{y_n} \end{bmatrix}$$

```
%%R

df.words <- read.csv('../data/tfidf.dat')
corpus_size <- 4460000000
docs_with_term <- 230000000

df.words$TF <- (df.words$term_occurrences / df.words$word_count)
idf <- log2(corpus_size / docs_with_term)
df.words$TFIDF <- (df.words$TF * idf)

df <- arrange(df, -TF.IDF)
latex <- xtable(df)
digits(latex) <- c(0, 3, 3, 3, 0)
```

Finally the `plyr` package is used to arrange the data in the proper order and `xtable` is used to generate a LaTeXtable with the desired significant digits.

TF-IDF	TF	IDF	URI
0.019	0.004	4.277	`www.generation-nt.com/nsa-harold-martin-vol-outils-hacking-tao-actualite-1939093.html`
0.010	0.002	4.277	`www.tekbotic.com`
0.008	0.002	4.277	`www.securezoo.com`
0.007	0.002	4.277	`www.govtech.com/security/national-cybersecurity-center-works-quickly-to-help-its-clients-recover.html`
0.007	0.002	4.277	`www.24brasil.com`
0.004	0.001	4.277	`www.grahamcluley.com/smashing-security-podcast-007-ascii-art-attack/`
0.003	0.001	4.277	`www.rtinsights.com/blockchain-technology-and-iot-security/`
0.003	0.001	4.277	`www.esquire.com/news-politics/a49791/russian-dnc-emails-hacked/`
0.003	0.001	4.277	`www.vanguardngr.com/2017/02/6-ways-bank-account-can-hacked/`
0.001	0.000	4.277	`www.theatlantic.com/technology/archive/2017/02/what-happened-to-trumps-secret-hacking-intel/515889/`

3 Rank by PageRank

The PageRank scores below were obtained using `http://pr.eyedomain.com/` and the URIs have been adjusted to show that they only calculate PageRank for the domain not the actual page.

PageRank	URI
0.8	`www.theatlantic.com/`
0.7	`www.esquire.com/`
0.6	`www.vanguardngr.com/`
0.5	`www.generation-nt.com`
0.5	`www.grahamcluley.com/`
0.4	`www.rtinsights.com/`
0.0	`www.securezoo.com`
N/A	`www.tekbotic.com`
N/A	`www.govtech.com/`
N/A	`www.24brasil.com`

Table 1: Source: `http://pr.eyedomain.com/`

The PageRank results are only concerned with the popularity of the site and not the content, for example a news site may only have one page that is relevant but is popular so it has more links and a higher page rank, but a site entirely devoted to the topic that has very relevant pages would be very low. In this regard the results from the TF-IDF calculations are a much better metric for topic. None of the sample URIs are of very high relevance and the sample is very small so it is hard to speculate, but both popularity (PageRank) and TF-IDF can certainly have false positives for relevance, perhaps the set of URIs with the highest results with both would hold the most relevance.