Sentiment and Politeness Analysis Tools on Developer Discussions Are Unreliable, but so Are People

Nasif Imtiaz, Justin Middleton, Peter Girouard, and Emerson Murphy-Hill
North Carolina State University
Raleigh, North Carolina
{simtiaz,jamiddl2,plgiroua,ermurph3}@ncsu.edu

ABSTRACT

Many software engineering researchers use sentiment and politeness analysis tools to study the emotional environment within collaborative software development. However, papers that use these tools rarely establish their reliability. In this paper, we evaluate popular existing tools for sentiment and politeness detection over a dataset of 589 manually rated GitHub comments that represent developer discussions. We also develop a coding scheme on how to quantify politeness for conversational texts found on collaborative platforms. We find that not only do the tools have a low agreement with human ratings on sentiment and politeness, human raters also have a low agreement among themselves.

KEYWORDS

sentiment, politeness analysis, GitHub comments

ACM Reference Format:
Nasif Imtiaz, Justin Middleton, Peter Girouard, and Emerson Murphy-Hill . 2018. Sentiment and Politeness Analysis Tools on Developer Discussions Are Unreliable, but so Are People. In *Proceedings of ICSE (SEmotion 18).* ACM, New York, NY, USA, 8 pages. https://doi.org/10.1145/nnnnnnn.nnnnnnn

1 INTRODUCTION

A programmer's mood affects their activities and performance as programming involves various forms of cognitive tasks [22]. The collaborative nature of today's software development means that a developer's mood can be affected by other developers, and conversely, collaborating developers can be affected by one developer's mood [9, 17, 25]. Studies in other domains also show that organizational events can cause affective reactions which in turn influence performance and job satisfaction [32].

Therefore, software engineering researchers have increasingly studied emotion in software engineering in recent years [21]. Recent studies involve mining software artifacts, understanding signals of human emotions hidden in those artifacts, and then analyzing the signals in automated ways. The findings of these studies could be used by teams to track mood and formulate new strategies to ensure a healthy environment.

SEmotion 18, Workshop, E

ACM ISBN 978-x-xxxx-xxxx-x/YY/MM…$15.00
https://doi.org/10.1145/nnnnnnn.nnnnnnn

Researchers often use natural language processing tools to capture the emotion in a team. One place where teams express emotion is on social collaborative sites like GitHub, where developers communicate with each other to maintain their projects [38]. When developers use these sites, researchers can analyze the detailed history of project development, as well as developers' communication about the project in the form of issues, for example. Such textual artifacts present the opportunity to use natural language processing tools to do different affect analyses for different purposes [12, 13, 28]. While analyzing sentiment is the most commonly used technique to measure emotions, researchers have also used natural language tools to measure politeness [28], which is an important factor in the on-boarding process [37]. Thus, in this paper, we focus on sentiment analysis and politeness tools.

Despite the use of automated tools for analyzing sentiment and politeness, significant questions remain about the tools' reliability. Researchers who focus on sentiment analysis have provided strong data that the tools are less effective when applied to new domains they were not trained on [14, 27]. In our domain, Jongeling and colleagues have shown that different sentiment analysis tools yield different results when used on data from JIRA repository [21]. Hence, understanding the reliability of these tools will help software engineering researchers know whether they can use these tools to reach conclusions confidently.

In this paper, we study the reliability of popular sentiment analysis and politeness tools in the context of developer discussions. To do so, we randomly chose 589 comments from pull requests and issues on GitHub, manually annotated them for sentiment and politeness using human raters, and compared those annotations against the results produced by the tools.

The major contributions of this paper are:

(1) A benchmark of 589 GitHub comments, hand-rated by humans for sentiment and politeness,
(2) An annotation scheme that can be used to rate the politeness of comments in discussions, and
(3) A reliability evaluation of six sentiment analysis tools and one politeness tool.

The dataset, coding schemes and other associated files have been made publicly available at https://tinyurl.com/yabncd7b.

2 RELATED WORK

2.1 Sentiment Analysis in Software Engineering

Sentiment analysis focuses on the application of classifying texts as to their polarity (positive, negative or neutral) [31]. Using data from GitHub, Guzman and colleagues applied sentiment analysis to study

commit comments [18], while Pletea and colleagues tried to find a correlation between security-related discussions and fluctuating sentiments [33]. Other platforms, including the Gentoo community and Stack Overflow, have also been studied to understand the role of developers' sentiments in the process of software engineering [15, 19, 20, 26].

While most of these works have used the popular common tools, research on specialized tools for software engineering domain is ongoing. Customized tools try to overcome the problems of the prior common tools that were trained on texts from unrelated domains by having their own sentiment oracle for the software engineering domain [1, 6]. However, they have used different or no coding schemes while annotating the texts by human raters, which might lead them to have subtle differences in their understanding of sentiment in this domain.

2.2 Politeness Analysis in Software Engineering

Politeness can be described as "the practical application of good manners or etiquette" [7]. Politeness is a strong factor in social collaborations [29, 41]. Research is also going on in the software engineering domain to study the impact of politeness expressed by the developers. For example, Ortu and colleagues concluded that "the more polite developers were, the less time it took to fix an issue", while Tsay and colleagues studied GitHub discussions and found that, "the level of a submitter's prior interaction on a project changed how politely developers discussed the contribution and the nature of proposed alternative solutions" [29, 40]. However, similar to sentiment analysis, different studies had different methods to rate politeness [4, 40].

2.3 Challenges for Sentiment & Politeness Analysis in Software Engineering

Many researchers are planning to study the emotional awareness in collaborative software engineering [11]. However, challenges lie in correctly identifying any type of affect before we can use the data for further analysis. In a recent work, Murgia explores the possibilities of correctly judging emotions from texts extracted from software engineering platforms [25]. He found poor to moderate agreements between human raters on the different type of emotions. He concludes that "more investigation is needed before building a fully automatic emotion mining tool". Besides, Novielli and colleagues point towards the domain dependency of existing tools which makes it harder to apply them in new corpora [27].

Jongeling and his colleagues have shown how different tools lead to different results of sentiment in a new data set of JIRA issue reports [21]. They also show that "this disagreement [between different tools] can lead to diverging conclusions and that previously published results cannot be replicated when different sentiment analysis tools are used." Another work also used existing tools on code review comments from Gerrit and found a poor performance by the tools [1]. This establishes that reliability of a tool needs to be proven first, before the result of the tool can be used in any further analysis.

Just as Jongeling and colleagues evaluated the reliability of four sentiment analysis tools, we also perform a similar evaluation over a new dataset of GitHub comments with addition of tools that are specifically built for the software engineering domain. Moreover, we also extend our evaluation towards the reliability of politeness analysis on GitHub comments.

3 METHODOLOGY

Our goal is to evaluate the reliability of sentiment and politeness analysis tools in developer discussions by examining the tools' performance over GitHub comments. These discussions between developers can be thought of as the gateway to understand the emotional environment of a project, and how developers get along with each other. Before we evaluate the reliability of tools for doing affect analysis, we first need to define our gold standard for evaluation. Like other works [1, 6], we use human raters to create this gold standard. However, before taking human ratings at face value, we first ask to what extent human raters agree with each other:

- **RQ1.** How consistent are *human coders* to rate sentiment and politeness on GitHub comments?

Our next research questions evaluate the tools:

- **RQ2.** How reliable are *sentiment analysis tools* on GitHub comments?
- **RQ3.** How reliable are *politeness analysis tools* on GitHub comments?

3.1 Data Collection & Manual Raters

We chose GitHub as our research setting because it is the largest code host in the world [16]. The GitHub documentation designates issue and pull request sections as the appropriate place for both general and specific project discussion between developers[1]. Hence, we chose comments from these discussion threads to evaluate our tools on.

To get the comments, we use the GHTorrent project[2]. However, while GHTorrent does archive code-review comments, it does not contain the general discussion comments under issues and pull requests. Furthermore, GitHub comments may contain code snippets. To remove those code snippets, we need to be able to detect the HTML tags around the code, assuming that the authors highlighted them. For these purposes, we augment our data by mining GitHub's web pages to acquire the comments from the issue and pull request review sections. This way, we can also get the HTML tags.

As manual annotation is a time-consuming task, we estimated every hundred comment would take one hour to complete by one person. For each comment, we would have two human coders to manually go through and annotate, as a previous study found that "having more than two raters does not change the agreement significantly" [25]. We made an estimation that annotation of 600 comments can be completed within reasonable time and effort. Adding 40 comments more as a cautionary step, we randomly picked 640 comments from all the public projects from the data we collected. To estimate how much these comments represent developer discussions, We randomly picked 20 comments out of these 640 and manually investigated their origins. All the 20 comments came from 20 different projects and the projects represent valid software

[1] https://guides.github.com/features/issues/
[2] http://ghtorrent.org/

development. 18 out of these 20 projects have a clear description of the project in their README file. Out of these 20 projects, 4 projects had only 1 contributor, 3 projects had 2 contributors, and 1 project had 4 contributors. Rest of the 12 projects had more than 15 contributors.

During pre-inspection, we had to remove 51 comments as they were either incomplete fragments, duplicates, code snippets without commentary, non-English statements, or parts of discussions that are no longer available (due to, for example, invalid URLs). The rationale behind the last factor is that if the coders want to see the whole discussion of which a comment belongs to in order to have an understanding of the underlying context, they wouldn't be able to do so as the link is broken. We provide the final 589 comments to the human coders removing only the HTML tags. However, before feeding these comments to the tools, we also remove the code snippets and URLs, as they would get incorrectly processed by the tool wrongly considering them as English texts and hence biasing the result. The raters were also provided with the URL of the webpage containing the comment. Also, while we kept emoticons in the comments, we had to strip out the emojis before providing the comments to the human coders and the tool. The emojis in the GitHub comments are stored images that get fetched from a central hosting location. While we could replace the emojis by their titles using the images' HTML tags, the tools would have been unable to detect them as they are not trained in this way. So we removed all the emojis from the comments in our test data.

While the first coder (Coder 1) rated all the comments, we had two other coders who went through half of the data set each individually. All the coders were graduate and undergraduate students of the Computer Science department. We provided a coding guideline to the raters before providing the comments, explained in section 3.2, 3.3. The human coders had a short practice set consisting of 10 sample comments before starting with the main data set.

After the first iteration, where everybody separately annotated the texts, Coder 1 sat with each of the other coders to discuss the comments they disagreed on and their rationale behind their rating. This resulted in reaching a conclusion for the initially disagreed comments and bringing out insights over our coding guidelines.

3.2 Sentiment Annotation Scheme

As mentioned in section 2.1, previous works have used different or no coding schemes while doing sentiment rating in the software engineering domain. To not deviate a lot from the previous works, but still giving our coders a basic set of strategies, *we make use of the simple annotation scheme from a prior work developed by Mohammad* [24]. Based on this scheme, the raters were asked to label each comment as either positive, negative, neutral, mixed or sarcasm. The "mixed" here stands for a text containing both types of emotions. Our tools label texts as "neutral" where opposite emotions counter each other. From that perspective, we eventually classified the texts as "neutral" which were given a "mixed" rating by the human raters. We omitted the texts rated "sarcasm" by the human raters from our test data as there is no clear guideline on how to match its label with the possible ratings the tools have as output.

3.3 Politeness Annotation Scheme

We could not find any commonly used politeness scheme to rate textual documents. So, we developed and began with an experimental coding scheme[3] based on the work of Brown & Levinson's politeness theory [5] and Culpeper's work of impoliteness [8]. From these theories, we select the strategies that are relevant to written communication (filtering out the non-verbal cues of politeness). The strategies depend on the theory of "face", where positive face points towards the desire to be liked or appreciated or approved etc. and negative face is the desire to not to be imposed upon, intruded, or otherwise put upon.

Table 1: Coding Strategies for Politeness

Politeness		Impoliteness	
Strategies	Examples	Strategies	Examples
Positive Politeness: Helping hearer's positive face	Gratitude: "I really appreciate it"	Positive Impoliteness: threatening hearer's positive face	Seeking disagreement: "I don't agree with this style of coding"
Negative Politeness: Helping hearer's negative face	Use of Verbal Hedges: "I suggest we write it this way"	Negative Impoliteness: threatening hearer's negative face	Direct accusation: "You made these faulty changes"
Indirect Politeness: Offering advice through indirect implication	"It's really cold here" could imply a request to shut the window down	Sarcasm or Mock Politeness	having - sarcastic tone: "static??? really???"
–		Withhold Politeness[4]: Absence of politeness where it is expected	depends on context: Failing to thank somebody after a help

However, this being an experimental coding scheme, the raters were given the web page of the discussion thread containing the comment and were encouraged to dive into the detail of the context of the whole conversation and use their own judgment while rating. Additionally, they were asked to give remarks if they found any criterion that are not covered by the scheme or a strategy in the scheme that do not align with the real situation. The major criteria of our initial coding scheme are shown in Table 1, along with examples. Based on this scheme, the coders were asked to rate each text as very polite,polite,neutral,impolite, and very impolite. To measure the degree of politeness/impoliteness, the coders were suggested to

[3]https://tinyurl.com/ycts7vhr
[4]Excluded in the modified scheme after the experiment

consider the frequency of strategies used in the comment according to the scheme alongside their own judgment and the underlying context.

As mentioned in section 3.1, after the first iteration of rating, the Coder 1 had a discussion with the other coders about judging criteria and reaching a conclusion about the disagreed comments. Upon the discussion, we also modify our coding scheme and propose a complete and final one which is presented in Section 5.1. From the discussion, the coders also agreed that the initial coding scheme was not very clear on how to distinguish between the degrees of politeness/impoliteness, hence we got varying judgment on them. For the purpose of this study, we, therefore, merged "very polite" and "polite" into one single group of "polite" and "very impolite" and "impolite" into one group of "impolite".

3.4 Tool Selection

Sentiment analysis is an active field of research and also possesses commercial interest. Hence there are a lot of tools available for this. Jongeling and colleagues point out four tools as the most commonly used for sentiment analysis across all the domains including software engineering research (SentiStrength, NLTK, Alchemy, Stanford NLP). These tools, however, are trained on corpora that are not related to software engineering. Therefore, we take two other tools that have been trained on a dataset relevant to developer discussions (Senit4SD, SentiCR).

SentiStrength: Thelwall and colleagues' SentiStrength [39] was developed based on MySpace comments and has a "word strength list" at the core of its algorithm. The tool has been frequently used in software engineering research [15, 18, 19, 27, 35]. SentiStrength assigns an integer value between 1 to 5 for positive sentiment and -1 to -5 for negative sentiment. We add both the rating for a text, and identify as "positive" sentiment if the sum is greater than zero, "negative" if less than zero and "neutral" otherwise. This interpretation was followed in Jongeling's work [21].

Alchemy: IBM's Alchemy provides a text-processing API which returns a label for sentiment (positive/neutral/negative).

NLTK: Bird and colleagues' NLTK [2], which uses multiple corpora in its development, has also been used in previous works in the Software engineering domain [33, 34]. We use an API provided in www.text-processing.com to use this tool. For each text, it also returns a sentiment label (positive/neutral/negative).

Stanford NLP: Socher and colleagues' Stanford NLP divides the text into sentences, and performs a more advanced grammatical evaluation on each sentence by generating sentiment treebank through Recursive Neural Tensor Network [36]. It was trained on movie review excerpts from the rottentomatoes.com website. It returns an integer score for each sentence (0 for neutral, 1 for positive, -1 for negative) indicating its sentiment label. We rate a text under "positive" or "negative" based on the category that has greater number of sentences and "neutral" otherwise. This approach is similar to that of SentiStrength.

Senti4SD: This tool, developed by Calefato and colleagues [6], classifies each text by labeling their sentiment as positive, negative or neutral. The tool was built from scratch using questions, answers and comments from StackOverflow as its training data. Hence it has the claim to be of closer relevance to the software engineering domain.

SentiCR: Ahmed and colleagues' SentiCR [1] is trained on a dataset that comprises of code review comments from Gerrit and two other datasets developed in prior works [6, 30] using supervised learning techniques. Based on the oracle, the tool also return a sentiment label for a text (positive, neutral and negative).

For politeness analysis, we use a tool developed by Danescu-Niculescu-Mizil and his colleagues [10]. While the tool was trained on short texts that represent requests/questions, the tool identifies general patterns of politeness in written texts and has been used in recent software engineering studies. Ortu and colleagues have used this tool over a dataset collected from the Apache Software Foundation Issue Tracking system, the JIRA repository [28, 29]. They also show that sentiment and politeness are independent metrics having a weak correlation between [28]. As the tool does not have any distinct name, we will simply call it "Politeness tool".

Politeness tool: This tool [10] tries to measure politeness based on "domain-independent lexical and syntactic features operationalizing key components of politeness theory, such as indirection, deference, impersonalization and modality". It was trained and tested on different corpora (Wikipedia and Stack Exchange) and hence has the claim to be domain independent. The tool "predicts a politeness score between 0 to 1" for each text.

4 RESULTS

4.1 RQ1

In the first iteration, we had each comment rated by two coders individually on both sentiment and politeness. If for any comment, two coders agreed about the rating, we finalized that label for the comment. This resulted in 374 comments for politeness rating and 285 comments for sentiment rating. We call this the first set of data as the *"agreed"* set. In the second iteration, the Coder 1 had a discussion with each of the two other human coders. From this discussion, a conclusive rating was achieved for each of the initially disagreed comments. This resulted in the *"final"* data set containing the newly negotiated ones along with the agreed ones. This procedure aligns with a previous paper [1]. Also, during this discussion, some insights about how the comments should be rated both on "politeness" and "sentiment" were noted. These findings have been presented in Sections 5.1 and 5.2.

We used both the initially agreed and final set of data to evaluate our tools for RQ2 and RQ3. The amount of test data is comparable with the data set of a previous study which took 265 labeled texts to perform the evaluation of sentiment analysis tools [21].

Table 2 shows the inter-rater agreement during manual rating. We used Weighted Cohen's Kappa because our classes in both of the ratings have an implicit ordering. When two coders labeled different polarity for their ratings (positive vs negative and polite vs impolite), we counted it as a strong disagreement (weight=2). And when one of the labels was neutral, and other was a polar one, we counted it as a weak disagreement (weight=1).

Coder 1 had a fair and moderate agreement with one of the coders (Coder 2) for sentiment and politeness respectively and fair with the other coder (Coder 3) in both the ratings (according to magnitude guidelines presented by Landis and Koch [23]). This

Table 2: Inter-rater Agreement (Weighted Cohen's Kappa)

Weighted κ	Coder 1 and Coder 2	Coder 2 and Coder 3
Sentiment	.38	.27
Politeness	.48	.36

agreement rate is similar to what Murgia and colleagues found in their work for emotions like love, sadness, fear, and joy [25] and points towards the subjectivity of these affects. However, Coder 1's agreement with both the other coders on politeness is higher than sentiment. One reason behind this can be that the annotation scheme for politeness provided to the coders was more detailed with relevant examples than the simple scheme for sentiment.

The κ = .27 to .48 agreement suggests that even human ratings had low sentiment and politeness consistency on GitHub comments.

While we had substantial amount of disagreement between the coders in the first iteration and hence found the inconsistency of human rating, the coders reached a conclusion for the disagreed comments through discussion afterwards. This way, we complete a data set of comments that are hand-rated by humans and use it as a benchmark for our evaluation in RQ2 and RQ3.

4.2 RQ2

The initially agreed dataset contained 66 positives, 20 negatives, 198 neutral and 1 sarcasm out of total 285 comments. And for the final dataset, we got 93 positives, 73 negatives, 419 neutral and 4 sarcasm out of 589 comments. As our dataset is heavily "neutral" biased, only F-measures for the tools would be a misleading metric as the agreement by chance would be pretty high. So, we also calculated Weighted Cohen's Kappa as before, to compare the tools' output with the final human ratings. We got an output for each comment by all the tools except Alchemy, which failed to give an output for 46 and 70 comments respectively for the agreed and the final data set. For the first set of agreed data in Table 3, Senti4SD turned out to have the best performance among the tools. One point to note from the results is the relatively low F-measures with a low recall of negative comments for all the tools compared to neutral and positive comments. This confirms the "negative bias" of the existing tools, that is the misclassification of neutral technical texts as emotionally negative [3, 6, 27].

In Table 4, we show the results when we compare the tools with the final dataset. In this step, we see a major drop in the performance of all the tools except for SentiCR and a similar pattern of negative comments still doing worse.

The drop in the performance perhaps is not that surprising as we now added the comments that the human raters initially disagreed on between them. It indicates that these comments are harder to measure due to subtle subjectivity. However, the tools' performance over both the final and full dataset tells us about their overall unreliability.

Table 3: Performance of Sentiment Analysis Tools for First Set of Agreed Data

		F-measure		
tools	Weighted κ	neutral	positive	negative
SentiStrength	0.44	74.58%	61.54%	40.68%
NLTK	0.27	52.63%	55.42%	23.73%
Alchemy	0.33	58.78%	58.21%	32.65%
Stanford NLP	0.26	55.85%	59.83%	19.44%
Senti4SD	0.53	84.92%	68.18%	41.03%
SentiCR	0.13	57.04%	29.70%	29.91%

Table 4: Performance of Sentiment Analysis Tools for Final Set of Data

		F-measure		
tools	Weighted κ	neutral	positive	negative
SentiStrength	0.24	65.68%	43.01%	30.85%
NLTK	0.24	45.38%	45.26%	33.86%
Alchemy	0.23	51.56%	41.67%	34.48%
Stanford NLP	0.16	43.05%	45.09%	26.41%
Senti4SD	0.33	79.04%	50.47%	31.25%
SentiCR	0.24	64.87%	43.48%	31.02%

The κ = .16 to .33 agreement suggests that tools had low sentiment reliability on GitHub comments.

4.3 RQ3

For politeness, the initially agreed dataset contained 153 polite, 21 impolite and 200 neutral out of 374 comments while the final data set contained 221 polite, 46 impolite and 322 neutral out of 589 comments. As the politeness tool gives a rating between 0 to 1 for each text for its politeness, we have to first select a threshold where we can separate the polite, impolite and neutral comments. Over the first data set, we calculated F-measures at every .01 interval within the 0-1 range. We found the highest F-measure being 69.61% for polite at .57 threshold and 77.05% for neutral at .65 threshold. So, intuitively the ideal threshold to separate polite vs neutral would fall somewhere between that range. Our results align with the findings of Jongeling and colleagues, who found this threshold to be at .611 over their dataset in the software engineering domain. So, we use this threshold and rate all those comments as "polite" for which the tool gives a higher rating than .611.

However, we found very poor results for impolite comments as F-measures at all the ranges were below 20%. Thus, **we concluded that the tool has low reliability in deciding if a comment in our data set is impolite or not**. From this conclusion, we decided to merge the "neutral" and "impolite" class into one single class that is "non-polite". Thus, eventually we use the tool for a binary classification of the texts between polite and non-polite. We use our first agreed dataset of 374 comments for politeness rating to test the single tool and then with the final data set of 589 comments again. Similar to RQ1, we use Cohen's kappa (unweighted) and F-measures as our metrics. Table 5 shows that this tool also has "moderate" agreement over the first agreed data set while a "fair" agreement

Table 5: Performance of Politeness Analysis tool

First Data set			Final Data set		
	F-measure			F-measure	
κ	polite	neutral	κ	polite	neutral
.49	67.15%	75.94%	.39	60.24%	78.37%

over the final data set. We have the same rationale like sentiment analysis tools for the drop of performance as the added comments in the final dataset that were initially disagreed are harder to rate.

The κ = .39 agreement suggests that the tool had low politeness reliability on GitHub comments.

5 DISCUSSION

5.1 Final Politeness Coding Scheme

The coders had a discussion among themselves on the initial politeness coding scheme and we made some changes to the scheme based on that. We differentiated between implicit politeness and **explicit markers** of politeness within a text. For example, one coder rated a comment polite if he felt that the commenter is giving a detailed explanation of something and hence putting more effort into the communication. However, all the raters agreed that this is an implicit form of politeness. Another example would be the fourth strategy of impoliteness in our initial coding scheme which marks the absence of politeness where it is expected as a form of impoliteness. All the coders agreed that these are the implicit forms of politeness/impoliteness and cannot be clearly derived from a single comment without knowing the whole underlying context. Whereas, other strategies involve "explicit markers" like verbal hedges, indirection, and use of positive lexicons. So for the comments that were initially disagreed, we only counted those as 'polite' which have an explicit marker of politeness within the text itself.

Based on our discussion, we modified our initial experimental coding scheme and prepared a final one which has less ambiguous steps to detect politeness. We also give direction on how to rate the degree of politeness/impoliteness and what to do with the mixed comments. Although we did not use this final modified scheme in our work, we hope that this scheme will come useful in future works and help in maintaining a clear set of standards in rating politeness for texts found in online conversations. The scheme is also publicly available at https://tinyurl.com/y8g3b8zn

5.2 Insights on Sentiment Coding

The main confusion while rating the sentiment for the GitHub comments came from how to rate the statements of technical details containing *bug reports* or *bug fixes* (e.g. "fixed", "done") which are commonplace in Software engineering and don't necessarily always express an emotion. One coder initially rated all texts indicating bug fixes as positive as these comments mark positive results. However, the other coder rated the same texts as neutral unless they are explicitly emotive. Similar confusion arose with bug reports or crash reports. Upon discussion, the raters agreed that these texts are samples of normal day-to-day technical details in a software project. The raters reached a consensus that if a text doesn't explicitly show any emotions mentioned in the scheme, we would rate those as neutral. The initially disagreed comments were resolved on such conclusions. All of these confusions between the coders during rating points to the necessity of a customized annotation scheme for the software engineering which everyone can use as a standard one.

5.3 Tools' failure: Case analysis

To pinpoint the weaknesses in automated rating, we looked at the comments where the tools had a different rating from the humans. For the sentiment analysis tools, we perform a case study on Senti4SD, which had the best performance.

For many lexicons, Senti4SD might not have the accurate weight and hence inaccurately give a polar rating. The common words that are used to state technical detail like "error", "wrong" or "problem" have been rated as negative by the tool while the human raters found them neutral. There are also cases when the tool gave a neutral rating where humans found some pole of emotion. For example, the tool might not have an adequate weight for words like "thanks", "congrats", "LGTM" and the emoticons which human raters found as signs of positive sentiment. The tool could also not catch subtle criticism or frustration in the comments and failed on very short texts like "Yikes" and "Sorry". Again, for the texts containing mixed sentiments, the tool could not appropriately find the eventual intention of the commenter.

Other sentiment analysis tools did not exactly have the same points of differences and had varying precision and recalls over all the classes. The rationale behind this discrepancy is that different tools had different training data and therefore had different weights to different judging criteria. This also shows us the need of having a standard annotation scheme for software engineering domain, and train a tool based on that.

For the politeness tool, there are a lot of samples which went marginally wrong judging by the score that the tool provided for them. The tool also might not have enough weight for some positive lexicons and emoticons, the use of which human raters often found as polite. Similar to Senti4SD, this tool also has problems with very short texts. The failure for impolite comments is because impoliteness is much more subtle within the context. Also, even for some explicit syntactic features, like direct orders (e.g. "Do not add","see above","replace") came out as neutral by the tool. However, having distinct cases of failures for these tools indicate that an extensive training would help us to get better output from the tools.

6 THREATS TO VALIDITY

One major threat of our study is that we only had two human coders per each comment to annotate. Given by the subjectivity of such analysis, more coders could ensure more reliable human evaluation. Also, we had one coder rating all the comments and two other rating half of the data set each. This creates a possibility of human bias in our manual rating. However, we also present the results over the comments for which both the coders agree on the label without having a prior discussion. Besides this, Danescu-Niculescu-Mizil and his colleagues for their politeness tool used Brown & Levinson's theory to extract features of politeness out of

the data that were manually rated. Our coding scheme is also based on the same politeness theory. This biases our annotation of the comments towards the tool's internal working. If we had a different or no coding scheme during the rating, the performance of the politeness with respect to the human rating could have come out worse. Also, we removed the URLs before feeding the comments to the tool but not from the comments provided to the coders which creates a difference between the inputs. However, the human coders did not take the URLs into account as they are not commentary texts. Finally, while we randomly picked 589 comments, they might not be representative of the whole GitHub community.

7 CONCLUSION

Studying the emotions expressed by the developers is a fertile ground for research. However, in our study we find that the popular existing tools are unreliable in sentiment and politeness analysis over developer discussions. We also find that even humans have low consistency on correctly rating during these affect analyses. Through these conclusions, we demonstrate the need for a standard and more concrete definition of affects and their coding schemes to develop a reliable ground truth for affect analysis and train customized tools for the software engineering domain.

REFERENCES

[1] Toufique Ahmed, Amiangshu Bosu, Anindya Iqbal, and Shahram Rahimi. 2017. SentiCR: a customized sentiment analysis tool for code review interactions. In *Proceedings of the 32nd IEEE/ACM International Conference on Automated Software Engineering*. IEEE Press, 106–111.

[2] Steven Bird, Ewan Klein, and Edward Loper. 2009. *Natural language processing with Python: analyzing text with the natural language toolkit.* " O'Reilly Media, Inc.".

[3] Cássio Castaldi Araujo Blaz and Karin Becker. 2016. Sentiment analysis in tickets for it support. In *Mining Software Repositories (MSR), 2016 IEEE/ACM 13th Working Conference on*. IEEE, 235–246.

[4] Chris Brown, Justin Middleton, Esha Sharma, and Emerson Murphy-Hill. [n. d.]. How Software Users Recommend Tools to Each Other. ([n. d.]).

[5] Penelope Brown and Stephen C Levinson. 1987. *Politeness: Some universals in language usage*. Vol. 4. Cambridge university press.

[6] Fabio Calefato, Filippo Lanubile, Federico Maiorano, and Nicole Novielli. 2017. Sentiment Polarity Detection for Software Development. *Empirical Software Engineering* (2017), 1–31.

[7] Wikipedia contributors. 2018. Politeness — Wikipedia, The Free Encyclopedia. (2018). https://en.wikipedia.org/wiki/Politeness [Online; accessed 18 Jan, 2018].

[8] Jonathan Culpeper. 1996. Towards an anatomy of impoliteness. *Journal of pragmatics* 25, 3 (1996), 349–367.

[9] Bill Curtis, Herb Krasner, and Neil Iscoe. 1988. A field study of the software design process for large systems. *Commun. ACM* 31, 11 (1988), 1268–1287.

[10] Cristian Danescu-Niculescu-Mizil, Moritz Sudhof, Dan Jurafsky, Jure Leskovec, and Christopher Potts. 2013. A computational approach to politeness with application to social factors. *arXiv preprint arXiv:1306.6078* (2013).

[11] Prasun Dewan. 2015. Towards emotion-based collaborative software engineering. In *Proceedings of the Eighth International Workshop on Cooperative and Human Aspects of Software Engineering*. IEEE Press, 109–112.

[12] Felipe Ebert, Fernando Castor, Nicole Novielli, and Alexander Serebrenik. 2017. Confusion detection in code reviews. In *Software Maintenance and Evolution (ICSME), 2017 IEEE International Conference on*. IEEE, 549–553.

[13] Daviti Gachechiladze, Filippo Lanubile, Nicole Novielli, and Alexander Serebrenik. 2017. Anger and its direction in collaborative software development. In *Proceedings of the 39th International Conference on Software Engineering: New Ideas and Emerging Results Track*. IEEE Press, 11–14.

[14] Michael Gamon, Anthony Aue, Simon Corston-Oliver, and Eric Ringger. 2005. Pulse: Mining customer opinions from free text. *Lecture notes in computer science* 3646 (2005), 121–132.

[15] David Garcia, Marcelo Serrano Zanetti, and Frank Schweitzer. 2013. 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, 410–417.

[16] Georgios Gousios, Bogdan Vasilescu, Alexander Serebrenik, and Andy Zaidman. 2014. Lean GHTorrent: GitHub data on demand. In *Proceedings of the 11th working conference on mining software repositories*. ACM, 384–387.

[17] Daniel Graziotin, Xiaofeng Wang, and Pekka Abrahamsson. 2014. Happy software developers solve problems better: psychological measurements in empirical software engineering. *PeerJ* 2 (2014), e289.

[18] Emitza Guzman, David Azócar, and Yang Li. 2014. Sentiment analysis of commit comments in GitHub: an empirical study. In *Proceedings of the 11th Working Conference on Mining Software Repositories*. ACM, 352–355.

[19] Emitza Guzman and Bernd Bruegge. 2013. Towards emotional awareness in software development teams. In *Proceedings of the 2013 9th joint meeting on foundations of software engineering*. ACM, 671–674.

[20] Md Rakibul Islam and Minhaz F Zibran. 2016. Towards understanding and exploiting developers' emotional variations in software engineering. In *Software Engineering Research, Management and Applications (SERA), 2016 IEEE 14th International Conference on*. IEEE, 185–192.

[21] Robbert Jongeling, Proshanta Sarkar, Subhajit Datta, and Alexander Serebrenik. 2017. On negative results when using sentiment analysis tools for software engineering research. *Empirical Software Engineering* (2017), 1–42.

[22] Iftikhar Ahmed Khan, Willem-Paul Brinkman, and Robert M Hierons. 2011. Do moods affect programmersâĂŹ debug performance? *Cognition, Technology & Work* 13, 4 (2011), 245–258.

[23] J Richard Landis and Gary G Koch. 1977. The measurement of observer agreement for categorical data. *biometrics* (1977), 159–174.

[24] Saif Mohammad. 2016. A Practical Guide to Sentiment Annotation: Challenges and Solutions.. In *WASSA@ NAACL-HLT*. 174–179.

[25] Alessandro Murgia, Parastou Tourani, Bram Adams, and Marco Ortu. 2014. Do developers feel emotions? an exploratory analysis of emotions in software artifacts. In *Proceedings of the 11th working conference on mining software repositories*. ACM, 262–271.

[26] Nicole Novielli, Fabio Calefato, and Filippo Lanubile. 2014. Towards discovering the role of emotions in stack overflow. In *Proceedings of the 6th international workshop on social software engineering*. ACM, 33–36.

[27] Nicole Novielli, Fabio Calefato, and Filippo Lanubile. 2015. The challenges of sentiment detection in the social programmer ecosystem. In *Proceedings of the 7th International Workshop on Social Software Engineering*. ACM, 33–40.

[28] Marco Ortu, Bram Adams, Giuseppe Destefanis, Parastou Tourani, Michele Marchesi, and Roberto Tonelli. 2015. Are bullies more productive?: empirical study of affectiveness vs. issue fixing time. In *Proceedings of the 12th Working Conference on Mining Software Repositories*. IEEE Press, 303–313.

[29] Marco Ortu, Giuseppe Destefanis, Mohamad Kassab, Steve Counsell, Michele Marchesi, and Roberto Tonelli. 2015. Would you mind fixing this issue?. In *International Conference on Agile Software Development*. Springer, 129–140.

[30] Marco Ortu, Alessandro Murgia, Giuseppe Destefanis, Parastou Tourani, Roberto Tonelli, Michele Marchesi, and Bram Adams. 2016. The emotional side of software developers in JIRA. In *2016 IEEE/ACM 13th Working Conference on Mining Software Repositories (MSR)*. IEEE, 480–483.

[31] Bo Pang, Lillian Lee, et al. 2008. Opinion mining and sentiment analysis. *Foundations and Trends® in Information Retrieval* 2, 1–2 (2008), 1–135.

[32] Brian Parkinson, Peter Totterdell, Rob B Briner, and SA Reynolds. 1996. *Changing moods: The psychology of mood and mood regulation*. Longman.

[33] Daniel Pletea, Bogdan Vasilescu, and Alexander Serebrenik. 2014. Security and emotion: sentiment analysis of security discussions on GitHub. In *Proceedings of the 11th working conference on mining software repositories*. ACM, 348–351.

[34] Athanasios-Ilias Rousinopoulos, Gregorio Robles, and Jesús M González-Barahona. 2014. SENTIMENT ANALYSIS OF FREE/OPEN SOURCE DEVELOPERS: PRELIMINARY FINDINGS FROM A CASE STUDY/ANÁLISE DE SENTIMENTOS DE DESENVOLVEDORES DE SOFTWARE LIVRE: ACHADOS PRELIMINARES DE UM ESTUDO DE CASO. *Revista Electronica de Sistemas de Informaçao* 13, 2 (2014), 1.

[35] Vinayak Sinha, Alina Lazar, and Bonita Sharif. 2016. Analyzing developer sentiment in commit logs. In *Proceedings of the 13th International Conference on Mining Software Repositories*. ACM, 520–523.

[36] Richard Socher, Alex Perelygin, Jean Wu, Jason Chuang, Christopher D Manning, Andrew Ng, and Christopher Potts. 2013. Recursive deep models for semantic compositionality over a sentiment treebank. In *Proceedings of the 2013 conference on empirical methods in natural language processing*. 1631–1642.

[37] Igor Steinmacher, Tayana Conte, Marco Aurélio Gerosa, and David Redmiles. 2015. Social barriers faced by newcomers placing their first contribution in open source software projects. In *Proceedings of the 18th ACM conference on Computer supported cooperative work & social computing*. ACM, 1379–1392.

[38] Margaret-Anne Storey, Christoph Treude, Arie van Deursen, and Li-Te Cheng. 2010. 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, 359–364.

[39] Mike Thelwall, Kevan Buckley, Georgios Paltoglou, Di Cai, and Arvid Kappas. 2010. Sentiment strength detection in short informal text. *Journal of the Association for Information Science and Technology* 61, 12 (2010), 2544–2558.

[40] Jason Tsay, Laura Dabbish, and James Herbsleb. 2014. Let's talk about it: evaluating contributions through discussion in GitHub. In *Proceedings of the 22nd ACM SIGSOFT international symposium on foundations of software engineering*. ACM, 144–154.

[41] Ning Wang, W Lewis Johnson, Richard E Mayer, Paola Rizzo, Erin Shaw, and Heather Collins. 2008. The politeness effect: Pedagogical agents and learning outcomes. *International Journal of Human-Computer Studies* 66, 2 (2008), 98–112.