Evaluating Sentiment Analysis Tools Against Developer Commit Logs

Zachary Szewczyk
Department of Computer Science
Youngstown State University
Youngstown, Ohio
zjszewczyk@student.ysu.edu

***Abstract*—This paper examines the performance of two popular sentiment analysis algorithms, *SentiStrength* and *Stanford CoreNLP*, when used to evaluate developer commit logs. It then compares their performance to an up-and-coming sentiment analysis algorithm, *SentiCR*, created with the express intent of analyzing the type of short sentences found in developer commit logs.**

***Index Terms*—Sentiment analysis, SentiStrength, Stanford CoreNLP, developer commits, SentiCR**

I. Introduction

Sentiment analysis tools have found great popularity in the marketing field in particular, for their ability to quickly process vast amounts of data and provide companies with an accurate evaluation of the positive, negative, and neutral feelings large numbers of people harbor toward their products. [1] As many before me have already pointed out, though, the success those tools have found in that space does not necessarily translate well to the software engineering realm: given developers' tendency to write succinct and terse code review comments, and include technical jargon, code snippets, and other such information like URLs in their comments, today's popular sentiment analysis tools are not the silver bullet for processing developer comments that they have been elsewhere, in other industries. [2]

This paper compares SentiStrength, Stanford CoreNLP, and SentiCR on a commit-by-commit basis in order to:

1) Provide concrete examples of inconsistencies across these tools, and
2) Compare the performance of these tools against the gold standard, a human's analysis of the sentiment present in a subset of the commits processed by the algorithms.

II. Dataset

In order to capture a large data set for this project, I used Boa. Boa, a distributed data mining service that streamlines the process of querying hundreds of thousands of software development repositories, yielded a data set of 41,338 commit messages.

III. Methodology

After capturing the commits, some refactoring was necessary to transform Boa's output into a format SentiStrength, Stanford CoreNLP, and SentiCR could read. This entailed removing line prefixes in the case of ordered or unordered lists, and concatenating multi-line commits to make a single line. This ensured all three tools considered each commit in its entirety, and did not erroneously discard any data. A second script then aggregated the tools' output and compared them against each other on a commit-by-commit basis; for any commits whose categorization differed between the three, it generated a report containing the commit text, and the categorization–positive, negative, or neutral–from each sentiment analysis algorithm.

IV. Tools

The first tool I used in this project was Boa. The following script, in Figure 1, yielded a single text file containing 41,338 developer commit messages, when run against Boa's "2015 September/GitHub" data set:

```
foreach (i: int; def(p.code_repositories[i]))
    foreach (j: int; def(p.code_repositories[i].
        ↪ revisions[j]))
        if (!match("empty log message", p.
            ↪ code_repositories[i].revisions[j
            ↪ ].log))
            commits[p.id] << p.code_repositories[
                ↪ i].revisions[j].log;
```

Fig. 1. Boa commit retrieval code

The next script, contained in Figure 2, parsed the Boa output into a format the sentiment analysis algorithms could read, and provided some administrative data so that I could ensure all commits were being processed by the aforementioned tools.

```
# Function to strip special characters from
    ↪ beginning of each line
def sanitize(line):
    # Remove whitespace, ex. from indentation
    line = line.strip()
    line = re.sub("^[\*-]\s*", "", line) #
        ↪ Remove bullets from bulleted lists
    line = re.sub("^[A-Za-z0-9]\)\s+", "", line)
        ↪  # Remove item identifiers from alpha
        ↪ -numeric lists
    return line

...

# Output admin data
print "Number of commits processed: %d" %
    ↪ commit_count # Confirm with regex:
    ↪ commits\[[0-9]+
print "Number of blank commits: %d" %
    ↪ blank_commits # Confirm with regex:
    ↪ commits\[[0-9]+\]\s=\s+$
print "Number of commits algorithm should
    ↪ process: %d" % (commit_count-
    ↪ blank_commits)
```

Fig. 2. Commit normalization code

Once parsed, the script in Figure 3 fed the data to SentiStrength and recorded the results in an output file. Two other, similar scripts performed the same function for Stanford CoreNLP and SentiCR.

```
for line in fd:
    line = line.strip()
    line = re.sub("\t", ",", line)
    sentiment = int(line.split(",")[0])
    print "SENTIMENT:␣%d" % sentiment
    print line
    if (sentiment == 0):
        neutral += 1
    elif (sentiment == 1):
        positive += 1
    elif (sentiment == -1):
        negative += 1

print """\
Pos Neg Neu
%-6i%-6i%-6i
""" % (positive, negative, neutral)
```

Fig. 3. Sentiment analysis code

The final script, below in Figure 4, compared the output from each sentiment analysis algorithm, and reported any discrepancies between the three.

```
for line in c:
    if (re.search("sentimentValue", line)):
        core_nlp = line.split("sentiment=")[1].
            ↪ replace("\"", "").replace(">",
            ↪ "").replace("Verynegative", "
            ↪ Negative").replace("Verypositive
            ↪ ", "Positive").strip()
        # print core_nlp
        s_line = s_fd.readline()
        senticr_line = senticr_fd.readline()
        temp = int(s_line.split("\t")[0])
        if (temp == -1):
            sentistrength = "Negative"
        elif (temp == 0):
            sentistrength = "Neutral"
        elif (temp == 1):
            sentistrength = "Positive"
        if not (core_nlp == sentistrength ==
            ↪ senticr_line):
            i += 1
            print "\""+s_line.split("\t")[1].
                ↪ strip()+"\""
            print " -- Stanford CoreNLP rates
                ↪ this as %s, SentiStrength
                ↪ rates this as %s, SentiCR
                ↪ rates this as %s." % (core_nlp
                ↪ , sentistrength, senticr_line)
            print
```

Fig. 4. Algorithm results comparison code

V. Results

Table 1 contains the results from each algorithm, as a raw number of commits, out of the 41,338 commits processed processed in this data set.

TABLE I
Algorithm Results by Commits

Algorithm	Positive	Negative	Neutral
SentiStrength	3516	5827	31994
Stanford CoreNLP	3840	17408	19770
SentiCR	90	189	41059

Table 2 contains the results from each algorithm, as a percentage of the total number of commits processed in this data set.

TABLE II
Algorithm Results by Percentages

Algorithm	Positive	Negative	Neutral
SentiStrength	8.51%	14.10%	77.40%
Stanford CoreNLP	9.30%	42.87%	47.83%
SentiCR	.22%	.46%	99.33%

Table 3 contains the results from hand-categorizing a randomly selected subset of 1,000 commits from the

dataset, as both a raw number of commits in each category as well as a percentage total that fell into each category.

TABLE III
My caption

Representation	Positive	Negative	Neutral	Total
Raw	122	38	840	1000
Percentage	12.20%	3.80%	84.00%	100%

VI. Discussion

SentiStrength and Stanford CoreNLP rated close to the same number of commits as positive, differing by just 324 commits–or 0.78% of the dataset. The number of commits these two algorithms rated as negative and neutral diverged, differing by 11,581 and 12,224 commits, respectively. Overall, these two algorithms differed in classifying 18,805 commit messages–or 45.49% of the data set.

SentiCR rated just 90 commits as positive, and 189 commits as negative; the remaining 41,059 commits, this algorithm rated as neutral. Due to the preponderance of "Neutral" commit messages, as categorized by SentiCR, its rating matched SentiStrength for 32,084 commits; SentiCR agreed with Stanford CoreNLP in its categorization of 19,938 commits.

The gold standard, human categorization of the randomly selected dataset yielded results that–although close in some areas to those of the sentiment analysis algorithms–did not match them. This divergence, slight in some areas and significant in others, stems from these algorithms' lack of domain-specific training.

Consider Stanford CoreNLP's relatively extreme categorization of almost half the dataset as negative. According to the gold standard, less than ten percent of those commits actually deserved this categorization, yet Stanford CoreNLP classified almost 17,500 commits as negative. This is due to the fact that these algorithms in general place an inordinate amount of emphasis on words that–in the software engineering domain–do not carry the same emotional connotations that they might in movie or product reviews, for which these algorithms were created and on which they were trained. For example, Stanford CoreNLP considers "error", "remove", "compensate", and "change" negative when analyzing sentences, when in reality sentences like "Changed embedded tomcat to full tomcat." and "Renamed BeanFactory-¿BeanManager."–both of which Stanford CoreNLP categorized as negative–are not.

The same holds true for SentiStrength and its over-categorization of negative commits, and its under-categorization of positive commits: this algorithm, too, lacks the domain-specific knowledge to make accurate assessments of the sentiment present in developer commits.

SentiCR, on the other hand, erred too far in the opposite direction in its categorization of the dataset. While SentiStrength and Stanford CoreNLP both vastly over-categorized the number of negative commit messages and under-categorized the number of positive commit messages in the data set, SentiCR labeled almost the entire data set neutral.

Below, in figures five through eight, are several specific examples of developer commit messages and their categorization by each algorithm. The first example, in Figure 5, shows a commit message in which none of the sentiment analysis algorithms agreed. As is clear through observation, however, the correct sentiment present in this message is neutral.

```
"Returned back to FlowInput / FlowOutput
    ↪ approach, reinstated support."
 -- CoreNLP rates this as Negative,
     ↪ SentiStrength rates this as Positive,
     ↪ SentiCR rates this as Neutral.
```

Fig. 5. Example conflicting commit message

Figure 6 contains a commit message all three algorithms categorized as positive, and rightly so.

```
"Added a cool touchgraph view. Need to spruce it
    ↪  up a bit though."
 -- CoreNLP rates this as Positive,
     ↪ SentiStrength rates this as Positive,
     ↪ SentiCR rates this as Positive.
```

Fig. 6. Example positive commit message

Figure 7, below, contains a commit message all three algorithms erroneously agree is negative, due to the presence of a single key word: "fail". Once again, upon observation and in the context of this domain, the correct categorization of this message is neutral.

```
"tests ... changed folder-structure ... some
    ↪ tests fail at the moment!"
 -- CoreNLP rates this as Negative,
     ↪ SentiStrength rates this as Negative,
     ↪ SentiCR rates this as Negative.
```

Fig. 7. Example commit categorization

Figure 8, below, contains a commit message all three algorithms correctly identify as neutral.

```
"Renamed old model."
 -- CoreNLP rates this as Neutral, SentiStrength
     ↪  rates this as Neutral, SentiCR rates
     ↪ this as Neutral.
```

Fig. 8. Example commit categorization

VII. Related Work

In *Product Review Summarization from a Deeper Perspective*, Ly et al. [1] of the National University of Singapore highlight the importance of sentiment analysis algorithms in providing valuable real-world intelligence to corporations concerning public perception of their products and services. They recognize, however, the increased difficulty in providing reliable analysis as the input length grows, and so their proposed system seeks to mitigate this by providing accurate analysis regardless of input length.

On Negative Results when Using Sentiment Analysis Tools for Software Engineering Research, by Jongeling et al. [2] discusses how training many of the most popular sentiment analysis algorithms on product and movie reviews makes them ill-suited for the software engineering domain. Their extensive analysis of several data sets as well as previously published work highlights the inadequacies of these tools for the task at hand.

VIII. Conclusions & Further Work

While SentiStrength and Stanford CoreNLP produced almost the same results when it came to marking commit messages as positive, they diverged in terms of negative and neutral sentiment analysis. SentiCR also produced wildly divergent results when run across the dataset. It should be noted that due to SentiStrength and Stanford CoreNLP categorizing the majority of commit messages as neutral, and SentiCR doing the same, that although this may give the appearance of propriety, it is merely a fluke of the algorithms and not an intentional, data-based categorization and agreement. The inconsistencies across these three algorithms, and their lack of agreement with the gold standard categorization, makes it clear that they are not suited for this domain; thus, a new tool is necessary for proper sentiment analysis of developer commits, and as of this writing, SentiCR is not that tool.

References

[1] D. K. Ly, K. Sugiyama, Z. Lin, and M.-Y. Kan, "Product review summarization from a deeper perspective," in *Proceedings of the 11th Annual International ACM/IEEE Joint Conference on Digital Libraries*, ser. JCDL '11. New York, NY, USA: ACM, 2011, pp. 311–314. [Online]. Available: http://doi.acm.org/10.1145/1998076.1998134

[2] R. Jongeling, P. Sarkar, S. Datta, and A. Serebrenik, "On negative results when using sentiment analysis tools for software engineering research," *Empirical Softw. Engg.*, vol. 22, no. 5, pp. 2543–2584, Oct. 2017. [Online]. Available: https://doi.org/10.1007/s10664-016-9493-x