Sentiment Analysis for Software Engineering: How Far Can We Go?

Bin Lin
Software Institute
Università della Svizzera italiana (USI)
Switzerland

Fiorella Zampetti
Department of Engineering
University of Sannio
Italy

Gabriele Bavota
Software Institute
Università della Svizzera italiana (USI)
Switzerland

Massimiliano Di Penta
Department of Engineering
University of Sannio
Italy

Michele Lanza
Software Institute
Università della Svizzera italiana (USI)
Switzerland

Rocco Oliveto
STAKE Lab
University of Molise
Italy

ABSTRACT

Sentiment analysis has been applied to various software engineering (SE) tasks, such as evaluating app reviews or analyzing developers' emotions in commit messages. Studies indicate that sentiment analysis tools provide unreliable results when used out-of-the-box, since they are not designed to process SE datasets. The *silver bullet* for a successful application of sentiment analysis tools to SE datasets might be their customization to the specific usage context.

We describe our experience in building a software library recommender exploiting developers' opinions mined from Stack Overflow. To reach our goal, we retrained—on a set of 40k manually labeled sentences/words extracted from Stack Overflow—a state-of-the-art sentiment analysis tool exploiting deep learning. Despite such an effort- and time-consuming training process, the results were negative. We changed our focus and performed a thorough investigation of the accuracy of commonly used tools to identify the sentiment of SE related texts. Meanwhile, we also studied the impact of different datasets on tool performance. Our results should warn the research community about the strong limitations of current sentiment analysis tools.

CCS CONCEPTS

• **Information systems** → *Sentiment analysis*;

KEYWORDS

sentiment analysis, software engineering, NLP

ACM Reference Format:
Bin Lin, Fiorella Zampetti, Gabriele Bavota, Massimiliano Di Penta, Michele Lanza, and Rocco Oliveto. 2018. Sentiment Analysis for Software Engineering: How Far Can We Go?. In *ICSE '18: ICSE '18: 40th International Conference on Software Engineering , May 27-June 3, 2018, Gothenburg, Sweden.* ACM, New York, NY, USA, 11 pages. https://doi.org/10.1145/3180155.3180195

ICSE '18, May 27-June 3, 2018, Gothenburg, Sweden

ACM ISBN 978-1-4503-5638-1/18/05…$15.00
https://doi.org/10.1145/3180155.3180195

1 INTRODUCTION

Recent years have seen the rise of techniques and tools to automatically mine opinions from online sources [26]. The main application of these techniques is the identification of the mood and feelings expressed in textual reviews by customers (*e.g.,* to summarize the viewers' judgment of a movie [32]). Sentiment analysis [26] is a frequently used opinion mining technique. Its goal is to identify affective states and subjective opinions reported in sentences. In its basic usage scenario, sentiment analysis is used to classify customers' written opinions as negative, neutral, or positive.

The software engineering (SE) community has adopted sentiment analysis tools for various purposes. It has been used to assess the polarity of apps' reviews (*e.g.,* Goul *et al.* [6] and Panichella *et al.* [27]), and to identify sentences expressing negative opinions about APIs [38]. Tourani *et al.* [35] used sentiment analysis to identify distress or happiness in a development team, while Garcia *et al.* [5] found that developers expressing strong positive or negative emotions in issue trackers are more likely to become inactive in the open source projects they contribute. Ortu *et al.* [24] studied the impact of sentiment expressed in issues' comments and the issue resolution time, while Sinha *et al.* [31] investigated the sentiment of developers' commits.

Most prior works leverage sentiment analysis tools not designed to work on software-related textual documents. This "out-of-the-box" usage has been criticized due to the poor accuracy these tools achieved when applied in a context different from the one for which they have been designed and/or trained [16, 23, 35]. For example, the `Stanford CoreNLP` [32] opinion miner has been trained on movie reviews. In essence, *the silver bullet to make sentiment analysis successful when applied on software engineering datasets might be their customization to the specific context.*

Thus, the recent trend is to customize existing sentiment analysis tools to properly work on software engineering datasets [15, 36]. The most widely used tool in the SE community is `SentiStrength` [34]. `SentiStrength` assesses the sentiment of a sentence by looking at the single words the sentence is composed of, that is, it assigns positive/negative scores to the words and then sums up these scores to obtain an overall sentiment for the sentence. `SentiStrength` can be customized to provide the sentiment for domain-specific terms. For instance, Islam and Zibran [15] developed `SentiStrength – SE`, which improved identification performance for SE-related texts.

Inspired by these works, we started a research project to design and implement an approach to recommend software libraries to developers. The idea was to assess the quality of software libraries exploiting crowdsourced knowledge by mining developers' opinions on Stack Overflow. One key component needed to succeed was a reliable sentiment analysis tool (*e.g.,* to capture positive/negative developers' opinions about the usability of a library). Given the warning raised by previous work in our field [16, 23, 35] there was the need for training and customizing the sentiment analysis tool to the Stack Overflow context. Also, looking at the opinion mining literature, we decided to adopt a state-of-the-art approach based on a Recursive Neural Network (RNN), able to compute the sentiment of a sentence not by just summing up the sentiment of positive/negative terms, but by grammatically analyzing the way words compose the meaning of a sentence [32].

We built a training set by manually assigning a sentiment score to a total of ~40k sentences/words extracted from Stack Overflow. Despite the considerable manual effort, the empirical evaluation we performed led to negative results, with unacceptable accuracy levels in classifying positive/negative opinions. Given this, we started a thorough empirical investigation aimed at assessing the actual performance of sentiment analysis tools when applied on software engineering datasets with the goal of identifying a technique able to provide acceptable results. We experimented with all major techniques used in our community, by using them out-of-the-box as well as with customization designed to work in the software engineering context (*e.g.,* SentiStrength – SE [15]). Also, we considered three different software engineering datasets: (i) our manually built dataset of Stack Overflow sentences, (ii) comments left on issue trackers [25], and (iii) reviews of mobile apps [37].

Our results show that none of the state-of-the-art tools provides a precise and reliable assessment of the sentiments expressed in the manually labeled Stack Overflow dataset we built (*e.g.,* all the approaches achieve recall and precision lower than 40% on negative sentences). Results are marginally better in the app reviews and in the issue tracker datasets, which however represent simpler usage scenarios for sentiment analysis tools.

The goal of our paper is to share with the SE research community our negative findings, showing the current difficulties in applying sentiment analysis tools to software-related datasets, despite major efforts in tailoring them to the context of interest. Our results should also warn researchers to not simply use a (customized) sentiment analysis tool *assuming* that it provides a reliable assessment of the sentiments expressed in sentences, but to carefully evaluate its performance. Finally, we share our large training dataset as well as all the tools used in our experiments and the achieved results [18], to foster replications and advances in this novel field.

Structure of the paper. Section 2 presents the available sentiment analysis tools, and discusses sentiment analysis applications and studies in SE. Section 3 presents our original research plan. Section 4 reports and discusses the negative results we obtained when evaluating the sentiment analysis component of our approach. Section 5 reports the design and results of the study we performed to assess the performance of sentiment analysis tools on software engineering datasets, while Section 6 discusses the threats that could affect the validity of our results. Finally, after a discussion of lessons learned (Section 7), Section 8 concludes the paper.

2 RELATED WORK

We start by providing an overview of existing sentiment analysis tools and discuss the applications of these tools in the software engineering domain. Then, we present recent studies questioning the effectiveness of sentiment analysis when applied on SE-related datasets. Table 1 reports a summary of the main sentiment analysis tools used in software engineering application to date.

Table 1: Sentiment analysis tools used for SE applications.

Tool	Technique	Trained on	Used by
SentiStrength	Rule-based	MySpace	[7–11] [15, 22, 24] [31, 33] [36]
NLTK/VADER	Rule-based	Micro-Blog	[30] [28]
Stanford CoreNLP	Recurs. Neural Net	Movie Reviews	[29], our work
EmoText	Lexical Features	Stack Overflow, JIRA	[2]
SentiStrength – SE	SentiStrength	JIRA	[15]
Uddin and Khomh	Sentim. Orientation [13]	API Reviews	[36]

2.1 Sentiment Analysis Tools

There are several sentiment analysis tools available. Some of them are commercial tools, such as MeaningCloud[1], GetSentiment[2], or WatsonNaturalLanguageUnderstanding[3]. There are also sentiment analysis libraries available in popular machine learning tools, such as RapidMiner[4] or Weka [12], as well as SentiWordNet [1] an extension of a popular lexicon database (WordNet [20]) for sentiment analysis. The sentiment analysis tools applied to software engineering applications are:

- SentiStrength [34] is the most adopted one, originally trained on MySpace[5] comments. The core of SentiStrength is based on the *sentiment word strength list*, a collection of 298 positive and 465 negative terms with an associated positive/negative strength value. It also leverages a spelling correction algorithm as well as other word lists such as a *booster word list* and a *negating word list*, for a better sentiment assessment. SentiStrength assigns a sentiment score to each word composing a sentence under analysis, and derives the sentence sentiment by summing up the individual scores. The simple approach behind SentiStrength makes it easy to customize for a specific context by defining a list of domain-specific terms with associated sentiment scores. Despite this, only Islam and Zibran [15] adopted a customized version in software engineering.
- NLTK [14] is a lexicon and rule-based sentiment analysis tool having VADER (Valence Aware Dictionary and sEntiment Reasoner) at its core. VADER is specifically tuned to social media texts by incorporating a "gold-standard" sentiment lexicon extracted from microblog-like contexts and manually validated by multiple independent human judges.
- Stanford CoreNLP [32] is built on top of a Recursive Neural Network, which differs from SentiStrength and NLTK thanks to its ability to compute the sentiment of a sentence based on how words compose the meaning of the sentence, and not by summing up the sentiment of individual words. It has been trained on movie reviews.

[1] https://www.meaningcloud.com/developer/sentiment-analysis
[2] https://getsentiment.3scale.net/
[3] https://www.ibm.com/watson/services/natural-language-understanding/
[4] https://rapidminer.com
[5] https://myspace.com/

`EmoTxt` [2]. This is a toolkit for emotion recognition from text that combines a n-gram approach proposed by Ortu *et al.* [24] with lexical features conveying emotions in the input text: *emotion lexicon*, *politeness*, *positive and negative sentiment scores* (computed by using `SentiStrength`) and *uncertainty*. The novelty of `EmoTxt` relies on the recognition of specific emotions, such as joy, love, and anger. The tool has been preliminary evaluated on two datasets mined from Stack Overflow and JIRA [2].

2.2 Sentiment Analysis & Software Engineering

Sentiment analysis has been applied on different software engineering artifacts, such as technical artifacts (*e.g.,* issues and commit messages) and crowd-generated content (*e.g.,* forum messages and users' reviews), and to support different tasks.

Sentiment is commonly expressed in the developer-written commit messages and issues [17]. Guzman *et al.* [9] analyzed the sentiment of commit comments in GitHub and provided evidence that projects having more distributed teams tend to have a higher positive polarity in their emotional content. Additionally, the authors found that those comments written on Mondays tend to express more negative emotions. A similar study was conducted by Sinha *et al.* [31] on 28,466 projects within a seven year time frame. The results indicated that a majority of the sentiment was neutral and that Tuesdays seem to have the most negative sentiment overall. Also, the authors found a strong positive correlation between the number of files changed and the sentiment expressed by the commits the files were part of. Ortu *et al.* [24] analyzed the correlation between the sentiment in 560k JIRA comments and the time to fix a JIRA issue finding that positive sentiment expressed in the issue description might help issue fixing time. Finally, Souza and Silva [33] analyzed the relation between developers' sentiment and builds performed by continuous integration servers. They found that negative sentiment both affects and is affected by the result of the build process.

Analyzing the polarity of apps' reviews is particularly useful to support the evolution of mobile applications [3, 6, 11, 27]. Goul *et al.* [6] applied a sentiment analysis tool suite to over 5,000 reviews observing that sentiment analysis can address current bottlenecks to requirements engineering, but that certain types of reviews tend to elude algorithmic analysis. Carreño *et al.* [3] presented a technique based on Aspect and Sentiment Unification Model (ASUM) to extract common topics from apps' reviews and present users' opinions about those topics. Guzman *et al.* [8, 11] proposed the use of `SentiStrength` to support a similar task. Panichella *et al.* [27] used a Naive Bayes classifier to assign each sentence in users' reviews to a "sentiment class" among negative, neutral, and positive. This is one of the features they use to classify reviews on the basis of the information they bring (*e.g.,* feature request, problem discovery, etc.). Sentiment analysis has also been applied to classify tweets related to software projects [7]. The results of their empirical study indicated that searching for relevant information is challenging even if this relevant information can provide valuable input for software companies and support the continuous evolution of the applications discussed in these tweets.

As emotions can impact the developer productivity, task completion quality, and job satisfaction, sentiment analysis has also been used to detect the psychological state of developers [30]. Guzman and Bruegge [10] used sentiment analysis to investigate the role of emotional awareness in development teams, while Gachechiladze *et al.* [4] used sentiment analysis to build a fine-grained model for anger detection. In addition, the study by Pletea *et al.* [28] provided evidence that developers tend to be more negative when discussing security-related topics. Finally, Garcia *et al.* [5] analyzed the relation between the emotions and the activity of contributors in the Open Source Software project GENTOO. They found that contributors are more likely to become inactive when they express strong positive or negative emotions in the issue tracker, or when they deviate from the expected value of emotions in the mailing list.

Sentiment expressed on Q&A sites such as Stack Overflow is also leveraged by researchers to recommend comments on quality, deficiencies or scopes for further improvement for source code [29] or to identify problematic API design features [38].

2.3 Assessment of Sentiment Analysis Tools in Software Engineering Contexts

While the authors of the above works presented an extensive evaluation of the relationship between sentiment and other factors, no analysis is reported for what concerns the accuracy of the sentiment classification. Indeed, unsatisfactory results have been reported by researchers when using these sentiment analysis tools to analyze texts under software engineering contexts.

Tourani *et al.* [35] used `SentiStrength` to extract sentiment information from user and developer mailing lists of two major successful and mature projects from Apache software foundation: Tomcat and Ant. However, they found `SentiStrength` achieved a very low precision, *i.e.,* 29.56% for positive sentences and 13.1% for negative sentences. The low precision is caused by the ambiguous technical terms and the difficulty of distinguishing extreme positive/negative texts from neutral ones. Novielli *et al.* [23] highlighted and discussed the challenges of employing sentiment analysis techniques to assess the affective load of text containing technical lexicon, as typical in the social programmer ecosystem.

Jongeling *et al.* [16] conducted a comparison of four widely used sentiment analysis tools: `SentiStrength`, `NLTK`, `Stanford CoreNLP`, and `AlchemyAPI`. They evaluated their performance on a human labeled golden set from a developer emotions study by Murgia *et al.* [21] and found none of them can provide accurate predictions of expressed sentiment in the software engineering domain. They also observed that disagreement exists not only between sentiment analysis tools and the developers, but also between different sentiment analysis tools themselves. Their further experiment also confirmed that disagreement between these tools can result in contradictory results when using them to conduct software engineering studies.

The results achieved in these studies call for a sentiment analysis technique curated with software engineering related data to address the problem of low accuracy when dealing with technical terms.

Following this suggestion, sentiment analysis tools specific for software datasets have been proposed. Islam and Zibran [15] developed `SentiStrength – SE` based on `SentiStrength` to address the major difficulties by creating domain dictionary and introducing other heuristic rules. The presented evaluation showed that their tool significantly outperformed `SentiStrength`.

Uddin and Khomh [36] detected the polarity (positive, negative, neutral) of sentences related to API usage by using a customized version of the Sentiment Orientation algorithm [13]. The algorithm was originally developed to mine and summarize customer opinions about computer products. However, Uddin and Khomh customized the tool with words specific to API reviews. To the best of our knowledge, these are the only cases where the authors tried to customize state-of-the-art sentiment analysis tools to fit the software engineering domain.

3 METHODOLOGY

We briefly describe our initial plan to build a tool to recommend software libraries to developers given (i) a short description of a task at hand (*i.e.,* functional requirements) and (ii) a list of non-functional requirements considered more/less important by the developer for the specific implementation task (*e.g.,* security is of paramount importance, while high performance is nice to have but not really needed).

The basic idea was to leverage crowdsourced knowledge by mining opinions posted by developers while discussing on Q&A websites such as Stack Overflow. Our plan failed due to very poor results obtained when mining opinions from SE datasets. For this reason, while we present a detailed description of the opinion mining process we adopted, we only provide a brief overview of the overall idea and its different components.

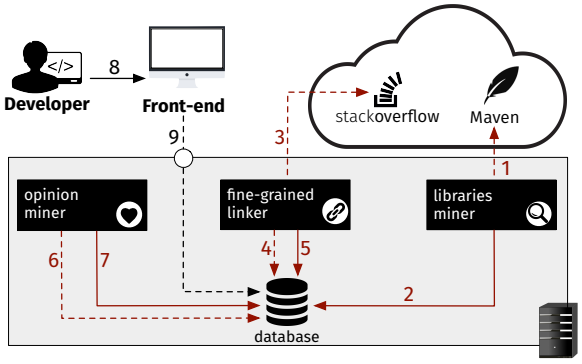

Figure 1: Our vision of the library recommender system.

The overall idea is depicted in Fig. 1. The dashed arrows represent dependencies (*e.g.,* ① and ③), while the full arrows indicate flows of information pushed from one component to another. The *libraries miner* mines from the maven central repository[6] all available Java libraries (① in Fig. 1). We extract for each library its: (i) name, (ii) description, (iii) link to the `jar` of the latest version, (iv) license, and (v) number and list of clients using it. All the information is stored in our database ②. The *fine-grained linker* mines Stack Overflow discussions to establish fine-grained links between the libraries stored in the database ④ and relevant sentences in Stack Overflow discussions ③.

[6]http://central.maven.org/maven2/maven/

Knowing the sentences related to a specific library, the *opinion miner* component can retrieve them ⑥, identify the expressed sentiments (*i.e., positive*, *neutral*, or *negative*), classify opinions on the basis of the non-functional requirements it refers to (*e.g.,* usability, performance, security, community support, etc.), and store them in the database ⑦.

Finally, the developer interested in receiving recommendations about software libraries submits a textual query describing the task in a Web-based front-end and important non-functional requirements ⑧.This information is provided to a Web service ⑨ to identify the most relevant and suitable libraries considering both functional and non-functional requirements.

In the following we detail our work to create the *opinion miner* component, where sentiment analysis plays a vital role. We report the **negative** results we achieved in Section 4.

3.1 Mining Opinions in Software Engineering Datasets

Previous work that attempted to mine opinions in SE datasets [16, 23, 35] offers a clear warning: *Using sentiment analysis/opinion mining techniques out-of-the-box on SE datasets is a recipe for negative results.* Indeed, these tools have been designed to work on user's reviews of products/movies and do not take into consideration domain-specific terms. For example, the word *robust* has a clear positive polarity when referred to a software product, while it does not express a specific sentiment in a movie review. This pushed researchers to create customized versions of these tools, enriching them with information about the sentiment of domain-specific terms (*e.g.,* `SentiStrength – SE` by Islam and Zibran [15]).

Despite the effort done by some authors in developing customized tools, there is a second major limitation of the sentiment analysis tools mostly used in SE (*e.g.,*SentiStrength [34]). Such tools assess the sentiment of a sentence by looking at the single words in isolation, assigning positive/negative scores to the words and then summing these scores to obtain an overall sentiment for the sentence. Thus, the sentence composition is ignored. For example, a sentence such as "*I would not recommend this library, even though it is robust and fast*" would be assessed by these techniques as positive in polarity, given the presence of words having a positive score (*i.e.,* robust, fast). Such a limitation has been overcome by the `Stanford CoreNLP` [32] approach used for the analysis of sentiment in movies' reviews. The approach is based on a Recursive Neural Network (RNN) computing the sentiment of a sentence based on how words compose the meaning of the sentence [32]. Clearly, a more advanced approach comes at a cost: The effort required to build its training set. Indeed, it is not sufficient to simply provide the polarity for a vocabulary of words but, to learn how positive/negative sentences are grammatically built on top of positive/negative words, it needs to know the polarity of all intermediate nodes composing a sentence used in the training set.

We discuss the example reported in Fig. 2. Gray nodes represent (sequences of) words having a *neutral* polarity, red ones indicate *negative* sentiment, green ones *positive* sentiment. Overall, the sentence has a negative sentiment (see the root of the tree in Fig. 2), despite the presence of several positive terms (the tree's leafs) and intermediate nodes.

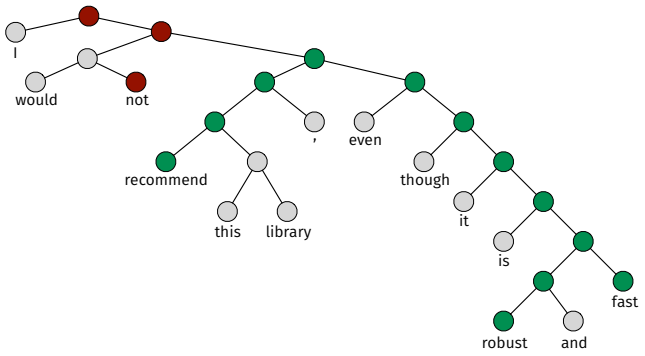

Figure 2: Example of the labeling needed to build the Stanford CoreNLP training set.

To use this sentence composed of 14 words in the training set of the RNN, we must provide the sentiment of all 27 nodes depicted in Fig. 2. This allows the RNN to learn that while "*it is robust and fast*" has a positive polarity if taken in isolation, the overall sentence is expressing a negative feeling about the library due to the "*I would not recommend this library*" sub-sentence.

Given the high context-specificity of our work to SE datasets (*i.e.,* Stack Overflow posts), we decided to adopt the `Stanford CoreNLP` tool [32], and to invest a substantial effort in creating a customized training set for it. Indeed, as highlighted in previous work [16, 23, 35], it makes no sense to apply an approach trained on movie reviews on SE datasets.

3.1.1 Building a Training Set for the Opinion Miner. We extracted from the latest available Stack Overflow dump (dated July 2017) the list of all discussions (i) tagged with Java, and (ii) containing one of the following words: *library*/*libraries*, *API*(*s*). Given our original goal (*i.e.*, recommending Java libraries on the basis of crowdsourced opinions), we wanted to build a training set as domain-specific as possible for the RNN. By applying these filters, we collected 276,629 discussions from which we extracted 5,073,452 sentences by using the `Stanford CoreNLP` toolkit [19]. We randomly selected 1,500 sentences and manually labeled them by assigning a sentiment score to the whole sentence and to every node composing it.

The labeling process was performed by five of the authors (from now on, evaluators) and supported by a Web application we built. The Web app showed to each evaluator a node (extracted from a sentence) to label with a sentiment going from -2 to +2, with -2 indicating strong negative, -1 weak negative, 0 neutral, +1 weak positive, and +2 strong positive score. The choice of the five-levels sentiment classification was not random, but driven by the observation of the movie reviews training set made publicly available by the authors of the `Stanford CoreNLP` [32] sentiment analysis tool[7]. Note that a node to evaluate could be a whole sentence, an intermediate node (thus, a sub-sentence), or a leaf node (*i.e.,* a single word). To avoid any bias, the Web app did not show to the evaluator the complete sentence from which the node was extracted. Indeed, knowing the context in which a word/sentence is used could introduce a bias in the assessment of its sentiment polarity. Finally, the Web application made sure to have two evaluators for each node, thus reducing the subjectivity bias. This process, which took ~90 working hours of manual labeling, resulted in the total labeling of the sentiment polarity for 39,924 nodes (*i.e.,* 19,962 nodes extracted from the 1,500 sentences × 2 evaluators per node).

Once the labeling was completed, two of the authors worked on conflicts resolution (*i.e.,* cases in which two evaluator assigned a different sentiment to the same node). All the 279 conflicts involving complete sentences (18.6% of the labeled sentences) were fixed. Indeed, it is of paramount importance to assign a consistent and double-checked sentiment to the complete sentences, considering the fact that they will be used as a ground truth to evaluate our approach. Concerning the intermediate/leaf nodes, we had a total of 2,199 conflicts (11.9% of the labeled intermediate/leaf nodes). We decided to only manually solve 123 strong conflicts, meaning those for which there was a score difference ≥ 2 (*e.g.,* one of the evaluators gave 1, the other one -1), while we automatically process the 2,076 having a conflict of only one point. Indeed, slight variations of the assigned sentiment (*e.g.,* one evaluator gave 1 and the other 2) are expected due to the subjectivity of the task. The final sentiment score was s, in case there was agreement between the evaluators, while it was $round[(s_1 + s_2)/2]$ in case of unsolved conflict, where *round* is the rounding function to the closest integer value and s_i is the sentiment assigned by the i^{th} evaluator.

4 NEGATIVE RESULTS

Before incorporating the opinion miner component, we decided to assess it individually, and not in the context of the whole library recommendation task. We performed this assessment on the dataset of manually labeled 1,500 sentences. Among those sentences, 178 are positive, 1,191 are neutral, and 131 are negative. We performed a ten-fold cross validation: We divided the 1,500 sentences into ten different sets, each one composed of 150 sentences. Then, we used a set as a *test set* (we only use the 150 complete sentences in the test set, and not all their intermediate/leaf nodes), while the remaining 1,350 sentences, with all their labeled intermediate/leaf nodes, were used for *training*[8]. Since we are mostly interested in discriminating between *negative*, *neutral*, and *positive* opinions, we discretized the sentiment in the test set into these three levels. Sentences labeled with "-2" and "-1" are considered negative (-1), those labeled with "0" neutral (0), and those labeled with "+1" and "+2" as positive (+1). We discretized the output of the RNN into the same three levels. We assessed the accuracy of the opinion miner by computing recall and precision for each category. Computing the overall accuracy would not be effective, given the vast majority of *neutral* opinions in our dataset (*i.e.,* a constant *neutral* classifier would obtain a high accuracy, ignoring *negative* and *positive* opinions).

Table 2 reports the results achieved by applying `Stanford CoreNLP SO`[9] on sentences extracted from Stack Overflow discussions.

[7] https://nlp.stanford.edu/sentiment/trainDevTestTrees_PTB.zip

[8] The `StanfordCoreNLP` tool requires—during the training of the neural network—a so called *development* set to tune some internal parameters of the network. Among the 1,350 sentences with intermediate/leaf nodes in training set we randomly selected 300 sentences for composing the development set at each run.

[9] `Stanford CoreNLP SO` is the name of the tool with our new model trained with Stack Overflow discussions, while `StanfordCoreNLP` is the sentiment analysis component of `StanfordCoreNLP` with the default model trained using movie reviews.

Table 2: Testing results of Stanford CoreNLP sentiment analyzer with new model trained with Stack Overflow discussions.

batch	# correct prediction	# positive sentences	positive precision	positive recall	# neutral sentences	neutral precision	neutral recall	# negative sentences	negative precision	negative recall
1	113	10	0.250	0.200	118	0.835	0.898	22	0.333	0.227
2	112	15	0.294	0.333	118	0.853	0.839	17	0.471	0.471
3	116	15	0.000	0.000	121	0.819	0.934	14	0.273	0.214
4	123	9	0.600	0.333	122	0.875	0.918	19	0.471	0.421
5	110	10	0.167	0.100	119	0.833	0.840	21	0.375	0.429
6	129	11	0.600	0.273	118	0.891	0.975	21	0.688	0.524
7	93	6	0.111	0.167	130	0.911	0.631	14	0.196	0.714
8	117	17	0.400	0.118	116	0.809	0.948	17	0.556	0.294
9	111	18	0.333	0.056	113	0.770	0.947	19	0.375	0.158
10	115	20	1.000	0.050	116	0.799	0.957	14	0.300	0.214
Overall	1139	131	0.317	0.145	1191	0.836	0.886	178	0.365	0.365

The table shows the number of correct predictions, the number of positive/neutral/negative sentences in the batch of testing sets and the corresponding precision/recall values, while the last row reports the overall performance on the whole dataset. Table 3 shows some concrete examples of sentiment analysis with `Stanford CoreNLP SO`.

Table 3: Examples of sentiment analysis results of Stanford CoreNLP SO.

Sentence	Oracle	Prediction
It even works on Android.	Positive	Positive
Hope that helps some of you with the same problem.	Positive	Negative
There is a central interface to access this API.	Neutral	Neutral
How is blocking performed?	Neutral	Negative
I am not able to deploy my App Engine project locally.	Negative	Negative
Anyway, their current behavior does not allow what you want.	Negative	Neutral

The results shown in Table 2 highlight that, despite the specific training, `Stanford CoreNLP SO` *does not achieve good performance in analyzing sentiment of Stack Overflow discussions.* Indeed, its precision and recall in detecting positive and negative sentiments is below 40%, thus discouraging its usage as a fundamental part of a recommendation system. Although `Stanford CoreNLP SO` can correctly identify more negative than positive sentences, only a small fraction of sentences with positive/negative sentiment is identified. Also, there are more mistakenly than correctly identified sentences in both sets.

Based on the results we achieved, it is impracticable to build on the top of `Stanford CoreNLP SO` an effective recommender system for libraries: The high percentage of wrong sentiment classification will likely result in the recommendation of the wrong library. Thus, besides the huge effort we spent to train `Stanford CoreNLP SO` with a specific and large software dataset, we failed in achieving an effective sentiment analysis estimator. For this reason, we decided to change our original plan and perform a deeper analysis of the accuracy of sentiment analysis tools when used on software-related datasets. Specifically, we aim to understand whether (i) domain specific training data really helps in increasing the accuracy of sentiment analysis tool; and whether (ii) other state-of-the-art sentiment analysis tools are able to obtain good results on software engineering datasets, including our manually labeled Stack Overflow dataset. Understanding how these tools perform can also help us in gaining deeper insights into the current state of sentiment analysis for software engineering.

5 EVALUATING SENTIMENT ANALYSIS FOR SOFTWARE ENGINEERING

The *goal* of the study is to analyze the accuracy of sentiment analysis tools when applied to software engineering datasets, with the *purpose* of investigating how different contexts can impact their effectiveness. The *context* of the study consists of text extracted from three software-related datasets, namely Stack Overflow discussions, mobile app reviews, and JIRA issue comments.

5.1 Research Questions and Context

The study aims to answer the following research questions:

- **RQ$_1$:** *How does our* `Stanford CoreNLP SO` *perform compared to other sentiment analysis tools?* We want to verify whether other state-of-the-art tools are able to achieve better accuracy on the Stack Overflow dataset we manually built, thus highlighting limitations of `Stanford CoreNLP SO`. Indeed, it could be that our choice of the `Stanford CoreNLP` and therefore of developing `Stanford CoreNLP SO` was not the most suitable one, and other existing tools already provide better performance.
- **RQ$_2$:** *Do different software-related datasets impact the performance of sentiment analysis tools?* We want to investigate the extent to which, analyzing other kinds of software engineering datasets, *e.g.,* issue comments and app reviews, sentiment analysis tools would achieve different performance than for Stack Overflow posts. For example, such sources might contain less neutral sentences and, the app reviews in particular, be more similar to the typical training sets of sentiment analysis tools.

The context of the study consists of textual documents from three different SE repositories, *i.e.,* (i) Question & Answer forums, *i.e.,* Stack Overflow discussions, (ii) app stores, *i.e.,* users' reviews on mobile apps, and (iii) issue trackers, *i.e.,* JIRA issue comments.

We chose these types of textual documents as they have been studied by SE researchers, also in the context of sentiment analysis [2, 24, 27, 36]. As our goal is to evaluate the accuracy of different sentiment analysis tools on these three datasets, we need to define the ground truth sentiment for each of the sentences/texts they contain.

The following process was adopted to collect the three datasets and define their ground truth:

- **Stack Overflow discussions.** We reuse the ground truth for the 1,500 sentences used to evaluate `Stanford CoreNLP SO`.
- **Mobile app reviews.** We randomly selected 341 reviews from the dataset of 3k reviews provided by Villarroel *et al.* [37], which contains manually-labeled reviews classified on the basis of the main information they contain. Four categories are considered: *bug reporting*, *suggestion for new feature*, *request for improving non-functional requirements* (*e.g.,* performance of the app), and *other* (meaning, reviews not belonging to any of the previous categories). When performing the random selection, we made sure to respect the proportion of reviews belonging to the four categories in the original population in our sample (*e.g.,* if 50% of the 3k reviews belonged to the "other" category, we randomly selected 50% of our sample from that category). The 341 selected reviews represent a statistically significant sample with 95% confidence level ±5% confidence interval.
  Once selected, we manually labeled the sentiment of each review. The labeling process was performed by two of the authors (from now on, evaluators). The evaluators had to decide where the text is positive, neutral, or negative. A third evaluator was involved to solve 51 conflict cases.
- **JIRA issue comments.** We use the dataset collected by Ortu *et al.* [25], containing 4k sentences labeled by three raters with respect to four emotions: *love*, *joy*, *anger*, and *sadness*. This dataset has been used in several studies as the "golden set" for evaluating sentiment analysis tools [15, 16]. During the original labeling process, each sentence was labeled with one of six emotions: *love*, *joy*, *surprise*, *anger*, *sadness*, *fear*. Among these six emotions, *love*, *joy*, *anger*, and *sadness* are mostly expressed. As also done by Jongeling *et al.* [16], we map the sentences with the label *love* or *joy* into positive sentences, and those with label *anger* or *sadness* into negative sentences.

Table 4 reports for each dataset (i) the number of sentences extracted, and (ii) the number of positive, neutral, negative sentences.

Table 4: Dataset used for evaluating sentiment analysis tools in software engineering

Dataset	# sentences	# positive	# neutral	# negative
Stack Overflow	1,500	178	1,191	131
App reviews	341	186	25	130
JIRA issue	926	290	0	636

5.2 Data Collection and Analysis

On the three datasets described above we experimented with the following tools, which are popular in the SE research community:

- **SentiStrength**. `SentiStrength` does not give the sentiment of the text directly, instead, it reports two sentiment strength scores of the text analyzed: one score for the negative sentiment expressed in the text from -1 (not negative) to -5 (extremely negative), the other for the positive sentiment expressed from 1 (not positive) to 5 (extremely positive). We sum these two scores, and map the sum of over 0, 0, and below 0 into positive, neutral, and negative, respectively.
- **NLTK**. Based on VADER Sentiment Analysis, `NLTK` reports four sentiment strength scores for the text analyzed: "negative", "neutral", "positive", and "compound". The scores for "negative", "neutral", and "positive" range from 0 to 1, while the "compound" score is normalized to be between -1 (most extreme negative) and +1 (most extreme positive). As suggested by the author of the VADER component[10], we use the following thresholds to identify the sentiment of the text analyzed: $score \geq 0.5$: positive; $-0.5 < score < 0.5$: neutral; $score \leq -0.5$: negative.
- **Stanford CoreNLP**. By default, `Stanford CoreNLP` reports the sentiment of the text on a five-value scale: very negative, negative, neutral, positive, and very positive. Since we are only interested in discriminating between negative, neutral, and positive opinions, we merged very negative into negative, and very positive into positive.
- **SentiStrength-SE**. As it is a tool based on `SentiStrength`, and uses the same format of reported results, we interpret its sentiment score by adopting the same approach we used for `SentiStrength`.
- **Stanford CoreNLP SO**. Similarly, we use the same approach adopted for `Stanford CoreNLP` to convert five-scale values into three-scale values. To examine the performance on app reviews and JIRA issue comments, we used the Stack Overflow labeled sentences (including internal nodes) as training set[11].

We assess the accuracy of the tools by computing recall and precision for each of the three considered sentiment categories (*i.e.,* positive, neutral, negative) in each dataset.

5.3 Results

Table 5 reports the results we achieved by applying the five sentiment analysis approaches on the three different SE datasets. The table reports the number of correct predictions made by the tool, and precision/recall for predicting sentiment of positive/neutral/negative sentences. For each dataset/metric, the best achieved results are highlighted in **bold**. In the following we discuss the achieved results aiming at answering our research questions.

5.3.1 RQ_1: How does our `Stanford CoreNLP SO` perform as compared to other sentiment analysis tools? To answer RQ_1, we analyze the results achieved by the five tools on the Stack Overflow dataset we built.

As for the comparison of `Stanford CoreNLP SO` with the original model of `Stanford CoreNLP`, the results show that on neutral sentences `Stanford CoreNLP SO` achieves a better recall while keeping almost the same level of precision. Also, on positive and negative sentences `Stanford CoreNLP SO` is still able to provide a good increment of the precision.

[10] https://github.com/cjhutto/vaderSentiment
[11] In this case, 20% of the training set was used as development set.

Table 5: Evaluation results for sentiment analysis tools applied in software engineering domain. In bold the best results.

dataset	tool	# correct prediction	positive precision	positive recall	neutral precision	neutral recall	negative precision	negative recall
Stack Overflow	*SentiStrength*	1,043	0.200	**0.359**	0.858	0.772	0.397	0.433
	NLTK	1,168	**0.317**	0.244	0.815	**0.941**	**0.625**	0.084
	Stanford CoreNLP	604	0.231	0.344	**0.884**	0.344	0.177	**0.837**
	SentiStrength-SE	**1,170**	0.312	0.221	0.826	0.930	0.500	0.185
	Stanford CoreNLP SO	1,139	**0.317**	0.145	0.836	0.886	0.365	0.365
App reviews	*SentiStrength*	213	0.745	**0.866**	0.113	0.320	0.815	0.338
	NLTK	184	0.751	0.812	0.093	**0.440**	**1.000**	0.169
	Stanford CoreNLP	**237**	**0.831**	0.715	**0.176**	0.240	0.667	**0.754**
	SentiStrength-SE	201	0.741	0.817	0.106	0.400	0.929	0.300
	Stanford CoreNLP SO	142	0.770	0.253	0.084	0.320	0.470	0.669
JIRA issues	*SentiStrength*	**714**	0.850	**0.921**	-	-	0.993	0.703
	NLTK	276	0.840	0.362	-	-	**1.000**	0.269
	Stanford CoreNLP	626	0.726	0.621	-	-	0.945	0.701
	SentiStrength-SE	704	**0.948**	0.883	-	-	0.996	**0.704**
	Stanford CoreNLP SO	333	0.635	0.252	-	-	0.724	0.409

However, in this case the increment of precision has a price to pay: `Stanford CoreNLP SO` provides levels of recall lower than `Stanford CoreNLP`. The comparison between `Stanford CoreNLP` and `Stanford CoreNLP SO` should be read taking into account that the original `Stanford CoreNLP` model is trained on over 10k labeled sentences (*i.e.,* >215k nodes). `Stanford CoreNLP SO` is trained on a smaller training set. Thus, it is possible that a larger training set could improve the performance of `Stanford CoreNLP SO`. However, as of now, this is a mere conjecture.

When looking at other tools, the analysis of the results reveal that all the experimented tools achieve comparable results and—more important—none of the experimented tools is able to reliably assess the sentiment expressed in a Stack Overflow sentence. Indeed, while all the tools are able to obtain good results when predicting neutral sentences, their accuracy falls when working on positive and negative sentences. For example, even considering the tool having the highest recall for identifying positive sentences (*i.e.,* `SentiStrength`) (i) there is only 35.9% chance that it can correctly spot a positive sentence and (ii) one out of five sentences that it will label as positive will be actually false positives (precision=20%). The recall is almost the same as randomly guessing which has 33.3% chance of success. These results reveal that there is still a long way to go before researchers and practitioners can use state-of-the-art sentiment analysis tools to identify the sentiment expressed in Stack Overflow discussions.

RQ_1 main findings: (i) the training of `Stanford CoreNLP` on SO discussions does not provide a significant improvement as compared to the original model trained on movie reviews; (ii) the prediction accuracy of all tools are biased towards the majority class (neutral) for which a very good precision and recall is almost always achieved; and (iii) all tools achieve similar performance and it is impossible to identify among them a clear winner or, in any case, a tool ensuring sufficient sentiment assessment of sentences from Stack Overflow discussions.

5.3.2 RQ_2: Do different software-related datasets impact the performance of sentiment analysis tools? To answer RQ_2, we compare the accuracy of all tools on the three datasets considered in our study. When we look at results for app reviews, we can see that, differently from what observed in the Stack Overflow dataset, most tools can predict positive texts with reasonable precision/recall values. Even for negative reviews, the results are in general much better. It is worth noting that `Stanford CoreNLP` is competitive for identifying positive and negative sentiment as compared to other tools. Indeed, compared to other texts in software engineering datasets, such as Stack Overflow discussions and JIRA issues, app reviews can be less technical and relatively more similar to movie reviews, with which the original model of `Stanford CoreNLP` is trained. However, when identifying neutral app reviews, all tools exhibit poor accuracy. This is likely due to the fact that, while positive and negative app reviews could be easily identified by the presence/absence of some "marker terms" (*e.g.,* the presence of the *bug* term is likely related to negative reviews), this is not the case for the neutral set of reviews, in which a wider and more variegate vocabulary might be used.

When inspecting results for JIRA issue comments, we find that `SentiStrength` and `SentiStrength − SE` have better accuracy than other tools, with `SentiStrength − SE` providing a better precision-recall balance across the two categories of sentiment (*i.e.,* positive and negative). Despite the mostly good results achieved by the experimented tools on the JIRA dataset, there are some important issues in the evaluations performed on this dataset.

First, the absence of neutral sentences does not provide a clear and complete assessment of the accuracy of the tools. Indeed, as shown in the app reviews, neutral texts might be, in some datasets, the most difficult to identify, likely due to the fact that they represent that "grey zone" close to both positive and negative sentiment.

Second, the JIRA dataset is built by mapping emotions expressed in the comments (*e.g.,* joy or love) into sentiments (*e.g.,* positive).

Table 6: Confusion matrices on the Stack Overflow dataset.

	SentiStrength		
	positive	neutral	negative
positive	47	66	18
neutral	173	919	99
negative	15	86	77

	NLTK		
	positive	neutral	negative
positive	32	96	3
neutral	64	1121	6
negative	5	158	15

	Stanford CoreNLP		
	positive	neutral	negative
positive	45	30	56
neutral	145	410	636
negative	5	24	149

	SentiStrength-SE		
	positive	neutral	negative
positive	29	93	9
neutral	59	1108	24
negative	5	140	33

	Stanford CoreNLP SO		
	positive	neutral	negative
positive	19	96	16
neutral	39	1055	97
negative	2	111	65

However, such a mapping does not always hold. For instance, positive comments in issue tracker does not always express joy or love (*e.g., thanks for the updated patch*), thus allowing to obtain a very partial view of the accuracy of sentiment analysis tools.

To highlight the importance of *neutral* items in the evaluation of a sentiment analysis tool, Table 6 shows the confusion matrices obtained by the five different sentiment analysis tools on the Stack Overflow dataset (see Table 4).

All tools are effective in discriminating between positive and negative items. For example, our `Stanford CoreNLP SO` only misclassified two negative sentences as positive, and 16 positive sentences as negative. `NLTK` only misclassifies five negative sentences as positive, and three positive sentences as negative. The errors are mostly due to negative/positive sentences classified as neutral and *vice versa.* This confirms the issues found by Tourani *et al.* [35] when using `SentiStrength` on SE data, and this is why evaluating sentiment analysis tools on datasets not containing neutral sentences introduces a considerable bias. Similar observations hold for the app reviews dataset, in which the performance in classifying neutral reviews is, as shown in Table 5, extremely poor.

RQ_2 main findings: The accuracy of sentiment analysis tools is, in general, poor on software engineering datasets. We claim this because we found no tool able to reliably discriminating between positive/negative and neutral items. Indeed, while the accuracy on the app reviews and JIRA datasets are acceptable (i) in the app reviews dataset the accuracy in identifying neutral items is very low, and (ii) the data obtained with the JIRA dataset can not be considered as reliable due to the discussed issues.

6 THREATS TO VALIDITY

Threats to *construct validity* concern the relation between theory and observation. The first concern is related to our manual sentiment labeling. Sentiment expressed in the text might be misinterpreted by people. Also, the labeling might be impacted by subjective opinions of evaluators. Although we adopted an additional conflict resolving process, it is not guaranteed that the manually assigned sentiment is always correct.

Another threat is the sentiment score mapping, *i.e.,* mapping five-scale sentiment to three-scale sentiment. Indeed, sentiment expressed in the text have different degrees. Predicting slightly negative sentence as neutral should be considered a smaller mistake then predicting a very negative sentence as neutral, since the threshold to draw a line between the neutral and the negative sentiment can be more subjective.

Threats to *internal validity* concern internal factors we did not consider that could affect the variables and the relations being investigated. In our study, they are mainly due to the configuration of sentiment analysis tools/approaches we used. In most cases, we use the default or suggested parameters, for example, the threshold for `NLTK`. However, some parameters might be further tuned to increase the sentiment prediction performance.

Threats to *conclusion validity* concern the relation between the treatment and the outcome. We randomly selected sentences from Stack Overflow discussions and app reviews from an existing dataset [37]. While we considered statistically significant samples, we cannot guarantee that our samples are representative of the whole population.

Threats to *external validity* concern the generalizability of our findings. While the evaluation has considered the most commonly used sentiment analysis tools in software engineering, some less popular tools might have been ignored. Constantly there are lots of new ideas and approaches popping up in the natural language processing domain, but few of them have been examined and verified in the software engineering context. Since our goal is to seek a good sentiment analysis tool for software-related texts, in this paper we only select the tools already used in previous software engineering studies. Our datasets are limited to three frequently mined software engineering repositories, while texts in other contexts, such mailing list and IRC chats, are not considered.

7 LESSONS LEARNED

No tool is ready for real usage of identifying sentiment expressed in SE related discussions yet. No tool, including the ones specifically customized for certain software engineering tasks, is able to provide precision and recall levels sufficient to entail the tool adoption for a task such as recommending software libraries.

By relying on such tools, we would certainly generate wrong recommendations and miss good ones. Our results are a warning to the research community: Sentiment analysis tools should always be carefully evaluated in the specific context of usage before building something on top of them. For example, while Uddin and Khomh [36] presented a very interesting approach to mine APIs opinions from Stack Overflow, they do not report the accuracy of the sentiment analysis component they exploit to identify positive/negative opinions about APIs.

Specific re-training is required, but does not represent a silver bullet for improving the accuracy. Previous literature has pointed our that sentiment analysis tools cannot be used out-of-the-box for software engineering tasks [15, 16, 23, 35]. In some cases, tools have introduced a data preprocessing or a re-training to cope with the specific software engineering lexicon, in which there are positive or negative words/sub-sentences that are not positive or negative in other contexts, or *vice versa* (*e.g.,* the word *bug* generally carries a negative sentiment when referred to a library, while it can be considered neutral in movie reviews). However, as results have shown, this might still be insufficient to guarantee good accuracy in terms of both precision and recall on all polarity levels. Also, customization is very dataset specific, and therefore applying the tool on different datasets would require a new training. In other words, customizing a sentiment analysis tool for JIRA does not make it ready for Stack Overflow and *vice versa*. Finally, some algorithms, such as recursive neural networks, require costly re-training. In our case, the training performed with 1,500 sentences (which turned into labeling almost 40k nodes) revealed to be insufficient for a clear improvement of the `Stanford CoreNLP` accuracy.

Some software engineering applications make sentiment analysis easier than others. Sentiment analysis tools perform better on app reviews. App reviews contain sentences that, in most cases, clearly express the opinion of a user, who wants to reward an app or penalize it, by pointing out a nice feature or a serious problem. Hence, the context is very similar to what those sentiment tools are familiar with. Still, as observed, the tools' performance on the *neutral* category is very poor. Looking at the issue tracker data, besides the lack of neutral sentences in the JIRA dataset (which *per se* makes the life of the sentiment analysis tools much easier), again the predominance of problem-reporting sentences may (slightly) play in favour of such tools. Stack Overflow is a different beast. Posts mostly contain discussions on how to use a piece of technology, and between the lines somebody points out whether an API or a code pattern is good or less optimal. In many cases, without even expressing strong opinions. This definitely makes the applicability of sentiment analysis much more difficult.

Should we expect 100% accuracy from sentiment analysis tools? No, we should not. In our manual evaluation, out of the 1,500 Stack Overflow sentences we manually labeled, there were 279 cases of disagreement (18.6%). This means that even humans are not able to agree about the sentiment expressed in a given sentence. This is also in line with findings of Murgia *et al.* [21] on emotion mining: Except when a sentence expresses clear emotions of love, joy and sadness, even for humans it is hard to agree. Hence, it is hard to expect that an automated tool can do any better. Having said that, advances are still needed to make sentiment analysis tools usable in the software engineering domain.

Text reporting positive and negative sentiment is not sufficient to evaluate sentiment analysis tools. As discussed, the most difficult task for sentiment analysis tools is to discriminate between positive/negative *vs* neutral sentiment, while they are quite effective in discriminating between positive and negative sentiment. This is why datasets such as the JIRA one that we, and others, used in previous work [15, 16], is not sufficient to evaluate sentiment analysis tools. We hope that releasing our dataset [18] will help in more robust evaluations of sentiment analysis tools.

8 CONCLUSION

Some say that the road to hell is paved with good intentions. Our work started out with what we consider a promising idea: We wanted to develop an approach to automatically recommend APIs and libraries given a set of functional and non-functional requirements. To do so, we wanted to leverage the large body of knowledge that is stored in Q&A websites like Stack Overflow. The approach was going to exploit opinion mining using deep learning through recurrent neural network. However, as we finalized our work we noticed that it simply did not work, because the opinion mining component had unacceptable performance.

The reason for the failure is manifold. Firstly, it highlights how machine learning, even in its most advanced forms, is and remains a black box, and it is not completely clear what happens in that black box. To this one can add the design principle "*garbage in, garbage out*": No matter how advanced a technique, if the input is not appropriate, it is improbable that an acceptable output can be produced. In the specific case one might argue that Stack Overflow is not really the place where emotions run high: It is a place where developers discuss technicalities. Therefore it is rather obvious that opinion mining will have a hard time. While this might be true, our study revealed that also in datasets where emotions are more evident, like app reviews and issue trackers, there is an intrinsic problem with the accuracy of current state-of-the-art sentiment analysis tools.

In the end we decided to **write a "negative results" paper**. As Walter Tichy writes, "*Negative results, if trustworthy, are extremely important for narrowing down the search space. They eliminate useless hypotheses and thus reorient and speed up the search for better approaches*". We hope that the software engineering community can appreciate and leverage the insights that we obtained during our work. We are also releasing the complete dataset as a replication package. As a final word, we would like to stress that *we are not dismissing opinion mining in software engineering as impractical, but rather as not mature enough yet.* We believe there is promise in the field, but that a community effort is required to bring opinion mining to a level where it actually becomes useful and usable in practice.

ACKNOWLEDGMENTS

We gratefully acknowledge the financial support of the Swiss National Science Foundation for the projects PROBE (SNF Project No. 172799) and JITRA (SNF Project No. 172479), and CHOOSE for sponsoring our trip to the conference.

REFERENCES

[1] Stefano Baccianella, Andrea Esuli, and Fabrizio Sebastiani. 2010. SentiWordNet 3.0: An Enhanced Lexical Resource for Sentiment Analysis and Opinion Mining. In *Proceedings of LREC 2010 (International Conference on Language Resources and Evaluation)*.

[2] Fabio Calefato, Filippo Lanubile, and Nicole Novielli. 2017. EmoTxt: A Toolkit for Emotion Recognition from Text. In *Proceedings of ACII 2017 (7th International Conference on Affective Computing and Intelligent Interaction)*.

[3] L. V. G. Carreño and K. Winbladh. 2013. Analysis of user comments: an approach for software requirements evolution. In *Proceedings of ICSE 2013 (35th International Conference on Software Engineering)*. IEEE press, 582–591.

[4] Daviti Gachechiladze, Filippo Lanubile, Nicole Novielli, and Alexander Serebrenik. 2017. Anger and Its Direction in Collaborative Software Development. In *Proceedings of ICSE 2017 (39th IEEE/ACM International Conference on Software Engineering)*. IEEE, 11–14.

[5] David Garcia, Marcelo Serrano Zanetti, and Frank Schweitzer. 2013. The Role of Emotions in Contributors Activity: A Case Study on the GENTOO Community. In *Proceedings of CGC 2013 (3rd International Conference on Cloud and Green Computing) (CGC '13)*. 410–417.

[6] Michael Goul, Olivera Marjanovic, Susan Baxley, and Karen Vizecky. 2012. Managing the Enterprise Business Intelligence App Store: Sentiment Analysis Supported Requirements Engineering. In *Proceedings of HICSS 2012 (45th Hawaii International Conference on System Sciences)*. 4168–4177.

[7] Emitza Guzman, Rana Alkadhi, and Norbert Seyff. 2017. An exploratory study of Twitter messages about software applications. *Requirements Engineering* 22, 3 (2017), 387–412.

[8] Emitza Guzman, Omar Aly, and Bernd Bruegge. 2015. Retrieving Diverse Opinions from App Reviews. In *Proceedings of ESEM 2015 (9th ACM/IEEE International Symposium on Empirical Software Engineering and Measurement)*. IEEE, 21–30.

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

[10] Emitza Guzman and Bernd Bruegge. 2013. Towards emotional awareness in software development teams. In *Proceedings of ESEC/FSE 2013 (9th Joint Meeting on Foundations of Software Engineering)*. ACM, 671–674.

[11] Emitza Guzman and Walid Maalej. 2014. How do users like this feature? a fine grained sentiment analysis of app reviews. In *Proceedings of RE 2014 (22nd International Requirements Engineering Conference)*. IEEE, 153–162.

[12] Mark Hall, Eibe Frank, Geoffrey Holmes, Bernhard Pfahringer, Peter Reutemann, and Ian H. Witten. 2009. The WEKA Data Mining Software: An Update. *SIGKDD Explorations* 11, 1 (2009), 10–18.

[13] Minqing Hu and Bing Liu. 2004. Mining and summarizing customer reviews. In *Proceedings of KDD 2004 (10th ACM SIGKDD international conference on Knowledge discovery and data mining)*. 168–177.

[14] Clayton J Hutto and Eric Gilbert. [n. d.]. In *Proceedings of ICWSM 2014 (8th International AAAI Conference on Weblogs and Social Media*.

[15] Md Rakibul Islam and Minhaz F Zibran. 2017. Leveraging automated sentiment analysis in software engineering. In *Proceedings of MSR 2017 (14th International Conference on Mining Software Repositories)*. IEEE Press, 203–214.

[16] 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.

[17] Francisco Jurado and Pilar Rodriguez. 2015. Sentiment Analysis in monitoring software development processes: An exploratory case study on GitHub's project issues. *Journal of Systems and Software* 104 (2015), 82–89.

[18] Bin Lin, Fiorella Zampetti, Gabriele Bavota, Massimiliano Di Penta, Michele Lanza, and Rocco Oliveto. [n. d.]. Replication Package. https://sentiment-se.github.io/replication.zip. ([n. d.]).

[19] Christopher D. Manning, Mihai Surdeanu, John Bauer, Jenny Finkel, Steven J. Bethard, and David McClosky. 2014. The Stanford CoreNLP Natural Language Processing Toolkit. In *Association for Computational Linguistics (ACL) System Demonstrations*. 55–60.

[20] George A. Miller. 1995. WordNet: A Lexical Database for English. *Commun. ACM* 38, 11 (1995), 39–41.

[21] 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 MSR 2014 (11th Working Conference on Mining Software Repositories)*. ACM, 262–271.

[22] Nicole Novielli, Fabio Calefato, and Filippo Lanubile. 2014. Towards discovering the role of emotions in stack overflow. In *Proceedings of SSE 2014 (6th International Workshop on Social Software Engineering)*. ACM, 33–36.

[23] Nicole Novielli, Fabio Calefato, and Filippo Lanubile. 2015. The Challenges of Sentiment Detection in the Social Programmer Ecosystem. In *Proceedings of SSE 2015 (7th International Workshop on Social Software Engineering) (SSE 2015)*. 33–40.

[24] 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 MSR 2015 (12th Working Conference on Mining Software Repositories)*. IEEE Press, 303–313.

[25] 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 *Proceedings of MSR 2016 (13th International Conference on Mining Software Repositories)*. IEEE, 480–483.

[26] Bo Pang and Lillian Lee. 2008. Opinion Mining and Sentiment Analysis. *Foundations and Trends in Information Retrieval* 2 (2008), 1–135.

[27] Sebastiano Panichella, Andrea Di Sorbo, Emitza Guzman, Corrado A Visaggio, Gerardo Canfora, and Harald C Gall. 2015. How Can I Improve My App? Classifying User Reviews for Software Maintenance and Evolution. In *Proceedings of ICSME 2015 (31st International Conference on Software Maintenance and Evolution) (ICSME 2015)*. 281–290.

[28] Daniel Pletea, Bogdan Vasilescu, and Alexander Serebrenik. 2014. Security and emotion: sentiment analysis of security discussions on GitHub. In *Proceedings of MSR 2014 (11th Working Conference on Mining Software Repositories)*. ACM, 348–351.

[29] Mohammad Masudur Rahman, Chanchal K Roy, and Iman Keivanloo. 2015. Recommending insightful comments for source code using crowdsourced knowledge. In *Proceedings of SCAM 2015 (15th International Working Conference on Source Code Analysis and Manipulation)*. IEEE, 81–90.

[30] Athanasios-Ilias Rousinopoulos, Gregorio Robles, and Jesús M González-Barahona. 2014. Sentiment analysis of Free/Open Source developers: preliminary findings from a case study. In *Revista Eletronica de Sistemas de Informacao*, Vol. 13. 1–6.

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

[32] Richard Socher, Alex Perelygin, Jean Y Wu, Jason Chuang, Christopher D Manning, Andrew Y Ng, and Christopher Potts. 2013. Recursive deep models for semantic compositionality over a sentiment treebank. In *In Proceedings of EMNLP 2013 (2013 Conference on Empirical Methods in Natural Language Processing)*. Citeseer.

[33] Rodrigo Souza and Bruno Silva. 2017. Sentiment analysis of Travis CI builds. In *Proceedings of MSR 2017 (14th International Conference on Mining Software Repositories)*. IEEE Press, 459–462.

[34] 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.

[35] Parastou Tourani, Yujuan Jiang, and Bram Adams. 2014. Monitoring sentiment in open source mailing lists: exploratory study on the apache ecosystem. In *Proceedings of CASCON 2014 (24th Annual International Conference on Computer Science and Software Engineering)*. IBM Corp., 34–44.

[36] Gias Uddin and Foutse Khomh. 2017. *Mining API Aspects in API Reviews*. Technical Report. 10 pages. http://swat.polymtl.ca/data/opinionvalue-technical-report.pdf

[37] Lorenzo Villarroel, Gabriele Bavota, Barbara Russo, Rocco Oliveto, and Massimiliano Di Penta. 2016. Release planning of mobile apps based on user reviews. In *Proceedings of ICSE 2016 (38th International Conference on Software Engineering)*. 14–24.

[38] Yingying Zhang and Daqing Hou. 2013. Extracting Problematic API Features from Forum Discussions. In *Proceedings of ICPC 2013 (21st International Conference on Program Comprehension*. 141–151.