Introduction to Digital Libraries Assignment #3

James Tate II

March 31, 2015

1 Introduction

Assignment #3 required comparing the downloaded webpages from assignment #1 with and without the HTML templates[2]. I attempted to remove HTML templates using jusText, a heuristic boilerplate removal tool[1]. The input and output text of jusText was compared and jusText's performance was analyzed.

2 Methodology

I wrote two scripts for this utility, both of which are available in my git repository on GitHub[1].

2.1 jusText

The first step of this assignment was to run the webpages through jusText. To facilitate this, I generated a couple lists and wrote a utility. Listing 2.1 show how I generated a list of all files in my tweets directory.

Listing 2.1: Generating list of files in tweets directory

```
find ./tweets/ > tweets_file_list
```

Listing 2.2 shows how I generated the list of unique final URIs. The utility used is one I created in assignment #1.

Listing 2.2: Generating list of unique final URIs

```
./summary.py -r tweets.summary.json -Mm 10000 \
    > uniq_final_uris
```

Both of these lists are hard-coded in the *run_boilerpipe.py* utility shown in Listing 2.3. This utility identifies one representation of each unique final URI, and runs boilerpipe on that instance. It is necessary to filter the input, because I carelessly downloaded multiple copies of the same resource in the first assignment. The output from boilerpipe is saved in a *boilerpipe.output* file in the same directory as the previously saved representation.

Listing 2.3: Running jusText

```
./run_boilerpipe.py
```

[1] https://github.com/jamesbtate/cs851-s15

This utility makes one call to the jusText command in Listing 2.4 for each input file. The values *output-file* and *input-file* were automatically replaced with the correct values on each call.

Listing 2.4: Actual jusText command

```
python -m justext -s English -o output-file input-file
```

2.2 Word Counts

The second part of this assignment asked for the frequency of terms in the input and output to jusText. To easily calculate this data, I concatenated the unique downloaded representations into one file, and the same representations after they were processed by jusText into another file. These concatenations are shown in Listing 2.5. The extra *echo ""* is there to make sure any downloaded representations without a trailing newline are properly delimited from the next representation. Otherwise, there would be a potential to erroneously combine the last word of one document with the first word in the next document.

Listing 2.5: Concatenating input and output data.

```
grep "boilerpipe.output" tweets_file_list \
    | xargs cat > concat_boilerpipe
while read line; \
    do cat "$line" >> concat_original; \
    echo "" >> concat_original; \
    done < uniq_uri_content_files
```

The two concatenations, *concat_boilerpipe* and *concat_original*, were processed twice each by the *word_count.py* utility as shown in Listing 2.6. The default behaviour is to treat any term separated by whitespace as a "word" and to count the frequency of each unique "word," after converting all text to lowercase. With the *-l* flag, only consecutive letters, apostrophes and hyphens are treated as words. Apostrophes and hyphens are only accepted if they immediately follow a letter.

Listing 2.6: Counting words in concatenated data

```
./word_count.py concat_boilerpipe \
    > boilerpipe_words
./word_count.py -l concat_boilerpipe \
    > boilerpipe_words_letters-only
./word_count.py concat_original \
    > original_words
./word_count.py -l concat_original \
    > original_words_letters-only
```

3 Results

This section discusses the results of the boilerpipe program and analyzes the frequent words in the original representations and the versions that have been processed by jusText. The most common words in each dataset are compared to a stop word list, and the word frequencies are graphed.

3.1 jusText

Of my original 5,629 unique final URIs, 124 had to be excluded because they were not HTML. Most of the 124 were MP3 files of podcasts, and a few were PDF files. The output from the commands shown in Listing 3.1 show that 2,427 of my 5,505 remaining unique URIs had no output from jusText (because the output file has zero lines). This indicates a failure of jusText for 44% of my URIs.

Listing 3.7: Counting failed jusText runs

```
grep "boilerpipe.output" tweets_file_list \
    | xargs wc > wc_boilerpipe
grep " 0 " wc_boilerpipe | wc -l
```

Table 3.1 includes the number of total words, unique words and total characters in the input and output data. It has results for both filtering modes of the *word_count.py* utility.

Table 3.1: Word Count Data

Data Point	**Original**	**After jusText**[2]
Total bytes	646,619,835	12,133,748
Total words	33,594,568	2,035,935
Unique words	9,135,191	121,422
Total letter words	81,318,873	2,030,636
Unique letter words	1,271,593	57,434

3.2 Word Frequency

3.3 Stop Words

Appendices

A Streaming API Filter Keywords

These keywords were selected arbitrarily. Keywords were added to the list until the streaming API seemed to pull tweets at a strong, consistent rate.

- python
- fsf
- foss
- coding
- programming
- fedora
- rhel
- dovetail
- woodworking
- blizzard
- snowstorm
- colorado
- virginia
- internet
- library
- libraries
- json
- lemonade
- woodchuck
- iasip
- league
- awesomenaut
- tf2

References

[1] Michael Belica. jusText 2.1.0. `https://pypi.python.org/pypi/jusText/2.1.0`, 2014. Accessed: 2015-03-27.

[2] James Tate II. CS 751 Assignment #3. 2015.