SentiCR: A Customized Sentiment Analysis Tool for Code Review Interactions

Toufique Ahmed†, Amiangshu Bosu‡, Anindya Iqbal†, Shahram Rahimi‡
Department of Computer Science & Engineering, Bangladesh University of Engineering and Technology, Dhaka, Bangladesh†
Department of Computer Science, Southern Illinois University Carbondale, IL, USA‡
{toufiqueahmed, anindya}@cse.buet.ac.bd†, {abosu, rahimi}@cs.siu.edu‡

***Abstract*—Sentiment Analysis tools, developed for analyzing social media text or product reviews, work poorly on a Software Engineering (SE) dataset. Since prior studies have found developers expressing sentiments during various SE activities, there is a need for a customized sentiment analysis tool for the SE domain. On this goal, we manually labeled 2000 review comments to build a training dataset and used our dataset to evaluate seven popular sentiment analysis tools. The poor performances of the existing sentiment analysis tools motivated us to build SentiCR, a sentiment analysis tool especially designed for code review comments. We evaluated SentiCR using one hundred 10-fold cross-validations of eight supervised learning algorithms. We found a model, trained using the Gradient Boosting Tree (GBT) algorithm, providing the highest mean accuracy (83%), the highest mean precision (67.8%), and the highest mean recall (58.4%) in identifying negative review comments.**

I. Introduction

A person's sentiment (i.e., positive or negative attitude) towards another person, entity, or event significantly influences his/her decision-making process such as forming relationships, choosing candidates in a local election, selecting commercial products, reviewing movies, or predicting financial condition of a stock market [24]. Sentiments not only influence the quality of relationship between two persons but also have high impacts on productivity, task quality, task synchronization, and job satisfaction of collaborative activities [8] such as software development [14]. Due to the limited availability of face-to-face communications, OSS developers primarily use various text-based tools such as mailing lists, forums, source code repositories, code reviews, and issue tracking tools to manage their collaborations [43]. Prior Software Engineering (SE) studies found developers expressing sentiments in commit messages [14], issue tracking systems [19], [28], project artifacts [28], and mailing-lists [15], [48]. Yet, the lack of a reliable sentiment analysis (SA) tool [20], [48] for the SE domain has hindered evaluating the impacts of those sentiments.

Most of the prior research on SA techniques have focused on analyzing social media posts, product reviews or movie reviews. Although existing SA tools work well on social media posts or product reviews, those perform poorly on a SE dataset [20], [48]. The text of SE communications often differ from articles, books, or even spoken language as those often include technical jargons, word contractions, emoticons, URLs, and code snippets. Since SA tools need to be customized for each domain [23], the poor performances of existing SA tools on a SE dataset is not surprising.

To validate the need for a customized SA tool for the SE domain, we built a sentiment oracle by manually labeling 2000 randomly selected review comments from 20 popular OSS projects. Using our oracle, we evaluated the performances of seven commonly used SA tools. The poor performances of existing SA tools on our dataset motivated us to implement SentiCR, a supervised learning based SA tool for code reviews. We evaluated eight commonly used supervised learning algorithms based on hundred 10-fold cross-validations. We found a model, trained using the Gradient Boosting Tree (GBT) [37] algorithm, providing the highest mean accuracy (83%), the highest mean precision (67.8%), and the highest mean recall (58.4%) in identifying review comments expressing negative sentiments.

In summary, the primary contributions of this study are:

- An empirically built sentiment oracle for the SE domain.
- SentiCR: A supervised sentiment analysis tool for code review comments. Both SentiCR and our sentiment oracle are publicly available at:

  https://github.com/senticr/SentiCR/
- A comparative analysis of existing sentiment analysis tools on a SE dataset.

The remainder of the paper is organized as follows. Section II provides background about code review and sentiment analysis. Section III evaluates existing sentiment analysis tools. Section V describes the threats to validity of our findings. Finally, Section VI provides some directions for future work and concludes the paper.

II. Background

This section presents a brief background on two topics relevant to this study: peer code review and sentiment analysis.

A. Code Review

Code review is the practice where a developer submits his/her code change to a peer to judge its eligibility to be included into the main project code-base [6]. Compared with the traditional heavy-weight inspection process, peer code review is more informal, tool-based, and used regularly in practice [2]. Because the peer code review process has been established to be effective, many OSS projects have adopted it into their development process [2].

B. Sentiment Analysis

Sentiment Analysis is a natural language processing technique that analyzes the attitude of a speaker or an author of a body of text towards entities such as products, services, organizations, individuals, issues, or events [33]. Sentiment analysis techniques aim to identify polarity (i.e., positive, negative, or neutral) in a sentence or a paragraph.

Researchers have primarily used two types of sentiment analysis techniques. Supervised learning based techniques, which are able to adapt and create trained models for specific purposes and contexts, can be used in conjunction with any of the existing supervised learning methods (e.g., Naïve Bayes, and SVM) [31]. Instead of using standard supervised techniques, researchers have also proposed several custom techniques specifically for sentiment classification tasks that can take into account the contexts of words expressing sentiments [24]. However, supervised learning techniques require a labeled training dataset, which might be costly or even prohibitive.

On the other hand, lexicon-based analyzers, which do not require a training dataset, identify the sentiment for a document from the semantic orientations of words or phrases in the document [49]. Although lexical methods do not rely on a labeled dataset, it is hard to create a unique lexicon-based dictionary suitable for different contexts and often require customizations of the dictionary for each domain [44].

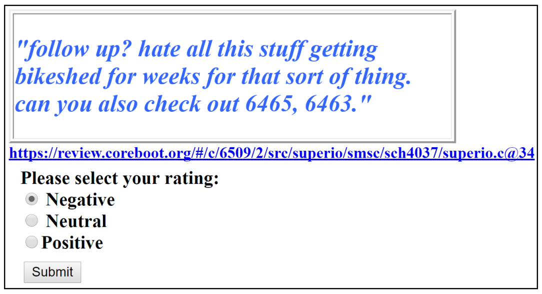
III. Evaluation of Sentiment Analysis Tools

Most of the prior SE studies used either SentiStrength [46] (a lexicon-based analyzer) [14], [15], [38] or NLTK [5] (a supervised learning based classifier) trained on movie reviews [12], [40]. However, a recent study [20] observed not only poor accuracies but also significant disagreements between these two tools; which could potentially lead to contradictory conclusions. To use in our study, an evaluation of the existing tools on a SE dataset is necessary. Since no labeled sentiment dataset exists in the SE domain, we built a sentiment dataset using randomly selected code review comments. Following sections describe our dataset generation process and an evaluation of seven popular sentiment analysis tools.

A. Training Dataset Generation

We adopted following eight-step approach to build a labeled sentiment dataset from code review comments.

1) We used our Gerrit-Miner [6] tool to mine the code review repositories of 20 popular OSS projects.Our project selection was based on two criteria: i) an open source project actively using Gerrit; and ii) each project has performed at least 10,000 code reviews.
2) A manual inspection of the comments posted by some accounts (e.g., 'Qt Sanity Bot' or 'BuildBot') suggested that those accounts were automated bots rather than humans. These accounts typically contain one of the following keywords: 'bot', 'auto', 'CI', 'Jenkins', 'integration', 'build', 'hook', 'recheck', 'travis', or 'verifier'. Because we wanted only code review comments from actual reviewers, we excluded these bot accounts after a manual inspection had confirmed that the comments were automatically generated.
3) We randomly selected total 2000 review comments each having at least 50 characters from the selected twenty projects (100 comments from each project). The randomness of our selection process ensures that all types of review comments are included in our dataset. We did not adopt a proportionate selection as some of the large projects (e.g., Android, Chromium OS, and OpenStack) had significantly higher number of code reviews than the others. A proportionate selection would have biased our dataset in favor of the vocabulary of those large projects.
4) We developed a web-app (Figure 1) to manually label the selected review comments. Three of the authors independently labeled each of the review comment as 'positive', 'negative' or 'neutral' based on what s/he would personally perceive if s/he was the recipient of a review comment. To eliminate potential biases, the raters could not view the ratings provided by others.
5) The three raters had consensus on 1,239 comments (62.5%) in their initial independent ratings. We used Fleiss' Kappa [11] (useful for more than two raters) to measure the level of agreement among the three raters. $\kappa = 0.408$ ($p < 0.001$), indicates a "moderate agreement" [21] for our initial ratings. The perception of sentiment from a text varies across different persons [32]. While it was possible to achieve higher consensus among our initial ratings with a prior discussed rating strategy, such a rubrics would have prevented us from capturing a real-world sentiment perception.
6) For the 771 comments, where one of the raters disagreed, we had discussion sessions to determine the final ratings.
7) The final distribution of the labeled comments were: 7.7% ('positive'), 19.9% ('negative'), and 72.4% ('neutral'). To improve the performances of sentiment classifiers, we converted our three-class dataset into a two-class dataset by relabeling both 'positive' and 'neutral' comments as 'non-negative'.


"follow up? hate all this stuff getting bikeshed for weeks for that sort of thing. can you also check out 6465, 6463."

https://review.coreboot.org/#/c/6509/2/src/superio/smsc/sch4037/superio.c@34

Please select your rating:

[✓ Negative] Neutral Positive

Fig. 1: Web app to manually label the review comments

TABLE I: Performances of seven sentiment analysis tools

	Precision	Recall	F-measure	Accuracy
Afinn [30]	35.9%	36.2%	0.36	68.6%
NLTK (Hu and Liu) [16]	41.6%	35.7%	0.38	72.1%
SentiStrength [46]	37.5%	37.9%	0.38	69.4%
TextBlog [26]	45.5%	40.7%	0.43	73.6%
USent [35]	47.1%	30.4%	0.37	74.7%
NLTK Vader [17]	51.7%	14.8%	0.23	75.8%
Vivekn [29]	33.2%	84.7%	0.47	43.5%

8) Since the labeled dataset is highly imbalanced, we used undersampling (i.e., randomly excluding a subset of the majority class) [9], [25] to exclude 400 neutral comments. We use this dataset of 1,600 labeled comments as an 'oracle' for all the subsequent analyses in this study. Our dataset satisfies Thewall's recommendation of minimum 1000 labeled text for sentiment training [45].

B. Evaluation Results

We evaluated seven popular sentiment analysis tools (five lexicon-based [16], [17], [30], [35], [46] and two supervised learning based [26], [29]) using our oracle. We measure the performances of the tools based on four measures: precision, recall, f-measure, and accuracy. Table I shows the performances of the seven tools. Each of the tools performed poorly in identifying negative review comments with the highest precision of only 52% (NLTK Vader). Although, Vivekn had the highest recall (84.7%), it had the lowest precision(33.2%). None of the tools achieved a f-measure above 0.5, validating the need for a customized sentiment analysis tool for code review comments. Since these tools are not trained using a SE dataset, their inaccuracies are not surprising. A reliable sentiment analyzer for the SE domain requires either a supervised model trained on a SE dataset or a customized lexicon-based SE dictionary.

IV. SentiCR: A Sentiment analysis tool for Code Reviews

The poor performances of existing sentiment analysis tools motivated us to implement *SentiCR*, a supervised learning based sentiment analysis tool for code review comments. We wrote SentiCR in Python using NLTK [5] for language preprocessing and scikit-learn [36] for supervised learning algorithms. In the following subsections, we describe the data preprocessing steps and evaluation of SentiCR.

A. Data Preprocessing

The text of a review comment differs from articles, books, or even spoken language. For example, review comments often contain word contractions, emoticons, URLs, and code snippets. Therefore, we implemented an eight step data preprocessing as follows.

1) Expansion of contractions: Contractions, which are shortened form of one or two words, are widely used in informal written communications. Some commonly used contractions and their expanded forms include: *I'm* $\rightarrow$ *I am*, *doesn't* $\rightarrow$ *does not*, and *don't* $\rightarrow$ *do not*. By creating two different lexicons of the same term, contractions increase the number of unique lexicons and misrepresent the real characteristics of a dataset. We replaced the commonly used 124 contractions, each with its expanded version.

2) URL removal: A review comment may include URL references (e.g., StackOverflow) supporting the reviewer's suggestion. While URLs do not contribute to sentiments, those can increase the number of features for supervised classifiers. We used a regular expression matcher to identify and remove all URLs from our dataset.

3) Special handling of emoticons: Emoticons are widely used in informal written communications and are very influential in expressing sentiments. Similar to Agarwal *et al.* [1], we replaced each of the emoticons with a word representing its sentiment polarity by looking up an emoticon dictionary. For example, both the happy face emoticon - ':)' and laughing emoticon - ':D' were replaced with the word '*PositiveEmoticon*'. Similarly, both crying - ':(' and annoyed - ':s' were replaced with the word '*NegativeEmoticon*'.

4) Negation pre-processing: Similar to prior studies, we also observed higher misclassifications for comments including negation words (e.g., 'not', 'never', 'neither', 'nothing', and 'nobody'). Since supervised learning based classifiers operate on bag-of-words representations, those classifiers often fail to identify negated opinions. For example, *'I like your changes'* expresses a positive sentiment but *'I do not like your changes'* expresses a negative sentiment. To differentiate words in negated contexts from the same words in non-negated contexts, prior supervised learning based sentiment analysis tools [34] adopted a simple approach of prepending *'not_'* with the succeeding words. For example, '*I do not like your changes*' will become '*I do not not_like not_your not_changes*'. However, for a complex sentence such as the following review comment in our dataset, this approach will overestimate the scope of negation and inaccurately negate additional words.

"Generally dropping these tables from the build is almost never worth it, since most OSes expect something there."

Moreover, negations usually impact verbs, adjectives, and adverbs but do not alter nouns, determiners, articles, adpositions, and particles [10]. We use NLTK universal parts-of-speech (POS) tagger to attach POS tags with each word. Then, we use chunking, a process of dividing a text into syntactically correlated parts of words (also know as 'shallow parsing') [47] to precisely determine the scope of negations. We developed following chunk grammar (i.e., rules indicating how sentences should be chunked) for `NLTK RegexpChunkParser` to identify negation phrases.

$$
\begin{array}{lll}
\langle NegP \rangle & ::= & \{\langle VERB\rangle?\langle ADV\rangle+\{\langle VERB\rangle|\langle ADJ\rangle\}? \\
& & \{\langle PRT\rangle|\langle ADV\rangle\langle VERB\rangle\}\} \\
& | & \{\langle VERB\rangle?\langle ADV\rangle+\{\langle VERB\rangle|\langle ADJ\rangle\}* \\
& & \{\langle ADP\rangle|\langle DET\rangle\}?\langle ADJ\rangle?\langle NOUN\rangle?\langle ADV\rangle?\}
\end{array}
$$

Our chunk grammar had 96.5% recall in identifying negation phrases on our oracle. Since we had a second stage filtering (Algorithm 1-line 9), we consider only recall to

TABLE II: Words removed from feature vector

Stopwords	Programming keywords
i, me, my, myself, we, our, ourselves, you, your, yourself, he, him, his, himself, she, her, herself, it, itself, they, them, their, themselves, this, that, these, those, am, is, are, was, were, be, been, being, have, has, had, having, do, does, did, doing, a, an, the, and, if, or, as, until, while, of, at, by, for, between, into, through, during, to, from, in, out, on, off, then, once, here, there, all, any, both, each, few, more, other, some, such, than, too, very, s, t, can, will, don, should, now	abstract, and, assert, bool, boolean, break, byte, case, catch, char, class, clone, const, continue, def, default, delegate, delete, do, double, each, echo, elif, else, elseif, endfor, endforeach, endif, endwhile, enum, event, except, explicit, export, extends, fi, final, finally, float, for, foreach, function, get, global, goto, if, implements, import, in, include, instanceof, int, interface, lambda, long, namespace, native, new, null, or, out, override, package, print, private, protected, public, raise, readonly, require, return, set, short, signed, static, struct, super, switch, synchronized, than, this, throw, throws, try, union, var, virtual, void, with, yield

measure the performance of our grammar. We modified all the verbs, adjectives, and adverbs in a negation phrase as identified by the chunker by prepending *'not_'*. Algorithm 1 describes our negation preprocessing.

Algorithm 1 Negation pre-processing

```
1:  procedure HANDLE_NEGATION($text$)          ▷ Input: A code review comment.
2:      $sentences \leftarrow sentence\_tokenize(text)$      ▷ Tokenize the sentences

3:      for all sent in sentences do
4:          $words \leftarrow word\_tokenize(sent)$      ▷ Tokenize sentence into words
5:          if words contain negation_words then
6:              $tagged\_words \leftarrow pos\_tag(words)$
7:              $chunk\_phrases \leftarrow regex\_chunk\_parse(tagged\_words)$
8:              for all phrase in chunk_phrases do
9:                  if phrase contain negation_words then
10:                     for all word in phrase do
11:                         if pos(word) in [ADJ, ADV, VERB] then
12:                             ▷ Modify the original word by prepending 'not_'
13:                             $word \leftarrow prepend\_not(word)$
14:                         end if
15:                     end for
16:                 end if
17:             end for
18:         end if
19:     end for
        return $modified\_text$
20: end procedure
```

5) Word stemming: We used NLTK word tokenizer to parse each text into a list of words. Next, we applied Snowball Stemmer [39] to convert each word to its stem.

6) Stop-word removal: Many stopwords (usually non-semantic words such as articles, prepositions, conjunctions, and pronouns) do not play significant roles to express sentiments. Popular natural language processing tools, such as NLTK [5] and Stanford CoreNLP [27] provide lists of stopwords. However, some of the words (e.g., 'no', 'not', 'why', and 'what') from those lists are influential in the particular context of expressing sentiments. Hence, we used a customized stopword list (Table II- column stopwords) and removed words belonging to that list from the review comments.

7) Code snippet removal: Code review comments often include improvement suggestions as code snippets. Since code snippets do not express sentiments, SeintCR removes those. Whereas we could have used an approach similar to Bacchelli *et al.* [3] to separate code snippets from comments, we observed that a simpler solution worked in our case. Code snippets primarily include keywords, identifiers, and punctuation characters. We prepared a list of keywords (Table II-second column) from the popular programming languages (e.g., Java, C, C++, Python, JavaScript, and PHP) and added those to our stopword list for removal. As popular open source projects use descriptive variable names, we did not observe same identifier present in more than two comments in our oracle. Therefore, the exclusion of words present in less than three comments removes all the identifiers. We did not require any preprocessing for punctuation marks as the tokenization step discards those.

8) Feature vector generation: Similar to prior sentiment analysis studies [18], [22], we computed TF-IDF (Term Frequency - Inverse Document Frequency) to extract the features for classification. In this step, we excluded the words that are present in less than three code review comments in our oracle. We use this feature vector to train our classifiers.

B. Evaluation of SentiCR

Using the feature vector generated after the data preprocessing steps, we evaluated following eight supervised algorithms, which are commonly used for sentiment analysis.

1) Adaptive Boosting (ADB) [42],
2) Decision Tree (DT) [4],
3) Gradient Boosting Tree (GBT) [37],
4) Naïve Bayes (NB) [31],
5) Random Forest (RF) [13],
6) Multilayer Perceptron (MLPC) [41],
7) Support Vector Machine with Stochastic Gradient Descent (SGD) [4], and
8) Linear Support Vector Machine (SVC) [31].

We have used the Scikit-learn [36] implementations of those algorithms. We validated each of the algorithms using 10-fold cross-validations, where the dataset was randomly divided into 10 groups and each of the ten groups was used as test dataset once, while the remaining nine groups were used to train the classifier. Since our training dataset is imbalanced (i.e., almost 75% comments were 'non-negative' in the oracle of 1600 comments), we observed higher classification errors for the negative comments. We used SMOTE (synthetic minority over-sampling technique) [7] to improve the ratio of 'negative' to 'non-negative' comments to *0.5* in the training samples. We repeated this process over hundred times and computed the mean performances of the classifiers. Following subsections discuss the two questions to evaluate SentiCR.

1) Which supervised learning algorithm provides the best performance on our oracle?: Both Table III and Figure 2 shows the performances (i.e., precision, recall, accuracy, and f-measure) of the eight supervised learning algorithms based on one hundred 10-fold cross-validations. The results suggest that the traditional methods (i.e., Naïve Bayes and DT) did not perform well on our oracle. GBT performed the best among the supervised ensemble learning methods (i.e., AdaBoost, GBT, and RandomForest).The boxplots in Figure 2 shows the variations in performances over 100 runs, where 'X' denotes

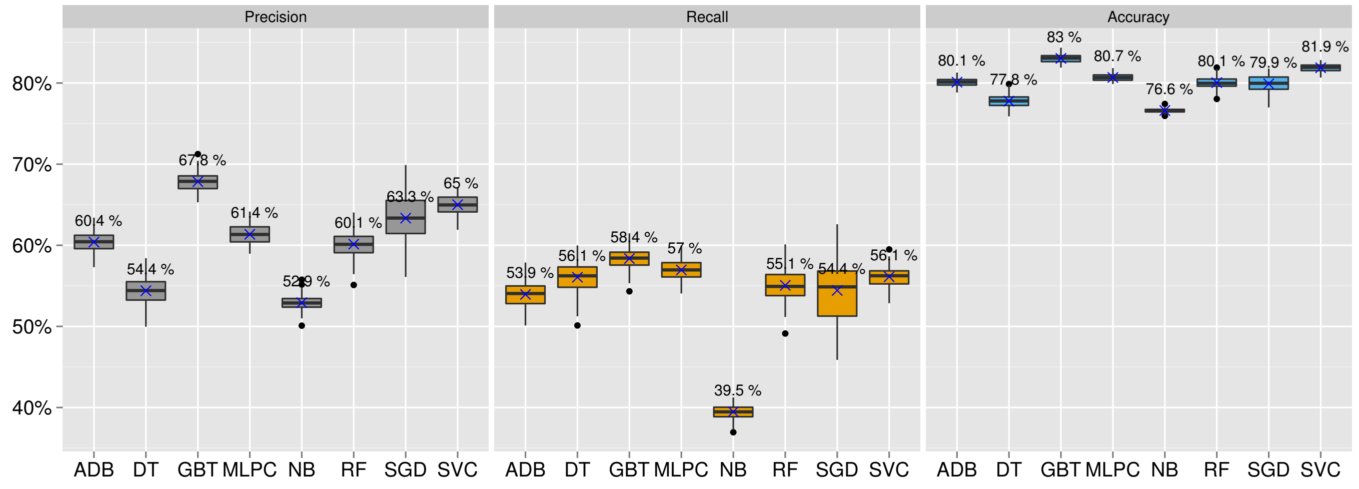


Fig. 2: Performances of the eight supervised algorithms during one hundred ten-fold cross-validations

TABLE III: Performance of SentiCR on eight supervised learning algorithms with and without negation preprocessing

Algo.	**With negation preprocessing**				**Without negation preprocessing**			
	P	*R*	*F*	Accuracy	*P*	*R*	*F*	Accuracy
ADB	60.39%	53.94%	0.57	80.10%	59.03%	52.77%	0.55	79.50%
DT	54.39%	56.06%	0.55	77.70%	51.94%	54.20%	0.53	76.52%
GBT	**67.84%**	**58.35%**	**0.62**	**83.03%**	66.17%	56.19%	0.60	82.27%
MLPC	61.35%	56.96%	0.59	80.69%	57.32%	54.12%	0.55	78.90%
NB	52.92%	39.48%	0.45	76.61%	47.37%	33.65%	0.39	74.66%
RF	60.14%	55.06%	0.57	80.05%	57.76%	47.80%	0.52	78.69%
SGD	63.34%	54.43%	0.56	79.90%	59.05%	52.45%	0.53	78.16%
SVC	65.01%	56.12%	0.60	81.88%	59.76%	52.71%	0.56	79.77%

the mean. The narrow boxplots of GBT also indicates a stable performance over the one hundred runs.

Prior research on natural language processing indicates that even human raters can achieve at most 80% accuracies in predicting sentiments [32]. Since five out of the eight models achieve human level accuracy (as defined by Thelwall [46]).

2) Does our negation preprocessing improve the performances of the supervised models?: We repeated our hundred 10-fold cross validations without negation preprocessing. Table III includes the performances of each of the eight algorithms with and without negation preprocessing. The results suggest that each of the eight models had improved their precisions between 1% to 5%, with SVC and NB models gaining the most. Similarly, each of the eight models also had improved recalls, improved accuracies, and improved f-measures. Therefore, the negation preprocessing steps are useful in improving the performances of our classifiers.

V. Threats to Validity

We included 20 publicly accessible OSS projects that practice tool-based code reviews supported by the same tool (i.e., Gerrit). Though, it is possible that projects supported by other code review tools (e.g., ReviewBoard, Github pull-based reviews, and Phabricator) could have behaved differently, sentiments expressed in review comments may not depend on any feature that is exclusive to Gerrit only.

Overfitting is a potential threat for any supervised learning based model. To combat overfitting, we employed hundred ten-fold cross validations of our models. We included only those words that are present in at least three comments. To further validate our classifier, we randomly selected 200 review comments that the model had not seen before and manually classified them. Against our manual classification, the model had 86% accuracy, suggesting no performance degradation. To extend this analysis, we manually rated 300 more review comments from 5 professional developers (60 each). For this cross-validation, SentiCR was 82% accurate (consistent with our cross-validation results). Since we adopted several measures to mitigate overfitting, we do believe that SentCRs accuracy should not degrade on a large dataset.

VI. Future Works and Conclusion

In this study, we empirically built and validated a sentiment oracle from 2000 randomly selected code review comments. We have implemented SentiCR, a sentiment analysis tool for the SE domain. In the future, We plan to use SentiCR to find out the impact of negative review comments on project outcomes.

References

[1] A. Agarwal, B. Xie, I. Vovsha, O. Rambow, and R. Passonneau, "Sentiment analysis of twitter data," in *Proceedings of the workshop on languages in social media*. Association for Computational Linguistics, 2011, pp. 30–38.

[2] A. Bacchelli and C. Bird, "Expectations, outcomes, and challenges of modern code review," in *Proceedings of the 2013 International Conference on Software Engineering*. San Francisco, CA, USA: IEEE Press, 2013, pp. 712–721.

[3] A. Bacchelli, M. Lanza, and R. Robbes, "Linking e-mails and source code artifacts," in *Proceedings of the 32Nd ACM/IEEE International Conference on Software Engineering - Volume 1*, ser. ICSE '10, 2010.

[4] A. Bifet and E. Frank, "Sentiment knowledge discovery in twitter streaming data," in *International Conference on Discovery Science*. Springer, 2010, pp. 1–15.

[5] S. Bird, "NLTK: The Natural Language Toolkit," in *Proceedings of the COLING/ACL on Interactive presentation sessions*. Association for Computational Linguistics, 2006, pp. 69–72.

[6] A. Bosu and J. C. Carver, "Impact of peer code review on peer impression formation: A survey," in *2013 ACM/IEEE International Symposium on Empirical Software Engineering and Measurement*. IEEE, 2013, pp. 133–142.

[7] N. V. Chawla, K. W. Bowyer, L. O. Hall, and W. P. Kegelmeyer, "Smote: synthetic minority over-sampling technique," *Journal of artificial intelligence research*, vol. 16, pp. 321–357, 2002.

[8] M. De Choudhury and S. Counts, "Understanding affect in the workplace via social media," in *Proceedings of the 2013 conference on Computer supported cooperative work*. ACM, 2013, pp. 303–316.

[9] C. Drummond, R. C. Holte *et al.*, "C4. 5, class imbalance, and cost sensitivity: why under-sampling beats over-sampling," in *Workshop on learning from imbalanced datasets II*, vol. 11. Citeseer, 2003.

[10] X. Fang and J. Zhan, "Sentiment analysis using product review data," *Journal of Big Data*, vol. 2, no. 1, p. 5, 2015.

[11] J. L. Fleiss, "Measuring nominal scale agreement among many raters." *Psychological bulletin*, vol. 76, no. 5, p. 378, 1971.

[12] D. Garcia, M. S. Zanetti, and F. Schweitzer, "The role of emotions in contributors activity: A case study on the Gentoo community," in *Cloud and Green Computing (CGC), 2013 Third International Conference on*. IEEE, 2013, pp. 410–417.

[13] A. Gupte, S. Joshi, P. Gadgul, A. Kadam, and A. Gupte, "Comparative study of classification algorithms used in sentiment analysis," *International Journal of Computer Science and Information Technologies*, vol. 5, no. 5, pp. 6261–6264, 2014.

[14] E. Guzman, D. Azócar, and Y. Li, "Sentiment analysis of commit comments in github: an empirical study," in *Proceedings of the 11th Working Conference on Mining Software Repositories*. ACM, 2014, pp. 352–355.

[15] E. Guzman and B. Bruegge, "Towards emotional awareness in software development teams," in *Proceedings of the 2013 9th Joint Meeting on Foundations of Software Engineering*, ser. ESEC/FSE 2013, 2013, pp. 671–674.

[16] M. Hu and B. Liu, "Mining and summarizing customer reviews," in *Proceedings of the tenth ACM SIGKDD international conference on Knowledge discovery and data mining*. ACM, 2004, pp. 168–177.

[17] C. J. Hutto and E. Gilbert, "Vader: A parsimonious rule-based model for sentiment analysis of social media text," in *Eighth International AAAI Conference on Weblogs and Social Media*, 2014.

[18] D. Isa, L. H. Lee, V. Kallimani, and R. Rajkumar, "Text document preprocessing with the bayes formula for classification using the support vector machine," *IEEE Transactions on Knowledge and Data engineering*, vol. 20, no. 9, pp. 1264–1272, 2008.

[19] M. R. Islam and M. F. Zibran, "Leveraging automated sentiment analysis in software engineering," in *Proceedings of the 14th International Conference on Mining Software Repositories*. IEEE Press, 2017, pp. 203–214.

[20] R. Jongeling, S. Datta, and A. Serebrenik, "Choosing your weapons: On sentiment analysis tools for software engineering research," in *Software Maintenance and Evolution (ICSME), 2015 IEEE International Conference on*. IEEE, 2015, pp. 531–535.

[21] J. R. Landis and G. G. Koch, "The measurement of observer agreement for categorical data," *biometrics*, pp. 159–174, 1977.

[22] L. H. Lee, C. H. Wan, R. Rajkumar, and D. Isa, "An enhanced support vector machine classification framework by using euclidean distance function for text document categorization," *Applied Intelligence*, vol. 37, no. 1, pp. 80–99, 2012.

[23] B. Liu, "Sentiment analysis and opinion mining," *Synthesis lectures on human language technologies*, vol. 5, no. 1, pp. 1–167, 2012.

[24] B. Liu and L. Zhang, "A survey of opinion mining and sentiment analysis," in *Mining text data*. Springer, 2012, pp. 415–463.

[25] X.-Y. Liu, J. Wu, and Z.-H. Zhou, "Exploratory undersampling for class-imbalance learning," *IEEE Transactions on Systems, Man, and Cybernetics, Part B (Cybernetics)*, vol. 39, no. 2, pp. 539–550, 2009.

[26] S. Loria, "Textblob: simplified text processing," *Secondary TextBlob: Simplified Text Processing*, 2014.

[27] C. D. Manning, M. Surdeanu, J. Bauer, J. R. Finkel, S. Bethard, and D. McClosky, "The Stanford CoreNLP Natural Language Processing Toolkit." in *ACL (System Demonstrations)*, 2014, pp. 55–60.

[28] A. Murgia, P. Tourani, B. Adams, and M. Ortu, "Do developers feel emotions? an exploratory analysis of emotions in software artifacts," in *Proceedings of the 11th Working Conference on Mining Software Repositories*. ACM, 2014, pp. 262–271.

[29] V. Narayanan, I. Arora, and A. Bhatia, "Fast and accurate sentiment classification using an enhanced naive bayes model," in *International Conference on Intelligent Data Engineering and Automated Learning*. Springer, 2013, pp. 194–201.

[30] F. Å. Nielsen, "A new anew: Evaluation of a word list for sentiment analysis in microblogs," *arXiv preprint arXiv:1103.2903*, 2011.

[31] B. Pang and L. Lee, "A sentimental education: Sentiment analysis using subjectivity summarization based on minimum cuts," in *Proceedings of the 42nd annual meeting on Association for Computational Linguistics*. Association for Computational Linguistics, 2004, p. 271.

[32] ——, "Seeing stars: Exploiting class relationships for sentiment categorization with respect to rating scales," in *Proceedings of the 43rd annual meeting on association for computational linguistics*. Association for Computational Linguistics, 2005, pp. 115–124.

[33] ——, "Opinion mining and sentiment analysis," *Foundations and trends in information retrieval*, vol. 2, no. 1-2, pp. 1–135, 2008.

[34] B. Pang, L. Lee, and S. Vaithyanathan, "Thumbs up?: sentiment classification using machine learning techniques," in *Proceedings of the ACL-02 conference on Empirical methods in natural language processing-Volume 10*. Association for Computational Linguistics, 2002, pp. 79–86.

[35] N. Pappas, G. Katsimpras, and E. Stamatatos, "Distinguishing the popularity between topics: A system for up-to-date opinion retrieval and mining in the web," in *Computational Linguistics and Intelligent Text Processing*, 2013, vol. 7817, pp. 197–209.

[36] F. Pedregosa, G. Varoquaux, A. Gramfort, V. Michel, B. Thirion, O. Grisel, M. Blondel, P. Prettenhofer, R. Weiss, V. Dubourg, J. Vanderplas, A. Passos, D. Cournapeau, M. Brucher, M. Perrot, and E. Duchesnay, "Scikit-learn: Machine Learning in Python," *Journal of Machine Learning Research*, vol. 12, pp. 2825–2830, 2011.

[37] M. Pennacchiotti and A.-M. Popescu, "Democrats, republicans and starbucks afficionados: user classification in twitter," in *Proceedings of the 17th ACM SIGKDD international conference on Knowledge discovery and data mining*. ACM, 2011, pp. 430–438.

[38] D. Pletea, B. Vasilescu, and A. Serebrenik, "Security and emotion: sentiment analysis of security discussions on github," in *Proceedings of the 11th Working Conference on Mining Software Repositories*. ACM, 2014, pp. 348–351.

[39] M. F. Porter, "Snowball: A language for stemming algorithms," 2001.

[40] A.-I. Rousinopoulos, G. Robles, and J. M. González-Barahona, "Sentiment analysis of free/open source developers: Preliminary findings from a case study." *Revista Eletrônica de Sistemas de Informação*, vol. 13, no. 2, 2014.

[41] D. E. Rumelhart, G. E. Hinton, and R. J. Williams, "Learning representations by back-propagating errors," *Cognitive modeling*, vol. 5, no. 3, p. 1, 1988.

[42] N. F. F. d. Silva, E. R. Hruschka, E. R. Hruschka Junior *et al.*, "Biocom_usp: tweet sentiment analysis with adaptive boosting ensemble," in *International Workshop on Semantic Evaluation, 8th*. ACL Special Interest Group on the Lexicon-SIGLEX, 2014.

[43] M.-A. Storey, C. Treude, A. van Deursen, and L.-T. Cheng, "The impact of social media on software engineering practices and tools," in *Proceedings of the FSE/SDP workshop on Future of software engineering research*. ACM, 2010, pp. 359–364.

[44] M. Taboada, J. Brooke, M. Tofiloski, K. Voll, and M. Stede, "Lexicon-based methods for sentiment analysis," *Computational linguistics*, vol. 37, no. 2, pp. 267–307, 2011.

[45] M. Thelwall, "Heart and soul: Sentiment strength detection in the social web with sentistrength," *Proceedings of the CyberEmotions*, pp. 1–14, 2013.

[46] M. Thelwall, K. Buckley, G. Paltoglou, D. Cai, and A. Kappas, "Sentiment strength detection in short informal text," *Journal of the American Society for Information Science and Technology*, vol. 61, no. 12, pp. 2544–2558, 2010.

[47] E. F. Tjong Kim Sang and S. Buchholz, "Introduction to the conll-2000 shared task: Chunking," in *Proceedings of the 2nd workshop on Learning language in logic and the 4th conference on Computational natural language learning-Volume 7*. Association for Computational Linguistics, 2000, pp. 127–132.

[48] P. Tourani, Y. Jiang, and B. Adams, "Monitoring sentiment in open source mailing lists: exploratory study on the apache ecosystem," in *Proceedings of 24th annual international conference on computer science and software engineering*. IBM Corp., 2014, pp. 34–44.

[49] P. D. Turney, "Thumbs up or thumbs down?: semantic orientation applied to unsupervised classification of reviews," in *Proceedings of the 40th annual meeting on association for computational linguistics*. Association for Computational Linguistics, 2002, pp. 417–424.