Achieving Reliable Sentiment Analysis in the Software Engineering Domain using BERT

Eeshita Biswas, Mehmet Efruz Karabulut, Lori Pollock, K. Vijay-Shanker
Computer and Information Sciences
University of Delaware
Newark, DE, United States
{biswas, mehmetef, pollock, vijay}@udel.edu

***Abstract*—Researchers have shown that sentiment analysis of software artifacts can potentially improve various software engineering tools, including API and library recommendation systems, code suggestion tools, and tools for improving communication among software developers. However, sentiment analysis techniques applied to software artifacts still have not yet yielded very high accuracy. Recent adaptations of sentiment analysis tools to the software domain have reported some improvements, but the f-measures for the positive and negative sentences still remain in the 0.4-0.64 range, which deters their practical usefulness for software engineering tools.**

In this paper, we explore the potential effectiveness of customizing BERT, a language representation model, which has recently achieved very good results on various Natural Language Processing tasks on English texts, for the task of sentiment analysis of software artifacts. We describe our application of BERT to analyzing sentiments of sentences in Stack Overflow posts and compare the impact of a BERT sentiment classifier to state-of-the-art sentiment analysis techniques when used on a domain-specific data set created from Stack Overflow posts. We also investigate how the performance of sentiment analysis changes when using a much (3 times) larger data set than previous studies. Our results show that the BERT classifier achieves reliable performance for sentiment analysis of software engineering texts. BERT combined with the larger data set achieves an overall f-measure of 0.87, with the f-measures for the negative and positive sentences reaching 0.91 and 0.78 respectively, a significant improvement over the state-of-the-art.

***Index Terms*—Sentiment Analysis, Software Engineering, BERT**

I. Introduction

Sentiment Analysis or opinion mining, is a Natural Language Processing (NLP) technique that automatically classifies whether the opinion/emotion expressed in a textual unit is positive, negative, or neutral. Researchers have been exploring sentiment analysis for various Software Engineering (SE) applications, such as detecting negative comments in Application Programming Interface (API) reviews to extract problematic API features [41], analyzing emotions in software developer Q&A forums such as Stack Overflow to recommend deficiencies or potential improvements of source code [34], and assessing the emotions in open source mailing lists to understand the sentiment status of a development team [37].

To support the use of sentiment analysis in SE, researchers have conducted studies of various off-the-shelf sentiment analysis tools in the SE domain. Jongeling et al. studied off-the-shelf sentiment analysis tools NLTK, SentiStrength, Stanford NLP, and Alchemy and reported their highest Adjusted Rand Index score of 0.21 [17], [18]. Recently, several studies investigated the accuracy of off-the-shelf sentiment analysis tools adapted to the SE domain by training on SE-related texts or by adding software-specific heuristic rules, dictionary, etc. [3], [6], [10], [14], [15], [22], [27]. While Imtiaz et al. [14] reported a Weighted Cohen's Kappa ranging from 0.16 to 0.33 agreement between manual annotations and sentiment analysis tools, Novielli et al. [27] and Lin et al. [22] reported f-measures ranging from 0.15 to 0.5 for the negative class, and 0.15 to 0.4 for the positive class on a Stack Overflow data set. On the same data set, our group's previous work (which we refer to as RNN4SentiSE in this paper) obtained f-measures of 0.54 and 0.41 for negative and positive sentences respectively by additional customization of a neural network-based sentiment classifier to the SE domain with the help of software-specific Word2Vec [24] word embeddings [4]. Chen et al. developed SEntiMoji [8], by emoji-powered customization and obtained f-measures of 0.64 and 0.47 for negative and positive sentences, respectively, on Lin et al.'s data set. Unfortunately, the effectiveness of these techniques still remains quite low for practical use in the SE domain.

In this paper, we describe our investigation into how much improvement can be made to sentiment analysis for the SE domain, by applying and adapting a language representation model called BERT (Bidirectional Encoder Representations from Transformers) developed by Devlin et al. [9]. BERT has been shown to advance the state-of-the-art in eleven NLP tasks (including sentiment analysis), often by significant margins. While we use a BERT model pre-trained on general corpus, the fine-tuning process will allow it to learn the software domain specific knowledge.

We also created a data set of 4000 manually annotated sentences from Stack Overflow posts to serve as an extension to Lin et al.'s software-specific data set of 1500 sentences from Stack Overflow [22]. We then compare the effectiveness of the BERT-based sentiment classifier, BERT4SentiSE, and RNN4SentiSE, using both the extended data set of 5500 sentences and Lin et al.'s data set of 1500 sentences. Our results indicate that BERT4SentiSE outperforms RNN4SentiSE by a substantial margin of over 20% improvement on the extended data set. Furthermore, BERT4SentiSE's performance is considerably better on the Lin et al. data set compared to RNN4SentiSE, and all tools evaluated by Chen et al. [8] and Lin et al [22], including SentiMoji.

This paper makes several contributions:

- To our knowledge, this work is the first to explore the potential of applying and adapting BERT sentence representations for sentiment analysis in the software engineering domain. BERT4SentiSE combined with a larger data set achieves an overall f-measure of 0.87, with the f-measures for the positive and negative sentences reaching 0.78 and 0.91 respectively, a significant improvement over the state-of-the-art, achieving significantly more reliability in sentiment analysis for the SE domain.
- We created a software-specific data set of 4000 sentences from Stack Overflow posts, manually labeled with sentiments to serve as an extension to Lin et al.'s [22] original data set of 1500 sentences for a combined set of 5500 annotated sentences.

II. State of the Art

As sentiment analysis is leveraged in various software development, maintenance, and evolution tasks, typically off-the-shelf sentiment analysis tools are used which are trained on non-SE texts [12], [13], [17], [28]–[32], [34], [35]. However, many studies have reported negative results when applying these off-the-shelf sentiment analysis tools on software-related texts [17], [18], [26], [37]. Novielli et al. [26] employed an off-the-shelf lexicon-based sentiment analysis tool, SentiStrength, to assess affective states in Stack Overflow and noted how the presence of software domain-specific lexicon increases false positives for the negative sentiment class. Jongeling et al. [17], [18] investigated off-the-shelf sentiment analysis tools: SentiStrength, NLTK, Stanford CoreNLP, and AlchemyAPI; their findings revealed that these tools do not agree well with manual sentiment labeling, with the highest Adjusted Rand Index score of 0.21 achieved by NLTK. Tourani et al. [37] assessed emotions in the open-source mailing lists of Apache software projects using SentiStrength and obtained precision of 0.13 and 0.296 for negative and positive classes, respectively.

Several off-the-shelf sentiment analysis tools have been adapted to the SE domain to hopefully improve effectiveness. Ahmed et al. developed SentiCR [3], a sentiment analysis tool for code review comments based on supervised learning algorithms. Islam and Zibran customized SentiStrength by adding heuristic rules, calling their tool SentiStrength-SE [15], which achieved improvement over SentiStrength when applied on JIRA issue comments. Islam and Zibran also developed DEVA [16], a dictionary-based lexical approach for detecting emotions (excitement, stress, depression, relaxation) in JIRA issue comments, but their findings revealed various limitations in handling complex structures of negations and subtle emotional expressions. Ding et al. developed SentiSW [10], an entity-level sentiment analysis tool consisting of sentiment classification and entity recognition, and used it to classify GitHub issue comments into (sentiment, entity) tuples. Although their highest overall accuracy was 0.77, their highest f-measure for the negative sentiment was 0.39.

Calefato et al. developed Senti4SD [6] by exploiting lexicon-based, keyword-based, and semantic features utilizing continuous bag-of-words embeddings. Senti4SD is trained on a gold set of Stack Overflow questions, answers, and comments with a balanced distribution of positive, negative, and neutral sentiment, and it reduced the number of neutral and positive posts misclassified as negative, compared to the baseline SentiStrength. Wrobel et al. [39] aimed at examining whether lexicon adaptation with a focus on the emotional intensity of words in the context of the SE domain improves the reliability of sentiment analysis. However, a comparative experiment of sentiment analysis based on generic and SE-specific lexicon did not show any significant difference in the results.

Lin et al. [22] set out to build a software library recommender that leverages developers' opinions mined from Stack Overflow (SO). On getting negative results while mining developer opinions, they investigated the accuracy of off-the-shelf sentiment analysis tools (SentiStrength, NLTK, and Stanford CoreNLP) and their adaptations to the SE domain (SentiStrength-SE and Stanford CoreNLP SO). They adapted Stanford CoreNLP SO to the SE domain by training on a data set that they built from Stack Overflow posts and manually labeled with sentiments. Novielli et al. [27] also carried out an investigation of SentiStrength, Senti4SD, SentiStrength-SE, and SentiCR using the same data set developed by Lin et al. [22]. These studies by Novielli et al. [27] and Lin et al. [22] reported "unacceptable accuracy levels in classifying positive/negative opinions" with f-measures ranging from 0.15-0.5 and 0.15-0.4 for the negative and positive classes, respectively. Imtiaz et al. [14] investigated sentiment analysis tools (SentiStrength, NLTK, Alchemy, Stanford CoreNLP, Senti4SD and SentiCR) on manually rated GitHub comments, and found these tools to have low agreement with human ratings on sentiment and politeness, with Weighted Cohen's Kappa ranging from 0.16 to 0.33 agreement between manual annotations and the sentiment analysis tools.

In our previous work [4], we investigated the effectiveness of two potential improvements to the training of sentiment analysis for SE artifacts when taking a neural network approach based on Word2Vec [24] word embeddings. The sentiment classifier RNN4SentiSE, was customized to the software domain by using software-specific word embeddings learned from Stack Overflow posts. However, this additional customization did not result in significant improvement. RNN4SentiSE based on generic word embeddings (learned from Google News data) performed either similar or better than RNN4SentiSE based on software-specific Stack Overflow word embeddings. We then investigated sampling techniques (over- and under-sampling) on the training data [7], [21], [25], [33]. Our results revealed that sampling the training data to address the skewed distribution of minority classes improved the classification of the minority classes with f-measures of 0.54 and 0.41 for negative and positive sentences respectively on Lin et al.'s data [22], with improvement in both precision

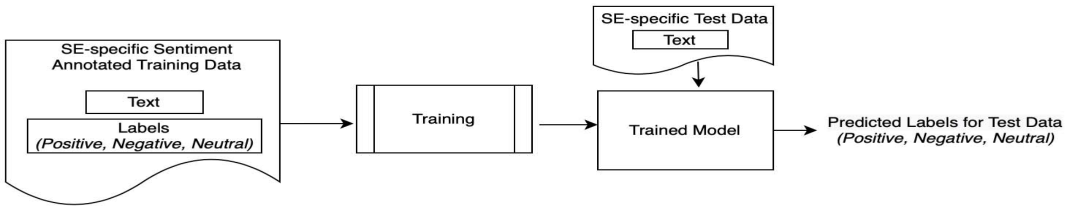
SE-specific Sentiment Annotated Training Data
Text
Labels (Positive, Negative, Neutral)
Training
SE-specific Test Data
Text
Trained Model
Predicted Labels for Test Data (Positive, Negative, Neutral)

Fig. 1. Key Components of a Sentiment Classifier

and recall.

Chen et al. developed SEntiMoji [8], an SE-customized sentiment classifier, built upon DeepMoji [11], an off-the-shelf approach based on emojis. SEntiMoji learns vector representations of texts based on how emojis are used alongside words on Twitter and GitHub, which are then used to predict sentiments. Their results show that SEntiMoji outperforms existing methods on Lin et al.'s data set [22], achieving f-measures of 0.64 and 0.47 for negative and positive sentences, respectively.

In summary, there have been many varied approaches towards improving the effectiveness of sentiment classification for the SE domain. The evaluation results indicate that domain adaptation of off-the-shelf sentiment analysis tools to the SE domain does indeed improve their performance; however, additional work is needed towards increasing their effectiveness beyond the 0.4-0.65 f-measure range for reliable sentiment analysis for SE applications.

III. Classifying Sentiment In The SE Domain Using Machine Learning

Before describing how we apply BERT (a neural network-based approach) for the sentiment analysis task, we begin here with an overview of the construction and application of a sentiment classifier, specifically for the SE domain. Figure 1 depicts the key components of a sentiment classifier based on machine learning.

The training data contains text and the corresponding sentiment (e.g. positive, negative, or neutral) for each unit of text, such as a sentence. Depending on the source of the text used for training, we may need to perform some pre-processing such as removal of URLs or code segments to remove noise from the text. As noted in related work, research has shown that providing software-related text can improve the effectiveness of the sentiment classifier.

During training, the text from the training data is usually converted to feature vectors, which are then fed to the machine learning component such as a neural network that generates a trained sentiment classifier model. This model learns to associate a particular input (text) to its corresponding sentiment based on the training data. For classification, the test data, which is SE-specific text, is fed into the trained classifier/model, which predicts sentiment labels for each text unit, typically each sentence.

IV. Introducing BERT

This section presents a brief description of BERT, including the key insights leading to its design, why it is so effective for various NLP tasks in general, and how it works.

A. What Is BERT?

BERT (Bidirectional Encoder Representations from Transformers) is a language representation model developed by Google in 2018 (Devlin et al. [9]). Since then, BERT has been considered one of the most prominent breakthroughs in the field of NLP and has achieved state-of-the-art results in a variety of tasks including question-answering, sentence tagging, textual entailment, reading comprehension, extractive summarization tasks and learning task-independent sentence representations [2], [9], [23]. This success can be attributed to the value of transfer learning and fine-tuning of a pre-trained model's parameters on a task-specific annotated data set and the use of the transformer, an elaborate attention-based architecture.

Through transfer learning, BERT helps mitigate the problem of shortage of task-specific labeled data, by pre-training a task-neutral language representation model using a huge corpus of unlabeled data. The pre-trained model can then be fine-tuned for specific tasks such as sentiment analysis using task-specific labeled data. This approach advanced the state-of-the-art in eleven NLP tasks, often by significant margins, without requiring task-specific neural architectures. This includes the sentiment analysis task, where BERT achieved a high GLUE score of 94.9 on the Stanford Sentiment Treebank data set [36], a benchmark data set for sentiment analysis.

B. Unique Attributes of BERT

BERT has some features that differentiate it from other language representations. Context-free word-embedding models such as Word2Vec and GloVe generate a single representation for each word in the vocabulary, despite the fact that words can have different meanings in different contexts. BERT provides a contextualized representation, generating representations of words based on their context.

Another differentiating feature of BERT is in the way it creates a bidirectional representation. Typically, with the use of recurrent neural networks, such as Long Short Term Memory (LSTM), text is processed in a single direction, typically left to right. Since such models do not see the future context at the time of predictions, they are augmented by a second

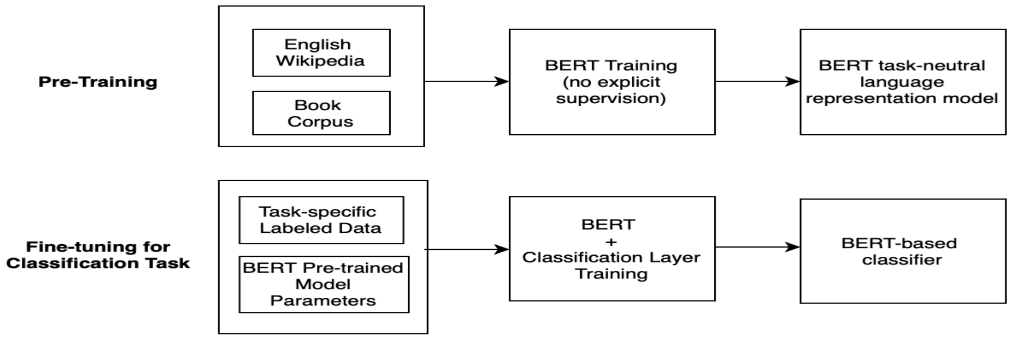
Pre-Training
English Wikipedia
Book Corpus
BERT Training (no explicit supervision)
BERT task-neutral language representation model
Fine-tuning for Classification Task
Task-specific Labeled Data
BERT Pre-trained Model Parameters
BERT + Classification Layer Training
BERT-based classifier

Fig. 2. BERT's Pre-Training and Fine-Tuning for Classification Tasks

recurrent network that processes text in the reverse direction. Together the use of the two RNNs produces a bidirectional representation of the input text. In contrast, BERT captures contextual information from left and right simultaneously to create a unique bidirectional representation.

Finally, we note that BERT is based on the transformer architecture introduced in [38]. It replaces recurrent units such as LSTM and Gated Recurrent Units (GRU) and instead uses multi-head attention and feed-forward components. Using self-attention components, BERT relates words in different positions to compute its contextualized representations.

C. *Two Phases of BERT Training*

As shown in Figure 2, the training of BERT is divided mainly into two parts:

1) Pre-training: Pre-training a BERT model is the first step of using BERT. BERT is pre-trained on a large unlabeled corpus that consists of text from English Wikipedia (2,500 million words) and Book Corpus (800 million words), allowing it to capture extensive knowledge about the language. The result of pre-training is a task-neutral language representation which can then be fine-tuned for specific downstream NLP tasks with minimal architectural modifications.

2) Fine-tuning: Akin to transfer learning, fine-tuning is a crucial second step in the use of BERT for specific tasks. While the pre-training produces a deep bidirectional unsupervised language representation, fine-tuning allows this representation to be used in learning to solve a new NLP task. For fine-tuning, the BERT model is first initialized with the pre-trained parameters, and these parameters are then adjusted using labeled data from the downstream NLP task. For classification tasks (such as sentiment classification in our case), fine-tuning occurs through $\mathbf{T}_{[CLS]}$, which is the representation of [CLS] in the output layer. [CLS] is a special classification token added at the beginning of each input sequence in BERT training[1]. $\mathbf{T}_{[CLS]}$ is taken as the aggregated representation of the input sequence; it allows the context information learned from other tokens in the transformer's self-attention layers to be represented in the $\mathbf{T}_{[CLS]}$ vector.

[1]Another token, [SEP], is used to separate different parts of the text and is used for tasks such as question-answering.

V. Using BERT For Sentiment Analysis Of SE Text

This section describes the main customization to create our sentiment classifier BERT4SentiSE for the SE domain by incorporating the software knowledge implicit in Stack Overflow's SE-specific text. To provide an overview of how we fine-tune BERT for the NLP task of sentiment analysis for the SE domain, we modified Devlin et al.'s figure called *Illustrations of Fine-tuning BERT on Different Tasks* [9] to create Figure 3. Specifically, we needed to choose a pre-trained BERT model, provide SE-specific input data, and fine-tune the model for sentiment classification by providing a sentiment classification layer.

Pre-trained BERT Model: We downloaded the pre-trained BERT base model[2] to use in BERT4SentiSE. BERT base consists of 12 layers, hidden state size (dimension of the encoder layers) of 768, and all together is comprised of 110 million parameters. This BERT model comes with pre-trained model parameters (weights) trained on English Wikipedia and Book Corpus for 1M steps, and TensorFlow [1] code for the BERT model implementation, which we modified for fine-tuning the model parameters for our sentiment analysis task on SE-specific sentiment annotated data.

Input to BERT Model: BERT's architecture allows flexible input and output representations to accomplish various NLP tasks. Input to a BERT model can be a single sentence or a sentence pair with two BERT special tokens [CLS] and [SEP] to indicate the start and end of the sentences (details in section IV-C2). The input sentence/s are tokenized into WordPiece tokens [40] by the BERT base implementation package. WordPiece tokens are subwords that are frequent/likely combinations of characters. The primary reason for using WordPiece is to tackle the out-of-vocabulary (OOV) problem. By using common subwords obtained from a large corpus, no special treatment is needed even if the input sentence contains unknown/unseen words. BERT accepts a sequence of up to 512 tokens (sub-words, punctuation).

For our sentiment analysis task, we represent the input sequence as a single sentence from our SE-specific sentiment annotated data sets. The input sentence is first tokenized into

[2]http://goo.gl/language/bert

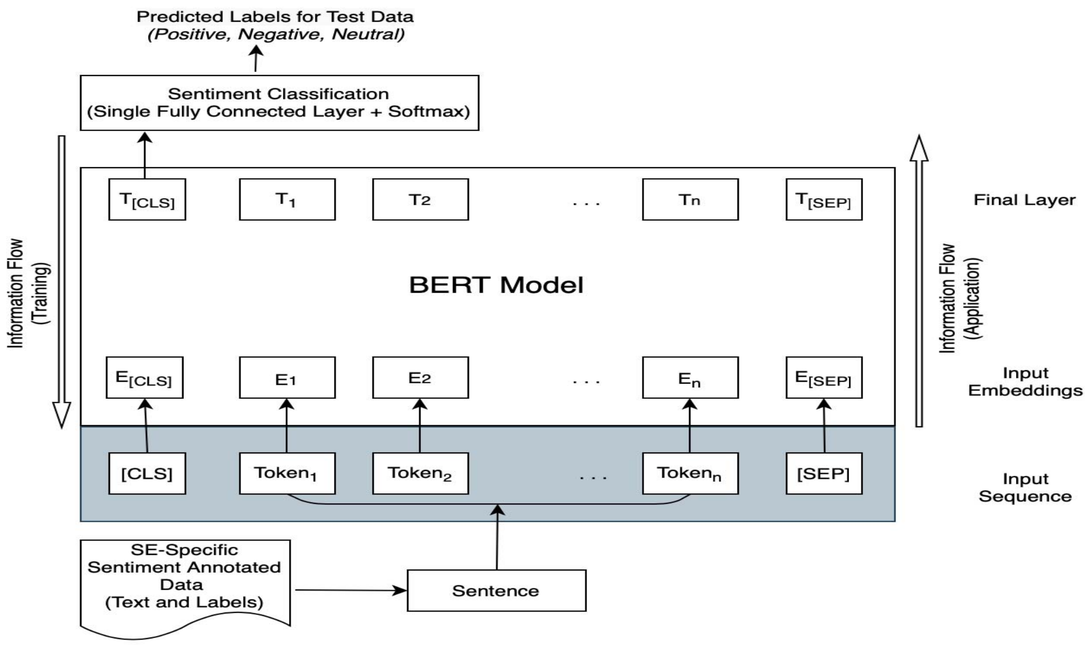
Predicted Labels for Test Data
(Positive, Negative, Neutral)
Sentiment Classification
(Single Fully Connected Layer + Softmax)
Information Flow (Training)
T[CLS]
T1
T2
...
Tn
T[SEP]
Final Layer
BERT Model
Information Flow (Application)
E[CLS]
E1
E2
...
En
E[SEP]
Input Embeddings
[CLS]
Token1
Token2
...
Tokenn
[SEP]
Input Sequence
SE-Specific Sentiment Annotated Data (Text and Labels)
Sentence

Fig. 3. Sentiment Classification using BERT

WordPiece tokens, and as required in BERT, the two tokens [CLS] and [SEP] are inserted at the beginning and end of the sentence, respectively. With BERT base, each token will have 12 intermediate representations. In Figure 3, E_i represents the input embedding for token i. Each input embedding, E_i, is the sum of the token, position, and segment embedding for token i. Token embedding is the WordPiece embedding for a specific token. Position embedding is used to indicate the position of the token in the sequence, since the transformer does not capture this information like in RNNs. We do not calculate segment embedding as our input includes single sentences. Throughout the 12 layers, attention computation is performed on the representation of the previous layer to create a new intermediate representation. In the final layer, BERT outputs a contextual representation for each token; T_i represents the contextual representation of token i.

Fine-Tuning for Sentiment Classification: To fine-tune for the sentiment analysis task, we use the Tensorflow library [1] to initialize the model with the pre-trained parameters. Then, we add task-specific modifications in the implementation code. We use the Adam Optimization Algorithm [19] as was used in BERT pre-training. Recall from section IV-C2, the token $\mathbf{T}_{[CLS]}$ is the aggregated representation of the input sequence. We add a fully connected layer with a softmax activation function (classification layer) to the $\mathbf{T}_{[CLS]}$ vector, to learn a mapping from sentence representation to the correct sentiment label. The classification layer has a dimension of K x H, where K is the number of sentiment labels (3) and H is the size of the hidden state (768). All the parameters in the network are fine-tuned in an end-to-end fashion to maximize the log-probability of the correct sentiment label, which is then passed to the softmax activation function. Finally, the softmax activation function outputs a probability for the three sentiment classes: positive, negative and neutral.

We have released the modified code[3] for replication purposes.

VI. Increasing Sentiment Analysis Data For SE Domain

In the SE domain, data sets for supervised learning are typically relatively small because they involve tedious manual labeling that requires familiarity with the SE domain and knowledge of technical terms. However, neural networks typically perform best when a large amount of data is available for training. It is possible that a main cause of the low performance of neural network-based sentiment classifiers for SE is the lack of larger training sets.

Towards addressing this potential source of poor performance, as part of our research reported in this paper, we also release a data set of 4,000 sentences from Stack Overflow[3], manually annotated with sentiment. The goal is to significantly extend the Lin et al. data set [22] which we call LinData for the rest of this paper. From the Stack Exchange archive, we downloaded the Stack Overflow data dump dated January 2019 - June 2019 to avoid duplicating sentences in the Lin et al. data set, and used only the Stack Overflow posts (questions and answers) for building the data set. Stack Overflow posts contain elements such as code segments, urls, and HTML tags, which do not convey information about the sentiment of the text. Removing such elements is common pre-processing that

[3]https://www.dropbox.com/sh/0dzw55rqo7e6k2g/AACoeQFRyx-VBoy0tC1EBCwJa?dl=0

TABLE I
SAMPLE GUIDELINES DERIVED FROM LINDATA FOR SENTIMENT ANNOTATION

Negative Sentiment

The given sentence indicates:

a. An Error, Exception Message, Error Description
Example: So, everything builds fine, but when we try to deploy the application to GFNUMBER we get the FILE_NAME file not found "error.

b. A Problem/Failure
Example: But sadly this is not working.

c. A Warning, Discontent, Complaint, Disappointment
Example: else your GUI will be Hanged.

Positive Sentiment

The given sentence refers to any good/useful qualities of code/tool/ features, etc, such as:

a. Providing support for certain features.
Example: Javolution Structs supports mapping of off heap memory as a data structure.

b. Providing advantages over others.
Example: Databases are much better at handling data than Java.

c. Serving as a useful or good example.
Example: There is a good example showing how to put a file onto WebDAV server.

d. Availability of certain features.
Example: APKInspector provides both analysis functions and graphic features for the users to gain deep insight into the malicious apps.

makes sentiment analysis more effective. We pre-processed the Stack Overflow posts using Python code to remove code segments, urls, and HTML tags from the body of the posts to focus only on English sentences, and finally split each of the posts into individual sentences. We then randomly selected 4000 sentences, and one of the authors manually labeled each sentence by assigning a sentiment (Positive, Negative, or Neutral) to the whole sentence. To reduce threat to validity, other authors confirmed a sample of these labels.

Here, we describe the process we used for developing and refining the guidelines and the manual labeling using the guidelines. Since our goal was to supplement the LinData, we wanted to assign sentiment similar to theirs, but at the sentence level. We could not follow the exact sentiment annotation process as Lin et al. since their approach (based on Recursive Neural Network) required them to assign a sentiment not just to the whole sentence but to all intermediate nodes (each word and phrase) composing the sentence. Instead, to assign sentiments in a similar way, we thoroughly studied the sentences in LinData and their respective assigned sentiments, and reverse engineered a set of guidelines for manual sentiment labeling to result in the same sentiments as much as possible. Our guidelines, which are part of our public release[3], specifically contain instructions about what type of sentences are indicative of negative or positive sentiment (with respect to the software domain). Table I shows sample guidelines for negative and positive sentences.

TABLE II
DATASETS WITH DISTRIBUTION OF SENTIMENT/POLARITY CLASSES

Dataset	Total Sentences	Sentiment Classes and Distribution					
		Negative		Positive		Neutral	
LinData	1500	178	11.87%	131	8.73%	1191	0.794
NewSESentiData	4000	1119	27.98%	576	14.40%	2305	0.5763
Total (CombinedData)	**5500**	**1297**	**23.58%**	**707**	**12.86%**	**3496**	**63.56%**

To verify our guidelines before manually creating a new labeled data set using the guidelines, we asked four doctoral students/annotators from the Computer Science department to apply these guidelines to a subset of LinData. This subset consisted of 30 sentences in total, 10 sentences from each of the positive, negative, and neutral category. After the 30 sentences were manually labeled by the four annotators, we discussed and analyzed all disagreements (i.e., cases in which the annotators assigned a different sentiment compared to the sentiment assigned by Lin et al.). Based on their reason for disagreement, we refined the guidelines accordingly and one of the authors used the refined guidelines to manually annotate a second subset (different from the above subset of 30 sentences) of LinData. This subset consisted of 250 sentences, which has been regarded as a sufficient sample size to compute agreement measures with high confidence. [5]. We computed Cohen's Kappa inter-rater agreement between the annotations of our author (based on the refined guidelines) and Lin et al., observing an agreement of 0.88, which is greater than the 0.6 that is considered to be sufficient [20].

Finally, one of the authors used the refined guidelines to manually label the data set of 4000 sentences, hereby referred to as the NewSESentiData. As shown in Table II, NewSESentiData consists of 1119 negative sentences, 576 positive sentences, and 2305 neutral sentences. For our experiments, we also combine the LinData and NewSESentiData sets to become the CombinedData set of 5500 sentiment-labeled Stack Overflow sentences.

VII. EVALUATION STUDY

The goal of our evaluation was to study how much BERT, which has been very successful on various NLP tasks on English text, can push the effectiveness of sentiment analysis in the SE domain, thus increasing the reliability of SE tools that depend on sentiment analysis.

A. RESEARCH QUESTIONS

Towards our evaluation goal, we designed our study to answer the following research questions:

- RQ1: How does BERT4SentiSE compare to existing techniques used for sentiment analysis of software-related texts?
- RQ2: What is the impact of a larger data set on sentiment analyzers for SE?
  - How does the larger data set affect results of existing sentiment analysis for SE?

– What role does the larger data set play in BERT4SentiSE's effectiveness?

B. *COMPARISON SENTIMENT CLASSIFIER*

We compared BERT4SentiSE with RNN4SentiSE, the Recurrent Neural Network (RNN) based sentiment classifier, from our group's previous work [4]. RNN4SentiSE provided one of the leading results previously obtained, including on the minority classes which has been the bane of previous systems, and RNN4SentiSE's code followed the 10-fold cross validation procedure and sampling techniques to address class imbalance, similar to our planned experiments. RNN4SentiSE uses Long Short-Term Memory (LSTM) for its RNN units.

Hyper-parameter Configurations: We ran experiments on BERT4SentiSE and RNN4SentiSE using various configurations for each. For BERT4SentiSE, we used the following hyper-parameter values: learning rates [**2e-5**, 5e-5], batch size of 16, and epochs [2, **3, 4, 5**], different epoch produced best results for the different data sets. In these listings, the best results were obtained for the hyper-parameters shown as the bolded values. The maximum sequence length was fixed to 256, and the dropout rate was kept at 0.1 across all the experiments. For RNN4SentiSE, we used the settings that had obtained the best results in our previous work. The Google News word embeddings by Mikolov et al. [24] were used as the word embedding model. The hyper-parameter configurations used were LSTM unit 30, batch size of 30, and a dropout rate of 0.2. For epochs, we tested the following values [100, 200] for each set of experiments and reported the best results obtained.

C. *EVALUATION METRICS*

We report the performance of the sentiment classifiers in terms of the overall f-measure (macro-averaged) over the three sentiment categories as well as the precision, recall, and f-measure for each individual sentiment/polarity category (positive, negative and neutral). The choice of these metrics is in accordance to the standard practices adopted for sentiment analysis tools. Precision for a category/class is the ratio of the number of sentences correctly labeled as belonging to a class (true positives) over the number of sentences labeled as belonging to that class (true positives and false positives). Recall for a class is the ratio of the number of sentences correctly labeled as belonging to a class (true positives) out of the total number of sentences in the class (true positives and false negatives). F-measure for a class is the harmonic mean of precision and recall for that particular class.

We use f-measure to represent the overall performance instead of accuracy because the unbalanced distribution of the three sentiments in our data sets would cause the overall accuracy to mostly reflect the performance on the majority class, neutral. In particular, we use the macro-averaged f-measure value. For computing macro-averaged f-measure, we first compute the precision and recall for each individual class, and then we compute the overall precision and recall values by computing the mean of the scores over the three sentiment categories. Thus, all three classes are weighted equally. We do not use micro-averaging, where the number of instances of each class are used in the computation, because the results would be influenced heavily by the majority sentiment class, neutral.

TABLE III
RESULTS OF THE SENTIMENT CLASSIFIERS TRAINED AND TESTED ON COMBINEDDATA

Sentiment Classifier	Training Data Sampling	F-Measure (Overall)
BERT4SentiSE	None	**0.87**
	Balanced	0.84
RNN4SentiSE	None	0.69
	Balanced	0.66
	SR_Biswas	0.61

SAMPLING RATES: None - No Sampling, Balanced - Undersamples Neutral to 40%, Neg, Pos stay same), SR_Biswas - Sampling rate reported best by Biswas et al. [2] (over-samples Neg by 7 and Pos by 9 times, Neutral stays same).

D. *PROCEDURE*

In accordance with the studies conducted by Lin et al. [22] and our previous work [4], we applied 10-fold cross-validation for training and testing of the sentiment classifiers. We split the data set into 10 subsets while maintaining the original distribution of sentences from each of the three sentiment categories. For each fold, we use one unique subset as the test set and the remaining nine subsets as the training set.

Training Data Sampling: Recall from Table II, both Lin-Data and CombinedData have unbalanced class distribution. In addition to reporting our results with no sampling, we also show the results (in Table III) obtained by sampling to get a roughly balanced distribution of all the three classes. This is done by under-sampling the majority neutral class to about 40% of the total neutral sentences, reducing the skew from a 2:1:5 to a 2:1:2 class distribution (negative:positive:neutral). We had also reported experiments with oversampling of the minority classes in our previous work [4]. For oversampling, the sentences in the minority classes (negative and positive) are increased in the training set by duplication. So for comparison reasons, we also show the results on the CombinedData for the RNN4SentiSE model using the sampling rate that gave the best results in our previous work.

E. *RESULTS*

The cost of BERT4SentiSE is substantially less than RNN4SentiSE. In our study, BERT4SentiSE took about 1 hour for fine-tuning and testing as compared to 3-4 hours taken by RNN4SentiSE for training and testing.

RQ1 - How does BERT4SentiSE compare to existing techniques used for sentiment analysis of software-related texts?

To answer RQ1, we ran BERT4SentiSE and RNN4SentiSE using CombinedData for both training and testing of the sentiment classifiers. Table III reports our results using no

TABLE IV
PRECISION, RECALL AND F-MEASURE FOR THE INDIVIDUAL SENTIMENT CLASSES (NO SAMPLING) ON COMBINEDDATA

Sentiment Classifier	**Negative**			**Positive**			**Neutral**		
	P	R	F	P	R	F	P	R	F
BERT4SentiSE	**0.91**	**0.9**	0.91	**0.77**	**0.79**	0.78	**0.93**	**0.92**	0.93
RNN4SentiSE	0.72	0.72	0.72	0.55	0.47	0.51	0.84	0.86	0.85

sampling (None), with sampling to achieve balanced distribution (Balanced). In addition, we also report the results for RNN4SentiSE using the sampling rate (SR_Biswas) that gave the best results for this model in [4].

BERT4SentiSE (No Sampling) achieves an overall f-measure of 0.87, a substantial 18% improvement over the RNN4SentiSE regardless of the sampling rates employed with the latter model.

The f-measures from RNN4SentiSE indicate that the over- and under- sampling techniques do not provide any improvement in the results, in contrast to the results in our previous work [4]. Instead, we observe a drop in the results for both Balanced and SR_Biswas sampled training sets while using RNN4SentiSE. Similar results are observed for BERT4SentiSE with Balanced sampled training set. It is possible that the ineffectiveness of sampling is due to the increased size of the CombinedData data set as compared to LinData, and the substantial increase in negative and positive sentiment sentences. In contrast to LinData's 178 negative and 131 positive sentences, CombinedData is comprised of 1297 negative and 707 positive sentences. Although BERT4SentiSE's f-measure drops 3% when using Balanced training set as compared to no sampled training set, it retains the same 18% increase over RNN4SentiSE using the Balanced sampling rate.

To delve more deeply into which sentiment classes are causing the overall improvement in performance, Table IV shows further breakdown of the scores for the three individual sentiment classes. In all previous work, all sentiment analysis tools struggled to classify negative and positive sentences correctly while performing better for neutral sentences. Based on the breakout among negative, positive and neutral classes, both cases of 18% improvement in overall f-measure (from Table III) can be attributed to the improvement in the two minority classes, negative and positive, and not by correct predictions of the majority class, neutral. BERT4SentiSE's f-measure (with no sampling) in detecting the negative and positive sentiments is 0.91 and 0.78, respectively, which shows that the majority of the sentences with negative and positive sentiment are identified correctly. In fact, BERT4SentiSE better classifies both the two minority sentiment classes, negative and positive, with increases in f-measure values by 19% and 27%, respectively, over RNN4SentiSE while still achieving an 8% increase for the neutral class.

For both precision and recall in the negative and neutral classes, BERT4SentiSE achieves greater than 0.9. Thus, not only does BERT4SentiSE recall 90% of negative sentences correctly but also less than 10% of the sentences classified as negative are incorrect (precision of 0.91), and most of these errors are due to negative sentences misclassified as neutral and vice-versa (see Table V). Another noteworthy achievement of BERT4SentiSE is the huge improvement over RNN4SentiSE in the positive class precision and recall, improvements of 22% and 32%, respectively. As mentioned in earlier works, identifying positive sentences correctly is the most difficult since positive sentiment texts are rare in online Q&A sites such as Stack Overflow. Even in CombinedData, less than 13% of all sentences are positive. Even with such a low distribution of positive sentiment sentences, BERT4SentiSE classifies 558 sentences correctly out of a total of 707 (Table V), reaching a recall of almost 0.8 (Table IV). This high recall shows the credibility of BERT4SentiSE in correctly identifying positive sentiment in texts. The 32% improvement in positive recall over RNN4SentiSE and overall high precision and recall for both negative and positive sentences highlights the improved performance of BERT4SentiSE on the two minority classes that appear less often in Stack Overflow posts, thus further supporting its application as a reliable tool for sentiment analysis in the SE domain.

To analyze misclassifications, Table V contains the confusion matrix obtained by the BERT4SentiSE on CombinedData. The matrix shows that most errors are due to negative or positive sentences misclassified as neutral and vice-versa. While differentiating between negative and positive sentiments, BERT4SentiSE only misclassified 12 negative sentences as positive, and 8 positive sentences as negative.

Examples of sentiment misclassifications are shown in Table VI. Most of these misclassifications can be attributed to the fact that most of these sentences exhibit the presence of both kinds of sentiments. For example, the sentence "*I have a recursive function that seems to be working properly up until I try and return from it.*" starts on a positive note, but later ends negatively. This also causes misclassifications of negative and positive sentences as neutral. Other reasons for misclassifications of negative or positive sentences as neutral are:

- The negative/positive sentiment present in the sentences are not explicit. (e.g., "*I wanted to see all the files I have in my external storage, I have this library that display the text to the user, but when I'm using it to show the sub files, it says something like.*")
- They can be viewed as either a question or a mere statement (e.g., "*Any idea what i might be doing wrong.*").

TABLE V
CONFUSION MATRIX FOR BERT4SENTISE (NO SAMPLING) ON COMBINEDDATA

		Predicted		
		Neg	***Pos***	***Neu***
Actual	***Neg***	1171	12	114
	Pos	8	558	141
	Neu	109	159	3228

RQ2 - What is the impact of a larger data set on sentiment analyzers for SE? How does the larger data set affect results of existing sentiment analysis for SE? What role does the larger data set play in BERT4SentiSE's effectiveness?

To investigate how the increase in the size of the training set affects the performance of the sentiment classifiers, we performed a set of experiments where we train two versions of each classifier, one trained only on LinData, another trained on CombinedData. We test both the versions of the classifiers on LinData as well as CombinedData. To get the best possible results while training on each data set, we conducted experiments with no sampling as well as various combinations of over- and under-sampling. We report only the best results obtained for each training data in Table VII, as our sole purpose of this study is to compare the results when the size of the labeled data used for training is increased. Using BERT4SentiSE, we ran ten-fold cross-validation experiments using CombinedData as the test data, while training on both LinData and CombinedData individually, with and without sampling. We performed similar experiments using LinData as the test data. We also performed similar experiments using RNN4SentiSE, but for the results of RNN4SentiSE trained and tested on LinData, we use the best scores reported in our previous work [4] by converting the reported precision and recall for each sentiment to their corresponding f-measures.

The results in Table VII indicate that the increase in the size of the training data provides improvement for both BERT4SentiSE and RNN4SentiSE especially when we test on the larger data set, CombinedData (highlighted in bold). For BERT4SentiSE tested on CombinedData, overall f-measure changes from 0.8 to 0.87, a 7% improvement. For RNN4SentiSE tested on CombinedData, f-measure increases from 0.58 to 0.69, and improvement of 11%. This improvement might be considered significant especially when we look at the performance breakdown for each of the three sentiment classes. With BERT4SentiSE, the positive f-measure increases by 14%, while still making small improvements of 6% for negative and 3% for neutral classes. In contrast, RNN4SentiSE makes quite a significant improvement for both the minority classes. While testing RNN4SentiSE on CombinedData, negative f-measure increases by 17% and the positive f-measure increases by 14%, while still making a 5% improvement for the neutral class.

We note that the improvements by training on the larger data set, CombinedData are not as significant when we test using only the smaller data set, LinData. For BERT4SentiSE, the overall improvement is about 3% and for RNN4SentiSE, just about 2%. However, our BERT-based classifier BERT4SentiSE still provides reliable results when tested on the smaller data set LinData, achieving f-measures of 0.81 and 0.64 for the negative and positive class, respectively, as a result of better classification of these minority class sentences.

Overall, these improvements show that increasing the size of the training data from 1500 to 5500 did indeed help in improving the sensitivity of both the sentiment classifiers. Thus, the results support our assumption that sentiment analysis of SE texts can continue to improve further by increasing the size of the software-related training data.

VIII. DISCUSSION AND LESSONS LEARNED

This section summarize the lessons learned from our study.

Lesson: Achieving reliable sentiment analysis in the software engineering (SE) domain is possible using BERT. Our results demonstrate that BERT (fine-tuned with SE-specific sentiment annotated data) enables reliable sentiment analysis. The BERT-based classifier, BERT4SentiSE, obtains high precision and recall, even for the minority classes of positive and negative, which have caused problems for previous sentiment analysis tools in the software engineering domain. In terms of performance, we see that BERT4SentiSE shows a significant improvement over an existing leading system, RNN4SentiSE. RNN4SentiSE recalls less than half of the positive sentences, and almost half of the sentences classified as positive are incorrect. In contrast, BERT4SentiSE recalls almost 80% of the positive sentences, and only a little over 20% of the sentences classified as positive are incorrect. When trained and tested on the original smaller data set LinData, we see that BERT4SentiSE significantly improves on the best results of previous studies [4], [8], [22], [27], especially on the minority classes. Thus, we can conclude that BERT4SentiSE can successfully identify the sentiment of SE-related texts and enable its practical usage as a sentiment analysis tool in the SE domain.

Lesson: Increasing the size of the data set indeed helps in improving the performance of BERT-based and existing sentiment classifiers. The performance difference between training on the smaller data set (LinData) and training on the larger data set (CombinedData) is significant when tested on CombinedData, especially for the minority sentiment classes, as evident from Table VII. The positive f-measure shows a huge improvement of 14% when training BERT4SentiSE on CombinedData. When training RNN4SentiSE on CombinedData, both the negative and positive f-measures improve by a huge margin of 17% and 14%, respectively. Although we do not observe such huge improvements when tested on only LinData, we still see some improvement. The results imply that the larger data set improve the effectiveness of BERT-based and existing sentiment classifiers.

The results will obviously vary based on the specific data, basically the quality of the data and how representative the data is of all kinds of possible communications in the specific domain. Also, we need to recall the class imbalance of the data

TABLE VI
EXAMPLES OF MISCLASSIFICATIONS OF NEGATIVE AND POSITIVE SENTENCES BY BERT4SENTISE

Negative Sentences Misclassified	Positive Sentences Misclassified
As Positive *It would really help me debug as I'm facing some strange issues.* *I have a recursive function that seems to be working properly up until I try and return from it.* *It's ugly and inefficient but it should work.* *But no Json response nor error is being returned.* *So then your migrated project doesn't build and you have lots of work to fix it.*	**As Negative** *The documentation was unclear there – I've fixed it.* *But this gets optimized by the JIT and there is no measurable impact in production.* *I still think it's not the best design, but I did quite a bit of testing and can say with a good bit of confidence that it had no real impact on performance.* *Not sure if this is what you're looking for but the following code catches and identifies specific add errors.*
As Neutral *Any idea what i might be doing wrong.* *As of now, I can download all csv files but I can't limit it to only today's date.* *I wanted to see all the files I have in my external storage, I have this library that display the text to the user, but when I'm using it to show the sub files, it says something like.*	**As Neutral** *If you are looking for a way to write to streams in a more intuitive way, try CODE_FRAGMENT.* *With the new design, they could also support other encodings in future.* *Javolution Structs supports mapping of off heap memory as a data structure.* *So no, it is not a bug, simply a design choice.*

TABLE VII
RESULTS FOR TESTING THE IMPACT OF THE INCREASE IN SIZE OF THE DATA SETS

Sentiment Classifier	Training Data	Test Data	F-Measure			
			Negative	Positive	Neutral	Overall
BERT4SentiSE	LinData	*CombinedData*	0.85	0.64	0.9	0.8
	CombinedData		**0.91**	**0.78**	**0.93**	**0.87**
	LinData	*LinData*	0.76	0.61	0.92	0.77
	CombinedData		0.81	0.64	0.93	0.8
RNN4SentiSE	LinData	*CombinedData*	0.55	0.37	0.8	0.58
	CombinedData		**0.72**	**0.51**	**0.85**	**0.69**
	LinData	*LinData*	0.54	0.41	0.89	0.61
	CombinedData		0.55	0.46	0.87	0.63

sets used in our study. Even though we increase the size of the data set from 1500 to 5500, the distribution of the negative and positive sentiment classes did not improve significantly. Further improvement in the distribution of the minority classes might provide additional improvement in the sensitivity of the sentiment classifiers.

IX. THREATS TO VALIDITY

Threats to Internal Validity are related to the configuration of the sentiment classifiers. To limit this threat, we tested various parameters for both sentiment classifiers. However, there exists possibility of further tuning these parameters to improve the performance of the classifiers. Other potential threats could be related to any errors in our scripts, which we have tried to eliminate by performing multiple code reviews and tests.

Threats to External Validity correspond to the generalizability of our experiments and findings. Sentiment annotation can be subjective, to limit this threat, we ensured our annotator had experience in the SE domain and extensive guidelines to follow. Our data set is limited to texts from Q&A site. Texts from other software artifacts, such as tutorials, mailing list might perform differently. Also, both the data sets have a total of 5500 sentences, this may not be representative of all kinds of possible communications present in online Q&A forums. Scaling to a much larger data set might lead to different results. For comparison baseline we used RNN4SentiSE, using a different architecture may impact the results. However, this would not deter the performance of BERT4SentiSE.

X. CONCLUSIONS AND FUTURE WORK

Indeed, our research has shown that the BERT language representation model that has been so successful for NLP tasks on English texts, can make significant improvements in the sentiment analysis task of software-related texts when adapted to the SE domain-knowledge of Stack Overflow posts. The improvements bring sentiment analysis for SE from the 0.4-0.6 f-measure range of the previous state-of-the-art up to an f-measure of 0.87. The primary reason for the improvements are the large improvements in the f-measure for the negative class to 0.91 and positive class to 0.78.

Future improvements to sentiment analysis for the SE domain could focus on more kinds of SE text data (tutorials, blog posts), more balanced and larger data sets. These improvements achieved by BERT for sentiment analysis in the SE domain can benefit developers in various downstream tasks, such as extracting problematic API features by detecting negative sentiment sentences in online API discussion forums, recommending software libraries by mining developers' opinions towards different libraries.

References

[1] Martín Abadi, Ashish Agarwal, Paul Barham, Eugene Brevdo, Zhifeng Chen, Craig Citro, Greg S. Corrado, Andy Davis, Jeffrey Dean, Matthieu Devin, Sanjay Ghemawat, Ian Goodfellow, Andrew Harp, Geoffrey Irving, Michael Isard, Yangqing Jia, Rafal Jozefowicz, Lukasz Kaiser, Manjunath Kudlur, Josh Levenberg, Dandelion Mané, Rajat Monga, Sherry Moore, Derek Murray, Chris Olah, Mike Schuster, Jonathon Shlens, Benoit Steiner, Ilya Sutskever, Kunal Talwar, Paul Tucker, Vincent Vanhoucke, Vijay Vasudevan, Fernanda Viégas, Oriol Vinyals, Pete Warden, Martin Wattenberg, Martin Wicke, Yuan Yu, and Xiaoqiang Zheng. TensorFlow: Large-scale machine learning on heterogeneous systems, 2015. Software available from tensorflow.org.

[2] Ashutosh Adhikari, Achyudh Ram, Raphael Tang, and Jimmy Lin. Docbert: Bert for document classification. *arXiv preprint arXiv:1904.08398*, 2019.

[3] Ahmed, Toufique and Bosu, Amiangshu and Iqbal, Anindya and Rahimi, Shahram. SentiCR: A Customized Sentiment Analysis Tool For Code Review Interactions. In *Proceedings of the 32nd IEEE/ACM International Conference on Automated Software Engineering*, pages 106–111. IEEE Press, 2017.

[4] Eeshita Biswas, K Vijay-Shanker, and Lori Pollock. Exploring word embedding techniques to improve sentiment analysis of software engineering texts. In *2019 IEEE/ACM 16th International Conference on Mining Software Repositories (MSR)*, pages 68–78. IEEE, 2019.

[5] Mohamad Adam Bujang and Nurakmal Baharum. A simplified guide to determination of sample size requirements for estimating the value of intraclass correlation coefficient: a review. *Archives of Orofacial Science*, 12(1), 2017.

[6] Calefato, Fabio and Lanubile, Filippo and Maiorano, Federico and Novielli, Nicole. Sentiment Polarity Detection For Software Development. *Empirical Software Engineering*, 23(3):1352–1382, 2018.

[7] Nitesh V Chawla, Kevin W Bowyer, Lawrence O Hall, and W Philip Kegelmeyer. Smote: synthetic minority over-sampling technique. *Journal of artificial intelligence research*, 16:321–357, 2002.

[8] Zhenpeng Chen, Yanbin Cao, Xuan Lu, Qiaozhu Mei, and Xuanzhe Liu. Sentimoji: an emoji-powered learning approach for sentiment analysis in software engineering. In *Proceedings of the 2019 27th ACM Joint Meeting on European Software Engineering Conference and Symposium on the Foundations of Software Engineering*, pages 841–852, 2019.

[9] Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. Bert: Pre-training of deep bidirectional transformers for language understanding. *arXiv preprint arXiv:1810.04805*, 2018.

[10] Ding, Jin and Sun, Hailong and Wang, Xu and Liu, Xudong. Entity-Level Sentiment Analysis Of Issue Comments. In *Proceedings of the 3rd International Workshop on Emotion Awareness in Software Engineering*, pages 7–13. ACM, 2018.

[11] Bjarke Felbo, Alan Mislove, Anders Søgaard, Iyad Rahwan, and Sune Lehmann. Using millions of emoji occurrences to learn any-domain representations for detecting sentiment, emotion and sarcasm. *arXiv preprint arXiv:1708.00524*, 2017.

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

[13] Emitza Guzman and Bernd Bruegge. Towards emotional awareness in software development teams. In *Proceedings of the 2013 9th Joint Meeting on Foundations of Software Engineering*, ESEC/FSE 2013, pages 671–674, New York, NY, USA, 2013. ACM.

[14] Imtiaz, Nasif and Middleton, Justin and Girouard, Peter and Murphy-Hill, Emerson. Sentiment And Politeness Analysis Tools On Developer Discussions Are Unreliable, But So Are People. In *2018 IEEE/ACM 3rd International Workshop on Emotion Awareness in Software Engineering (SEmotion)*, pages 55–61. IEEE, 2018.

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

[16] Islam, Md Rakibul and Zibran, Minhaz F. DEVA: Sensing Emotions In The Valence Arousal Space In Software Engineering Text. In *Proceedings of the 33rd Annual ACM Symposium on Applied Computing*, pages 1536–1543. ACM, 2018.

[17] Robbert Jongeling, Subhajit Datta, and Alexander Serebrenik. Choosing your weapons: On sentiment analysis tools for software engineering research. In *2015 IEEE International Conference on Software Maintenance and Evolution (ICSME)*, pages 531–535. IEEE, 2015.

[18] Robbert Jongeling, Proshanta Sarkar, Subhajit Datta, and Alexander Serebrenik. On negative results when using sentiment analysis tools for software engineering research. *Empirical Software Engineering*, 22(5):2543–2584, Oct 2017.

[19] Diederik P Kingma and Jimmy Ba. Adam: A method for stochastic optimization. *arXiv preprint arXiv:1412.6980*, 2014.

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

[21] Li, Shoushan and Wang, Zhongqing and Zhou, Guodong and Lee, Sophia Yat Mei. Semi-Supervised Learning For Imbalanced Sentiment Classification. In *Twenty-Second International Joint Conference on Artificial Intelligence*, 2011.

[22] Bin Lin, Fiorella Zampetti, Gabriele Bavota, Massimiliano Di Penta, Michele Lanza, and Rocco Oliveto. Sentiment analysis for software engineering: How far can we go? In *Proceedings of the 40th International Conference on Software Engineering*, ICSE '18, pages 94–104, New York, NY, USA, 2018. ACM.

[23] Yang Liu. Fine-tune bert for extractive summarization. *arXiv preprint arXiv:1903.10318*, 2019.

[24] Tomas Mikolov, Kai Chen, Greg Corrado, and Jeffrey Dean. Efficient estimation of word representations in vector space. *CoRR*, abs/1301.3781, 2013.

[25] Mubarok, Mohamad Syahrul and Adiwijaya and Aldhi, Muhammad Dwi. Aspect-Based Sentiment Analysis To Review Products Using Naïve Bayes. In *AIP Conference Proceedings*, volume 1867, page 020060. AIP Publishing, 2017.

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

[27] Novielli, Nicole and Girardi, Daniela and Lanubile, Filippo. A Benchmark Study On Sentiment Analysis For Software Engineering Research. In *2018 IEEE/ACM 15th International Conference on Mining Software Repositories (MSR)*, pages 364–375. IEEE, 2018.

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

[29] Marco Ortu, Giuseppe Destefanis, Steve Counsell, Stephen Swift, Roberto Tonelli, and Michele Marchesi. Arsonists or firefighters? affectiveness in agile software development. In *International Conference on Agile Software Development*, pages 144–155. Springer, Cham, 2016.

[30] Marco Ortu, Alessandro Murgia, Giuseppe Destefanis, Parastou Tourani, Roberto Tonelli, Michele Marchesi, and Bram Adams. The emotional side of software developers in jira. In *Proceedings of the 13th International Conference on Mining Software Repositories*, MSR '16, pages 480–483, New York, NY, USA, 2016. ACM.

[31] Sebastiano Panichella, Andrea Di Sorbo, Emitza Guzman, Corrado A Visaggio, Gerardo Canfora, and Harald C Gall. How can i improve my app? classifying user reviews for software maintenance and evolution. In *2015 IEEE International Conference on Software Maintenance and Evolution (ICSME)*, pages 281–290. IEEE, 2015.

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

[33] Prusa, Joseph and Khoshgoftaar, Taghi M and Dittman, David J and Napolitano, Amri. Using Random Undersampling To Alleviate Class Imbalance On Tweet Sentiment Data. In *2015 IEEE International Conference on Information Reuse and Integration*, pages 197–202. IEEE, 2015.

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

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

[36] Socher, Richard and Perelygin, Alex and Wu, Jean and Chuang, Jason and Manning, Christopher D and Ng, Andrew and Potts, Christopher.

Recursive Deep Models For Semantic Compositionality Over A Sentiment Treebank. In *Proceedings of the 2013 Conference on Empirical Methods in Natural Language Processing*, pages 1631–1642, 2013.

[37] Parastou Tourani, Yujuan Jiang, and Bram Adams. Monitoring sentiment in open source mailing lists: exploratory study on the apache ecosystem. In *Proceedings of 24th annual international conference on computer science and software engineering*, pages 34–44. IBM Corp., 2014.

[38] Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N Gomez, Łukasz Kaiser, and Illia Polosukhin. Attention is all you need. In *Advances in neural information processing systems*, pages 5998–6008, 2017.

[39] Michal R Wrobel. The impact of lexicon adaptation on the emotion mining from software engineering artifacts. *IEEE Access*, 2020.

[40] Yonghui Wu, Mike Schuster, Zhifeng Chen, Quoc V Le, Mohammad Norouzi, Wolfgang Macherey, Maxim Krikun, Yuan Cao, Qin Gao, Klaus Macherey, et al. Google's neural machine translation system: Bridging the gap between human and machine translation. *arXiv preprint arXiv:1609.08144*, 2016.

[41] Yingying Zhang and Daqing Hou. Extracting problematic api features from forum discussions. In *Program Comprehension (ICPC), 2013 IEEE 21st International Conference on*, pages 142–151. IEEE, 2013.