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. Note that only the output from the successful jusText invocations is counted in this data, because there was no output from the unsuccessful invocations.

Table 3.1: Word Count Data

Data Point	**Original**	**After jusText**
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

Most of the webpages that failed jusText processing have very little textual content. Many of these webpages are specific images on Facebook or specific tweets on twitter. A few have significant content in the form of an article, but the article is only accessible after navigating through some advertisement or splash page. Without examining the source of the program, it appears jusText may fail if it determines the input document does not have enough text to be anything meaningful.

Most of the successful jusText invocations were on news articles. One of my search terms in assignment #1 when downloading tweets was “blizzard.” This search term yielded many news articles regarding the then-recent snowstorm in the U.S. northeast. These types of documents work well in jusText.

There were also some successes on content that was not primarily text, such as the page for a YouTube video. It is possible that YouTube is such a popular site, that an algorithm for processing its webpages is hard-coded in the jusText utility. The output from jusText for YouTube video webpages is only the description of the video.

3.2 Word Frequency

Table 3.2 shows the top 50 most frequent words from the original downloaded webpages and from the successful outputs of jusText. The last column in the table shows words that were common to both top 50 lists.

Table 3.2: Word Count Data

Word Rank	**Original Words**	**jusText Words**	**Common Words**
1	<div	the	the
2	</div>	to	to
3	<a	and	and
4	=	<p>	
5	<span	of	of
6	the	a	a
7	{	in	in
8	to	is	is
9	}	that	that
10	/>	for	for
11	and	on	on
12	of	you	you
13	a	with	with
14	<li	it	
15	in	as	
16	</li>	are	
17	</a>	be	
18	var	this	this
19	–>	i	
20	<meta	at	
21	<script	have	
22	>	from	
23	</span>	or	
24	for	was	
25	<!–	by	by
26	<li><a	your	your
27	on	but	
28	+	will	
29	0	an	
30	if	not	if
31	is	we	
32	<img	<h>	
33	-	he	
34	</script>	they	
35	<link	can	
36	this	new	new
37	<option	has	
38	you	more	
39	with	their	
40	”>	his	
41	<li>	if	
42	</ul>	all	
43	</td>	about	
44	:	how	
45	your	one	
46	by	who	
47	</tr>	our	
48	new	what	
49	that	when	
50	<tr>	up	

3.3 Stopwords

Using the default English stopwords list from Ranks NL, I determined how many words from the top 50 lists are stopwords. In the original content top 50 words list, which clearly contains a lot of punctuation that would not normally be found on a stopwords list, 16 of the words are stopwords. These 16 words happen to be the 16 words common between the original top 50 words and the top 50 jusText words, except the word *new*. Of the top 50 jusText words, 44 of them are stopwords.

3.4 Distribution of Words

A Zipf distribution is a ranking of frequencies in which the rank of an item is inversely proportional to its frequency[3]. Both the original words and the jusText output words, when ordered by frequency, obey Zipf's Law and follow a Zipf distibution. This can be seen in Figures 3.1 and 3.2, where the plot closely follows the axes, similar to the plot of the continuous function $f(x) = 1/x, \forall x > 0$. Only the first 1,000 most frequent words in each dataset is plotted to make the curve more visible to the reader.

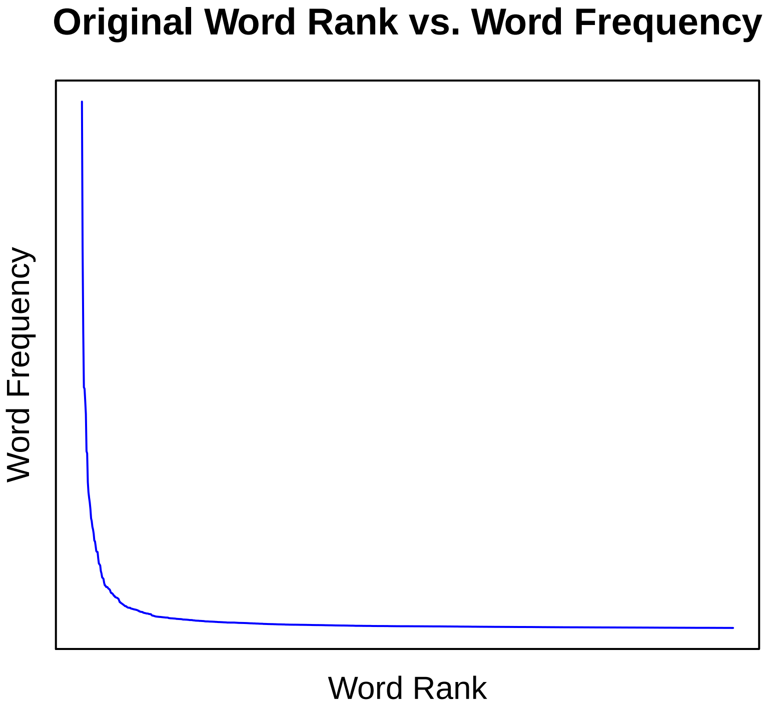

Figure 3.1: Rank of Original Words vs. Their Frequency

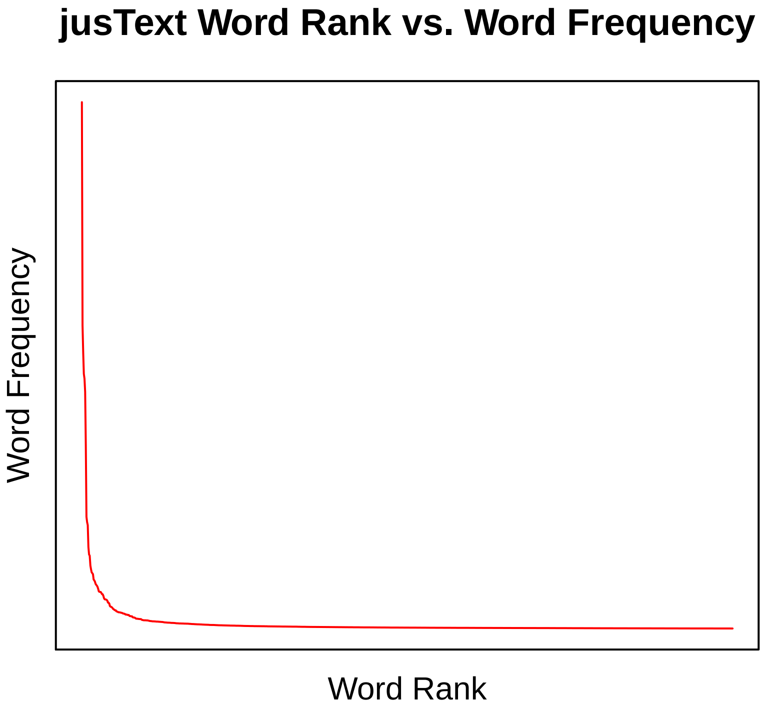

Figure 3.2: Rank of jusText Words vs. Their Frequency

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.

[3] Wikipedia. Zipf's law — wikipedia, the free encyclopedia, 2015. [Online; accessed 2-April-2015].