An Empirical Study of the Effectiveness of an Ensemble of Stand-alone Sentiment Detection Tools for Software Engineering Datasets

GIAS UDDIN, University of Calgary, Canada
YANN-GAËL GUÉHÉNUC, Concordia University, Canada
FOUTSE KHOMH, Polytechnique Montréal, Canada
CHANCHAL K. ROY, University of Saskatchewan, Canada

Sentiment analysis in **software engineering (SE)** has shown promise to analyze and support diverse development activities. Recently, several tools are proposed to detect sentiments in software artifacts. While the tools improve accuracy over off-the-shelf tools, recent research shows that their performance could still be unsatisfactory. A more accurate sentiment detector for SE can help reduce noise in analysis of software scenarios where sentiment analysis is required. Recently, combinations, i.e., hybrids of stand-alone classifiers are found to offer better performance than the stand-alone classifiers for fault detection. However, we are aware of no such approach for sentiment detection for software artifacts. We report the results of an empirical study that we conducted to determine the feasibility of developing an ensemble engine by combining the polarity labels of stand-alone SE-specific sentiment detectors. Our study has two phases. In the first phase, we pick five SE-specific sentiment detection tools from two recently published papers by Lin et al. [29, 30], who first reported negative results with stand alone sentiment detectors and then proposed an improved SE-specific sentiment detector, POME [29]. We report the study results on 17,581 units (sentences/documents) coming from six currently available sentiment benchmarks for software engineering. We find that the existing tools can be complementary to each other in 85-95% of the cases, i.e., one is wrong but another is right. However, a majority voting-based ensemble of those tools fails to improve the accuracy of sentiment detection. We develop Sentisead, a supervised tool by combining the polarity labels and bag of words as features. Sentisead improves the performance (F1-score) of the individual tools by 4% (over Senti4SD [5]) – 100% (over POME [29]). The initial development of Sentisead occurred before we observed the use of deep learning models for SE-specific sentiment detection. In particular, recent papers show the superiority of advanced language-based **pre-trained transformer models (PTM)** over rule-based and shallow learning models. Consequently, in a second phase, we compare and improve Sentisead infrastructure using the PTMs. We find that a Sentisead infrastructure with RoBERTa as the ensemble of the five stand-alone rule-based and shallow learning SE-specific tools from Lin et al. [29, 30] offers the best F1-score of 0.805 across the six datasets, while a stand-alone RoBERTa shows an F1-score of 0.801.

CCS Concepts: • **Information Systems** → *Sentiment analysis*; • **Computing Methodologies** → *Machine Learning;*

Authors' addresses: G. Uddin, University of Calgary, 2500 University Dr NW, Calgary, AB T2N 1N4, Canada; email: gias.uddin@ucalgary.ca; Y.-G. Guéhénuc, Concordia University, 1455 Boulevard de Maisonneuve O, Montréal, QC H3G 1M8, Canada; email: yann-gael.gueheneuc@concordia.ca; F. Khomh, Concordia University, 1455 Boulevard de Maisonneuve O, Montréal, QC H3G 1M8, Canada; email: foutse.khomh@polymtl.ca; C. K. Roy, University of Saskatchewan, Canada; email: croy@cs.usask.ca.

1049-331X/2022/04-ART48 $15.00
https://doi.org/10.1145/3491211

Additional Key Words and Phrases: Sentiment analysis, machine learning, ensemble classifier

ACM Reference format:
Gias Uddin, Yann-Gaël Guéhénuc, Foutse Khomh, and Chanchal K. Roy. 2022. An Empirical Study of the Effectiveness of an Ensemble of Stand-alone Sentiment Detection Tools for Software Engineering Datasets. *ACM Trans. Softw. Eng. Methodol.* 31, 3, Article 48 (April 2022), 38 pages.
https://doi.org/10.1145/3491211

1 INTRODUCTION

According to Bing Liu, "the textual information in the world can broadly be categorized into two main types: facts and opinions" [31]. Such conjecture can also be applied to **software-engineering (SE)** artifacts, e.g., the discussions in online developer forums, and so on. "Facts" are objective expressions (e.g., "I use this tool"). Opinions are subjective expressions that convey the sentiments towards entities (e.g., "I like this tool"). While the concept of opinion can be broad, the major focus of sentiment analysis is to detect polarity [31, 48], i.e., given a sentence or a post as a unit of analysis (e.g., a sentence/post - denoted as **"unit"** from now on), whether the unit exhibits positive, negative, or neutral sentiment (i.e., absence of positivity or negativity).

Opinions are key determinants to many of the activities in SE [18, 36, 60]. The productivity in a team may depend on developers' sentiments in/of their diverse development activities [18, 46, 51], while the analysis of sentiments towards software artifacts may lead to better selections of software artifacts [60, 62]. Therefore, it is necessary to accurately detect sentiments in software-engineering artifacts [41].

Most prior work on sentiment analysis in SE used cross-domain sentiment-detection tools. This "off-the-shelf" usage is insufficiently accurate for SE, due to the difference in the domain (see Section 7). Many cross-domain sentiment-detection tools were developed using movie reviews (e.g., Stanford [56]). The context of sentiments expressed in movie reviews is different from that in SE. For example, the 'API is simple' is a positive API review [60], but 'the movie is simple' is generally a negative movie review [31]. Over the last few years, several tools have been developed to detect sentiments in software artifacts, e.g., SentistrengthSE [21], Senti4SD [5], and SentiCR [1]. These tools offer better accuracy than off-the-shelf tools. However, recently Lin et al. [30] reported that the tools can still be significantly inaccurate.

Recent research in other domains has seen a rise in combining multiple sentiment detectors, such as hybrids of rule-based and supervised classifiers observed to be better detectors of sentiments in Twitter messages [12, 15]. In software engineering, the choice of classification methods and the combination of diverse classifiers is effective to detect defects [14, 45]. Intuitively, an ensemble of existing sentiment detection tools for SE can learn from the strengths and weaknesses of each tool for a given dataset, which then may offer it better insight to determine the correct sentiment polarity label of the different units in the dataset. This is because we find that different SE specific sentiment detection tools are developed using diverse rules and algorithms. We are particularly interested to study how a hybrid of existing sentiment detection tools for SE performs, because of the superiority of hybrid tools for sentiment detection in other domains like Twitter. For example, Balage et al. [12] found that a hybrid of rule-based (i.e., that uses sentiment lexicons as cues) and supervised tools (i.e., that uses features to learn) offered better sentiment detection performance for Twitter messages, because the tools could complement each other. The hybrid tool in Balage et al. [12] was superior to the stand-alone tools, because each stand-alone tool (e.g., the rule-based tool) focused on specific estimators (e.g., sentiment lexicons). In SE, sentiment detection tools like SentistrengthSE are rule-based while other tools like Senti4SD are supervised. Therefore, the tools can also exhibit strengths and weaknesses on particular features/estimators in a dataset, which a hybrid tool can find as complementary. However, we are aware of no such hybrid sentiment detection tool developed for SE so far.

To determine the feasibility of developing a hybrid of stand-alone SE-specific sentiment tools, our empirical study in this paper started in 2019 with an aim to gain more positive outlook on the state of sentiment detection than the negative results reported by Lin et al. [30]. Therefore, we initially intended to study all the sentiment detection tools developed for software engineering included in the study of Lin et al. [30] and that are currently available through GitHub or other online sources, including the latest such tool, POME, by also Lin et al. [29]. These tools are either rule-based or shallow machine learning models. Looking forward in 2020, we observed new uses of deep learning models to detect sentiment in software engineering [3, 4, 9, 74]. The most recent works are by Zhang et al. [74] and Biswas et al. [3], who investigated the effectiveness of advanced **pre-trained language based transformer models (PTMs)** like BERT [11]. They found that the PTMs outperform rule-based and shallow learning models. Therefore, we also study the effectiveness of those PTMs to support/outperform our hybrid engine of SE-specific sentiment detectors. Thus, we divided our empirical study into two phases:

- **Phase 1.** We studied the feasibility of developing a hybrid of stand-alone SE-specific sentiment tool, based on the tools studied by Lin et al. [29, 30].
- **Phase 2.** We studied how the hybrid tool that we developed in Phase 1 compares (and can be improved) with the recent, advanced pre-trained language-based models by Zhang et al. [74].

We studied a total of nine available sentiment detection tools for software engineering based on the studies of Lin et al. [29, 30] and Zhang et al. [74]: five from Lin et al. [29, 30] and four from Zhang et al. [74]. Through our empirical study, we answer a total of eight research questions as discussed below.

In Phase 1, our focus was the feasibility of developing an ensemble of the five tools that could potentially offer better performance than each of the stand-alone tools from Lin et al. [29, 30]. In Phase 1, we answered a total of five research questions as follows:

RQ$_1$. How frequently do we get at least one correct classification while using multiple SE-specific tools? On a total of 17,581 units from six software specific sentiment benchmarks from the literature, we ran five stand-alone SE-specific sentiment tools: Senti4SD [5], SentiCR [1], SentistrengthSE [21], Opiner [63], and POME [29]. We find that 85% of misclassification by the tools related to non-neutral polarity (positive, negative) and 99% neutral classes can be potentially corrected by at least one other tool. This finding motivated us to investigate whether a hybrid sentiment detector can be developed by a simple majority analysis as follows.

RQ$_2$. Can a majority voting-based classifier perform better than individual classifiers? For each of the 17,581 units, we assign a polarity label by simply taking the majority voting on the unit by the tools, i.e., if three tools say 'positive' and two say 'negative', we assign a final polarity as 'positive'. Unfortunately, this classifier only achieves an F1-score (Macro) of 0.746, which was outperformed by both supervised stand-alone detectors, Senti4SD (0.750) and SentiCR (0.756). This finding necessitates towards a more sophisticated design of the hybrid tool. To determine that, we analyzed why one tool misclassifies when another does not.

RQ$_3$. What are the error categories that could potentially be corrected by combining multiple SE-specific tools? We manually label 153 units where at least one tool was right and at least one other was wrong. For the labeling, we follow error categories similar to [43], i.e., we determine the specific reason for misclassification by a tool. We find five major error categories. The most prevalent category is the failure of a tool to understand the underlying context (related to an expressed sentiment). Such contexts can be better understood by analyzing the textual contents in the unit. This motivates us to investigate the possibility of leveraging ensemble learning to capture the strength of multiple sentiment detectors. We therefore answer the following research questions.

RQ_4. Can a supervised ensemble of classifiers offer better performance than individual classifiers? For each of the 17,581 units, we combine the polarity labels of the five individual sentiment tools with the bag of words of the unit as features. We train a supervised machine learning classifier based on shallow machine learning **Random Forest (RF)** model using the features. We name the classifier Sentisead_{RF}. Sentisead_{RF} outperforms each stand-alone sentiment detector from 4% (Senti4SD) to >100% (POME). We investigate the possibility to further improve Sentisead_{RF} with additional features through the following research question.

RQ_5. Can the addition of more features improve the performance of the hybrid tool? We introduce two new types of features in Sentisead_{RF} based on the analysis of polarity words and their positions in a given unit (e.g., first vs. last) and *entropy* measures using information theory. The enhanced hybrid engine, i.e., $\text{Sentisead}_{RF}+$, however, shows negligible performance increase over Sentisead_{RF} (increase of 0.001 in F1-score Macro).

In Phase 2, our focus was to study the effectiveness of Sentisead_{RF} against the most recently introduced advanced language-based models for SE-specific sentiment tools, which were studied in multiple papers like Zhang et al. [74]. We answer a total of three research questions as follows:

RQ_6. Can Sentisead_{RF} outperform the BERT-based advanced pre-trained language based models? Given that **pre-trained advanced language-based transformers (PTMs** for short) offer better performance than shallow learning models for sentiment detection in SE, we want to know whether the PTMs can also outperform Sentisead_{RF}, our hybrid tool of rule-based and shallow learning models based on the Random Forest model. We investigated four PTMs from Zhang et al. [74], ALBERT, BERT, RoBERTa, and XLNet, on our six datasets. We found that RoBERTa is the best performer among the four stand-alone PTMs with a Macro F1-score of 0.801, which is slightly better than Sentisead_{RF} by 1.8%. Following this minor improvement, we decided to investigate the following research question.

RQ_7. Can a deep learning model as ensembler for Sentisead outperform the stand-alone BERT-based PTMs? Motivated by superior performance of stand-alone PTMs over Sentisead_{RF} that uses a shallow learning Random Forest model, we replaced the Random Forest model in Sentisead_{RF} by the PTMs: we trained and tested four versions of Sentisead where features are the unit textual contents and the polarity labels from the five rule-based and shallow learning models from Lin et al. [29, 30]. However, the ensembler was a PTM instead of Random Forest. We found that the best performing PTM-based ensembler of the stand-alone rule-based and shallow learning sentiment detectors is based on RoBERTa, which we named $\text{Sentisead}_{RoBERTa}$. Across the six datasets (and using the same 10-fold cross validation), $\text{Sentisead}_{RoBERTa}$ shows a Macro F1-score of 0.805: it offers 0.5% increase over a stand-alone RoBERTa model and a 2.3% increase in performance over Sentisead_{RF}. This slight performance increase motivated us to include the stand-alone PTMs into the hybrid architecture as follows.

RQ_8. Can Sentisead based on an ensemble of all the models (i.e., PTMs + non-PTMs) offer the best performance of all tools? We investigated whether the performance of the ensembler could further increase using the sentiment polarity labels of a unit from all available stand-alone models (i.e., rule-based, shallow, and deep learning models) in addition to the textual content of the unit. We trained and tested four versions of Sentisead by taking as input the above polarity labels and textual content of a unit as features. The ensemblers are the four PTMs, which yield four ensemblers: $\text{Sentisead}_{ALBERT}+$, $\text{Sentisead}_{BERT}+$, $\text{Sentisead}_{RoBERTa}+$, and $\text{Sentisead}_{XLNet}+$. The best performing model is Sentisead$_{RoBERTa}$+ with a Macro F1-score of 0.8. However, Sentisead$_{RoBERTa}$+ is not more accurate than Sentisead$_{RoBERTa}$ from RQ$_7$. Therefore, the inclusion of polarity labels of individual PTMs does not contribute to any performance improvement in Sentisead.

Based on our empirical study findings, we make the following major observations and suggestions:

(1) Our hybrid classifier, Sentisead$_{RoBERTa}$, which is based on the PTM RoBERTa, offers the best Macro F1-score of 0.805. This classifier can be easily built by taking as input the sentiment polarity of the five stand-alone sentiment detectors (Opiner, POME, Senti4SD, SentiCR, and SentistrengthSE) and the unit textual contents. However, Sentisead$_{RoBERTa}$ shows only 0.5% performance improvement over the stand-alone RoBERTa model. Therefore, when a mere increase of 0.5% is not warranted, a stand-alone RoBERTa model should suffice.
(2) The hybrid classifier using Random Forest achieves the best Macro F1-score of 0.787 out of all shallow learning-based hybrid models. Thus, this Random Forest model shows a mere decrease of 0.018 in F1-score compared to the overall best hybrid model, i.e., the RoBERTa based PTM which shows a Macro F1-score of 0.805. Therefore, for systems where GPU-based servers are not readily available, the Random Forest-based hybrid model can suffice.
(3) On the six benchmarks, Sentisead$_{RF}$ or Sentisead$_{RoBERTa}$ show more than 0.8-0.9 F1-score (for positive and negative classes) for four datasets but only 0.3-0.5 for the other two datasets. Further improvement in the latter two datasets require the analysis of contextual information *surrounding* a given unit (e.g., preceding & following units); an aspect that current tools cannot handle. Thus, future improvement of stand-alone SE-tools should focus on developing algorithms to analyze surrounding contexts.

Our study benefited from the availability of open source code, tools, and datasets that were shared by software engineering researchers working on sentiment detection. In fact, the code base of Sentisead uses the SentiCR [1] code base to bootstrap. Given that our study shows promise to advance the state-of-the-art research in software engineering, we have open sourced the entire codebase of Sentisead in our online appendix [27].

2 STUDIED BENCHMARKS AND TOOLS

In this section, we describe the sentiment detection benchmarks and tools that we studied to develop our hybrid engine.

2.1 Studied Sentiment Benchmarks

We analyze the following six benchmarks available from software-engineering research. Each unit in a benchmark is labeled as a polarity (positive, negative, or neutral). A unit can be a sentence or a document (i.e., a list of sentences). Table 1 provides descriptive summary statistics of the benchmarks. Overall, 46% of the units in the benchmarks are labeled as positive or negative and the rest (64%) are labeled as neutral (see last column in Table 1).

(1) **Stack Overflow: Calefato et al. [5].** The benchmark is based on 4,423 randomly-sampled units from Stack Overflow. It was annotated for polarity by 12 coders. Each unit was annotated by three coders using majority voting. The benchmark was used to train and test Senti4SD [5]. It is available in the Senti4SD GitHub repository.

(2) **Stack Overflow: Lin et al. [30].** The benchmark contains 1,500 units, each manually assigned a sentiment score ranging from -2 (strong negative) to +2 (strong positive) by two coders. Each unit was divided into a number of nodes (e.g., clauses), which were assigned a sentiment

Table 1. Benchmarks Used in the Study (SO = Stack Overflow)

Dataset	#Units	+VE	−VE	±/#Units
SO Calefato et al. [5]	4423	1527	1202	62%
SO Lin et al. [30]	1,500	131	178	21%
SO Uddin et al. [63]	4,522	1,048	839	42%
JIRA Ortu et al. [46]	5,869	1,128	786	33%
JIRA Lin et al. [30]	926	290	636	100%
Mobile App Lin et al. [30]	341	186	130	93%
Total	17,581	4,310	3,771	46%

score. We use the final sentiment scores available in the replication package of [30] to assign a polarity: +1 positive, −1 negative, and 0 neutral.

(3) **Stack Overflow: Uddin et al. [63].** The benchmark consists of the 4,522 units in 71 randomly-sampled Stack Overflow threads. Each unit was manually labeled for polarity by at least two coders with a third coder consulted in case of disagreements. Unlike the above four benchmarks, this benchmark includes all the units from the threads, i.e., contextual information. We obtained this benchmark from the authors of Opiner [60], who used it to develop an online opinion summarization engine.

(4) **Jira: Ortu et al. [46].** The benchmark consists of 6,000 units (2,000 issue comments and 4,000 units contributed by developers using Jira). The issue comments were collected from four popular open-source frameworks/repositories: Apache, Codehaus, JBoss, and Spring. The original benchmark was annotated for emotions, such as love, joy, surprise, anger, fear, and sadness and later labeled with three polarities by Novielli et al. [43], which we obtained from the authors for this paper. In their translation, Novielli et al. coded "joy" and "love" as positive, while "sadness", "fear" and "anger" as negative. An absence of any emotion for a unit was considered neutral. They discarded 131 units labeled as "surprise" because of the lack of contextual information needed to identify the underlying polarity.

(5) **Jira: Lin et al. [30].** Lin et al. [30] took 926 units from Ortu et al. [46] dataset that was originally labeled as six different types of emotions: love, joy, surprise, anger, fear, and sadness. Lin et al. [46] considered the units labeled as joy and love as positive and the units labeled as anger and sadness as negative.

(6) **Apps Reviews: Lin et al. [30].** It contains 341 mobile-app reviews from Villarroel et al. [68], who labeled each review to identify its type (e.g., bug reporting, request for enhancement, etc.). Each review was labeled for polarity by two of the authors of [30].

2.2 Studied Sentiment Detection Tools in Phase 1

The observed shortcomings in cross-domain sentiment detection tools motivated the development of several recent sentiment detection tools for software engineering. In this paper, we have studied five sentiment detection tools developed for software engineering: SentistrengthSE [21], Senti4SD [5], SentiCR [1], Opiner [63], and POME [29]. While Senti4SD and SentiCR are supervised, the other three tools are rule-based classifiers. As we noted in Section 1, these tools were the subject of two papers by Lin et al. [29, 30]. From the two studies of Lin et al. [29, 30], we picked the tools whose trained model/source code were available online during the time of analysis. We also picked the tools that were developed specifically for software engineering artifacts. These five tools are also the five most cited sentiment detection tools developed for software engineering as of now.

(1) **Senti4SD [5].** Given as input a short text, Senti4SD detects the polarity of the text as positive, negative, or neutral. Senti4SD [5] is trained on a dataset of 4,000 posts (questions, answers, and

comments) from Stack Overflow. The classifier leverages a suite of textual features both based on input text (e.g., ngrams) as well as a pre-defined list of sentiment lexicons and word embeddings. The word embeddings are compiled from an entire Stack Overflow dump to offer domain specific information, such as semantic similarity between the input text and Stack Overflow texts that are closely similar to the 4,000 posts used in the benchmark. The supervised classifier is a trained **support vector machine (SVM)** model.

(2) **SentiCR [1].** SentiCR [1] was trained on a dataset of 1,600 code reviews from Gerrit. While Senti4SD uses an SVM classifier, the currently distributed version of SentiCR leverages the **Gradient Boosting Tree (GBT)** algorithm. Unlike Senti4SD, SentiCR handles the under-representation (i.e., class imbalance) of the polarity labels (i.e., positive and negative) compared to the neutral classes by using the **SMOTE** algorithm **(synthetic minority over-sampling technique**) [8]. Note that we train SentiCR to detect three polarity classes following Novielli et al. [43] by using numeric identifiers: positive (+1), negative (−1), and neutral (0).

(3) **SentistrengthSE [21].** SentiStrengthSE [21] was developed on top of Sentistrength [57] by introducing rules and sentiment words specific to the domain of software engineering [21]. Each negative word has a score ranging from −2 to −5, positive word has a score ranging from +2 to +5. The polarity scores are *a priori*, i.e., they do not depend on the contextual nature of the sentiment expressed in a unit. Similar to Sentistrength, SentistrengthSE outputs both positive and negative scores for an input text. Following state of art [43], the overall polarity score of an input text can be calculated by taking the algebraic sum of the positive and negative scores. The text is labeled as ‘positive’ if the sum of scores is greater than 0, negative if the sum is less than 0, and neutral otherwise.

(4) **Opiner [63]**. The tool adapts the **Domain Sentiment Orientation (DSO)** algorithm originally proposed by Hu and Liu [20] to collect online customer reviews. DSO assigns a polarity to a unit based on the presence of sentiment words. The SE-specific adaptation in Opiner was used to mine and summarize API reviews from Stack Overflow [60, 62, 63]. Opiner assigns polarity to an input text in three major steps: (a) *Detect potential sentiment words* by identifying adjectives in the unit that match the sentiment words in a list of predefined sentiment vocabularies. Opiner gives a score of +1 to a recorded adjective with positive polarity and a score of −1 to an adjective of negative polarity. (b) *Detect negations.* The sign for an adjective alternates if a negation word is found close to the word.

(c) *Label unit.* If the sum of all polarity scores is greater than 0, Opiner labels the unit as ‘positive’. If the sum is less than 0, Opiner labels it as ‘negative’.

(5) **POME [29]**. In 2019, Lin et al. [29] proposed POME, a pattern-based tool to mine opinions about software aspects from Stack Overflow. The tool serves two purposes simultaneously: detection of the aspects and of sentiment polarity in units (i.e., sentences). The patterns are derived by manually assessing 1,662 units from Stack Overflow. The manual assessment produced 157 distinct patterns. For example, a pattern can check the presence of specific API aspect (e.g., performance) and sentiment lexicons related to the API aspect in an input text. The pattern follows strict matching rules, such as subject (in a unit) followed by API aspect (e.g., performance, usability), followed/preceded by a list of predefined sentiment lexicons.

In a benchmark-based study against six state-of-the-art SE-specific sentiment detection tools including all four stand-alone rule-based and shallow sentiment detection tools that we have studied in this paper (i.e., Opiner, Senti4SD, SentiCR, StrengthSE), POME outperformed each tool in the benchmark. While POME serves two features (sentiment and aspect detection) compared to the other four studied tools that serve one feature (sentiment detection), POME does not limit itself to a specific purpose (e.g., only aspect detection or only sentiment detection). Indeed, in their evaluation of POME, Lin et al. [29] considered other sentiment detection tools like Opiner as baselines (see RQ_2 and RQ_3 in Lin et al. [29]). We, therefore, include POME as a stand-alone sentiment detection tool in our study.

2.3 Studied Deep Learning Tools for Sentiment Detection in Phase 2

Starting from mid 2019, the SE community started using deep learning models to detect sentiment in SE artifacts. Most recently, several research results showed the superior performance of pre-trained advanced language-based transformers for sentiment detection in SE. For the sake of brevity, we refer to those models as **PTMs (Pre-trained Transformer Models)** in the following. Among the PTMs, four models, ALBERT, BERT, RoBERTa, and XLNet, were studied in the most recent papers by Zhang et al. [74] and Biswas et al. [3], who found that these PTMs offer better performance than shallow learning models, like Senti4SD, SentiCR, etc. In the second phase of our study, we thus investigate the effectiveness of the PTMs with our ensemble sentiment detection model. We discuss briefly the PTMs below.

(1) **BERT [11]** stands for **Bidirectional Encoder Representation from Transformers**. Google developed this transformer-based ML technique for textual analysis. BERT is a transformer model that processes words in a unit in relation to all words in the unit, which is different from sequence to sequence models (e.g., LSTM), which utilizes the forward/backward sequences of words one-by-one in a unit. BERT models thus are capable of considering the full context of a given word in a unit by looking at the preceding and following words. Contextual word representations are learned based on two tasks:

(a) Masking some words from the input unit based on a **Masked Language Model (MLM)** and predicting the masked words based on the context.

(b) **Next unit prediction (NSP)** to determine whether one unit follows another.

Google provided two architectures of BERT, both based on a multi-layer bidirectional transformer model: $BERT_{BASE}$ and $BERT_{LARGE}$. $BERT_{BASE}$ has 12 bidirectional self-attention heads while $BERT_{LARGE}$ has 24 heads. Both models are trained using unlabeled data from English Wikipedia and a corpus of books. BERT achieved state-of-the-art performance on text understanding tasks in several benchmarks (e.g., GLUE).

(2) **RoBERTa [32]** is a **Ro**bustly Optimized **BERT a**pproach developed by Facebook to offer improvements over the BERT model developed by Google. RoBERTa follows BERT's strategy of masking, i.e., the system learns to predict intentionally masked words within a unit. In addition, the following modifications led to better performance than BERT for downstream tasks, e.g., classification:

(a) RoBERTa removes next unit prediction in BERT, which is not important for text classification (e.g., for our sentiment detection). This removal offers more flexibility to improve on the MLM of BERT.

(b) RoBERTa is trained and hyper-tuned on more data than BERT, for a longer amount of time.

(3) **XLNet [73]** is designed to address the concern in BERT that by "relying on corrupting the input with masks, BERT neglects dependency between the masked positions and suffers from a pretrain-finetune discrepancy". XLNet is a generalized auto-regressive pre-training method based on Transformer-XL, which learns bidirectional contexts by maximizing the expected likelihood of words over all permutations of the order. XLNet also provides a modified language model training objective over BERT to learn conditional distributions for all permutations of word tokens. In empirical evaluations, XLNet outperformed BERT in 20 tasks (e.g., sentiment analysis).

(4) **ALBERT [28]** offers an upgrade on BERT on 12 textual processing tasks (e.g., question answering) by allocating the model more efficiently for input-level embeddings (words, sub-tokens) that are needed to learn context independent representations and hidden-layer embeddings that are needed to refine the input-level embeddings into context-dependent representations. ALBERT

Table 2. Architecture Details of Variants of BERT Model
(L = Layers, H = Hidden, D = Heads, P = Parameters)

Architecture	Used Model	L	H	D	P
BERT	bert-base-uncased	12	768	12	110M
XLNet	xlnet-base-cased	12	768	12	110M
RoBERTa	roberta-base	12	768	12	125M
ALBERT	albert-base-v1	12	768	12	11M

factorizes the embedding parameterization, where the embedding matrix is split into input-level embedding with relative-lower dimension than then hidden-layer embeddings (e.g., 128 in input-level vs. 768 or more in hidden-layer), which allows ALBERT to achieve an 80% reduction in the parameters. Another design decision in ALBERT is to apply the same layer on top of each other, while BERT creates separate independent layers on top of each other. This decision in ALBERT is based on the observation that often multiple layers in BERT learn almost similar operations. Thus, ALBERT reduces the architectural complexity of BERT, but at the cost of a slight decrease in performance than BERT. As such, ALBERT is considered as "lite" BERT.

In our study, we use the implementations of the above four models from the Hugging Face transformer library, which is an open-source community around the PTM libraries. Table 2 shows the architecture of the four PTMs.

3 STUDY PHASE 1: ENSEMBLE OF RULE-BASED AND SHALLOW MACHINE LEARNING MODELS

We offer insights into the feasibility of developing a hybrid sentiment detection engine for the domain of software engineering. Our study has two phases. In the first phase, we answer five research questions we introduced in Section 1.

(1) How frequently can we correct the misclassification of a tool by another tool? (Section 3.1)
(2) Can a majority voting-based classifier perform better than individual classifiers? (Section 3.2)
(3) What are the error categories that could be potentially corrected by combining multiple SE-specific tools? (Section 3.3)
(4) Can an ensemble of classifiers offer better performance than individual classifiers? (Section 3.4)
(5) Can the addition of more features improve the performance of the hybrid classifier? (Section 3.5)

3.1 RQ_1 How Frequently Can We Correct the Misclassification of a Tool By Another Tool?

3.1.1 Motivation. The five SE-specific tools (SentiCR, Senti4SD, Opiner, SentistrengthSE, and POME) in our study can complement each other, if the misclassification of one tool can be *potentially* corrected by the correct polarity label from another tool. If all the tools are wrong for a given unit, the proper processing of the unit may require the development of a new tool. As noted in Section 1, our focus is to improve the accurate polarity classification probability of a given unit by analyzing the polarity labels of the units from the existing SE-specific tools. This concept is similar to the bagging or boosting combination of classifiers in machine learning [37, 50, 72, 75]. However, for the combination to perform better than the stand-alone SE-specific classifier, we first need to confirm whether another tool can offer correct polarity classification for a given unit when a tool is wrong on that unit.

3.1.2 Approach. We have six datasets in our study. For each dataset, we collect sentiment polarity labels on each unit from the five sentiment detection tools as follows. We run each of the three unsupervised classifiers (i.e., SentistrengthSE, Opiner, and POME) on the entire benchmark dataset. The output is a polarity label for each unit. The other two tools, Senti4SD and SentiCR, are supervised sentiment detectors so, for each dataset, we trained and tested as follows.

Literature on the evaluation of SE-specific supervised sentiment detection tools has so far split a given dataset into 70% training set and 30% testing set (e.g., Novielli et al. [43] Calefato et al. [5], Zhang et al. [74]). The split is done based on a stratified sampling, i.e., 70% of each polarity class will stay in the training set and 30% of that class will be in the testing set. The split is done once, i.e., before a classifier is trained or tested. In our paper, we improved on this approach as follows.

We split each dataset into 10-folds using a stratified sampling.[1] Thus, each fold contains the same proportion of a polarity class as in the entire dataset. Also, a unit cannot be present in two different folds, because otherwise the classifier would see the same unit in both training and testing sets, which would lead to overfitting. Then, we train and test a classifier 10 times. For example, in the first run, we train the classifier using folds 1-9 and then test it on Fold 10. In the second run, we train the classifier using folds 2-10 and test it on Fold 1. Once all the 10 runs are completed, we have test results on each of the 10 folds (e.g., test result of Fold 10 coming from Run 1, test result of Fold 1 coming from Run 2, etc.). We can then compute the performance of the classifier on all test results.

Therefore, our approach is similar to the existing 70-30 evaluation approach used in the literature but our training vs. testing split is 90-10. In addition, unlike current approaches in SE-specific sentiment detection, because we do the testing on the entire dataset by running a tool 10 times, our test results are more robust than a 70-30 split, which is run only once and is tested only on 30% of all data.

Because we are training and testing each tool 10 times compared to only once in the current literature, our approach also better addresses any variance in the machine learning classification model due to its stochastic nature. An alternative to our approach would have been to do a repeated sampling of training and testing sample from a given dataset for N times (e.g., N = 10 or N = 100) and then report the performance of the model by taking the average of the performance across all test samples. Our approach slightly differs from this alternative in that we created the 10 folds once, before we started all the classification tasks. This means that we were able to track and compare each classification model performance on each fold, where the training and testing for each model followed the same pattern (i.e., train using folds 1-9 and testing using fold 10 in run 1, train using folds 2-10 and test using fold 1 in run 2, etc.). This process then allowed us to also check for tool complementarity, i.e., see when a tool is wrong while another can be right for a given unit. Moreover, given that we train and test a classification model 10 times instead of one time, we also reduce any variance in the classification results. Nevertheless, for each model, we pre-set the random variables by using seeds, which is useful to get similar results for a model for a given dataset, even if we run the model 10, 20, or 100 times in the dataset.

Given that we have six datasets and five SE-specific sentiment detection tools, we analyze the performance of each tool across the six datasets as follows. First, we get the test results for each dataset following the steps listed above. Second, we compute the performance metrics of each model in each dataset by analyzing the polarity labels in the 10 test folds of the dataset. Third, we compute the performance metrics of each model across the six datasets by analyzing the

[1]We use the StratifiedKFold method from Python Scikit-learn to create the $k = 10$ folds. The folds preserve the percentage of samples for each of the polarity classes.

Table 3. How the Misclassification of a Tool Can Be Corrected By One Tool or at Least One (i.e., Last Column ≥ 1) Tool. O = Neutral

Tool Wrong	Polarity Wrong	Tools Could Correct Wrong Polarity Senti4SD	SentiCR	SentistrengthSE	Opiner	POME	≥1
Senti4SD	± (2685)		24%	29%	29%	7%	54%
	O (1281)		54%	47%	41%	88%	97%
SentiCR	± (2948)	31%		28%	37%	7%	59%
	O (917)	36%		37%	43%	88%	0.96
SentistrengthSE	± (2970)	36%	29%		32%	7%	59%
	O (1458)	53%	61%		40%	91%	98%
Opiner	± (4087)	54%	55%	51%		4%	70%
	O (2999)	75%	83%	71%		89%	99%
POME	± (7621)	67%	64%	64%	48%		84%
	O (471)	69%	77%	71%	31%		92%
≥1	± (7938)	66%	63%	63%	49%	4%	85%
	O (4049)	68%	77%	64%	26%	88%	99%

polarity labels across all the test folds in the six datasets, i.e., across the 60 test folds (10 each for each dataset). While we report the consolidated performance metric for a given model across the datasets to save space in the paper, we include the performance metric for each dataset in our online replication package [27].

In summary, given as input the benchmark datasets and the five sentiment detectors, we produce the final output after all the training and testing for each dataset as a matrix $M = U \times F \times L$, where $U = \{U_1, U_2, \ldots U_n\}$ ($n = 17,581$) is each unit, $F = \{F_1, F_2, \ldots F_k\}$ ($k = 6$) corresponds to the benchmark dataset where unit U_i is found, and $L = \{L_1, L_2, \ldots, L_m\}$ ($m = 5$) and L_j corresponds to the polarity label of unit U_i by a given sentiment detector j in our study.

We analyze the complementarity of the tools on the generated matrix M as follows. First, for each pair (e.g., (i, j)) of tools in our dataset (e.g., Senti4SD, SentiCR), we collect all the records $R_{i,j}$ in M where at least one of the tools is right (i.e., matches with the ground truth). For each tool in M, we then compute the fraction of misclassification of each tool, for which one or more of the other tools offer correct classifications.

3.1.3 Results. In Table 3, we show the percentage of misclassified textual units for each tool that can be potentially corrected by another tool. The second column ('Polarity Wrong') shows the polarity that is misclassified by the tool listed in the first column. The third to seventh column show how frequently each of the other tools can correct the misclassification of a tool listed in the first column. For example, for Senti4SD (i.e., S4SD in Table 3) the two unsupervised classifiers (SentistrengthSE and Opiner) offer the most complementarity (31% and 34%) to correct the positive polarity as well as the negative polarity. The last column (≥1) shows how the misclassification of a tool can be corrected by the correct classification of at least one of the tools. For example, 60% of the misclassified positive polarity by Senti4SD can be corrected by at least one of the other tools. The last row (≥ 1) shows the overall distribution of the positive, negative, and neutral units for which at least one tool is wrong. Around 87% of such misclassified positive polarity and around 82% of such negative polarity can be corrected by at least one of the other tools. The tool POME seems to be the most effective to correct the misclassifications of neutral units, i.e., when other tools incorrectly label a unit as non-neutral. Other tools (i.e., except POME) can complement more each other to correct the misclassifications for the positive and negative polarities. The primary

Table 4. Confusion Matrix for 3 Class Sentiment Detection

		Predicted		
		Positive (P)	**Negative (N)**	**Neutral (O)**
Actual	**Positive (P)**	TP_P	FP_N, FN_P	FP_O, FN_P
	Negative (N)	FP_P, FN_N	TP_N	FP_O, FN_N
	Neutral (O)	FP_P, FN_O	FP_N, FN_O	TP_O

challenge for a hybrid engine would be then to make a final decision from the different polarity labels on a given unit.

3.2 RQ_2 Can a Majority Voting-based Classifier Perform Better Than Individual Classifiers?

3.2.1 Motivation. The findings from RQ_1 confirm that the sentiment detectors can complement each other to correct their polarity misclassifications. However, the challenge is to determine the right polarity labels when multiple tools offer different labels. Given that we analyze five SE-specific sentiment detectors, the majority of the classifiers may agree on the correct classification for a given unit.

3.2.2 Approach. We take as input the matrix $M = U \times F \times L$ that we created for RQ_1, where for each unit U_i we have polarity labels from each of the five sentiment detectors, i.e., $L_i = \{L_i^1, L_i^2, L_i^3, L_i^4, L_i^5\}$. As we noted in Section 3.1, this matrix was created by retraining the supervised classifiers on the benchmark datasets using a 10-fold cross validation. Out of the five polarity labels in L_i, we assign U_i the label that was observed the most in L_i. We report performance using three standard metrics of information retrieval: precision (P), recall (R), and F1-score ($F1$).

$$P = \frac{TP}{TP + FP},\ R = \frac{TP}{TP + FN},\ F1 = 2 * \frac{P * R}{P + R}$$

TP = # of true positives, FN = # of false negatives, TN = # of true negatives, and FP =# of false positives. In addition, we also report the weighted κ value for each tool by computing the agreement between the tool and the manual labels. We use the Python ml_metrics to compute the weighted κ values.

We do not report accuracy as a metric while reporting the performance of the tools due to the following reason. As we presented the descriptive statistics of our six datasets in Table 1 (Section 2), our datasets are imbalanced. This means that we do not have each polarity class present in equal proportion in a given dataset. For example, for the SO Lin et al. [30] dataset, only 21% of the units are either positive or negative polarity and the rest (i.e., 79%) are neutral. Overall, only 46% of all units across the six datasets are either positive or negative and the rest (i.e., 54%) of the units are neutral. Therefore, we are analyzing highly skewed (i.e., imbalanced) datasets. Metrics like accuracy can give misleading information for imbalanced datasets. For example, if we label each unit as neutral, the overall accuracy across the datasets would have been 0.46. However, given the accurate detection of negative/positive polarity is more important in SE, the accuracy would have given us a false impression of the effectiveness of the model. Indeed, in machine learning literature and discussions, it is warned to not use accuracy as a measure for imbalanced datasets.[2]

To assess performance, we use the confusion matrix in Table 4, which is consistent with the state of the art [5, 43, 53, 63]. We report overall performance using both micro and macro averages, as well as by each polarity class. Macro average is useful to emphasize the performance

[2]see https://machinelearningmastery.com/failure-of-accuracy-for-imbalanced-class-distributions/.

Table 5. Performance Comparison of Majority Voting-based Hybrid Classifiers with the Baselines

		Macro			Micro		
	κ	**F1**	**P**	**R**	**F1**	**P**	**R**
Majority-All	0.56	0.746	0.806	0.694	0.768	0.768	0.768
Majority-Supervised	0.61	0.745	0.811	0.690	0.769	0.769	0.769
Majority-Unsupervised	0.42	0.630	0.715	0.563	0.671	0.671	0.671
Baseline - **Senti4SD**	0.63	0.750	0.768	0.732	0.774	0.774	0.774
Baseline - **SentiCR**	0.63	0.756	0.791	0.723	0.780	0.780	0.780
Baseline - **SentistrengthSE**	0.60	0.723	0.746	0.701	0.748	0.748	0.748
Baseline - **Opiner**	0.35	0.563	0.571	0.556	0.597	0.597	0.597
Baseline - **POME**	0.04	0.397	0.451	0.354	0.540	0.540	0.540

Performance analysis for each polarity class is provided in Table 22 of Appendix A.

on polarities with few instances (e.g., positive/negative in our case). In contrast, micro average is influenced mainly by the performance of the majority polarity (e.g., neutral), because it merely takes the average of each polarity. We use the F1-score to report the best tool, following standard practices [35, 43].

3.2.3 Results. In Table 5, we compare the performance of the developed majority-based voting detector against the stand-alone sentiment detectors. We show three versions of the majority-based detectors: (1) Majority-All: uses polarity outputs from all five sentiment detectors. (2) Majority-Supervised: uses polarity outputs from only the supervised sentiment detectors (i.e., Senti4SD and SentiCR), and (3) Majority-Unsupervised: uses polarity outputs from only the unsupervised sentiment detectors (i.e., POME, Opiner, and SentistrensghtSE). None the of above two majority-based detectors outperform the individual supervised detectors. The Majority-All detector shows an F1-score of 0.746 (Macro), while Senti4SD shows 0.750 and SentiCR 0.756. With regards to the Micro average, Majority-All shows 0.768, while Senti4SD shows 0.774 and SentiCR shows 0.780. The breakdown of the Majority-All into sub-components, i.e., Majority-Supervised and Majority-Unsupervised shows that the major contributors of success in the Majority-All are the supervised classifiers. For example, Majority-Supervised shows a Macro F1-score of 0.745 (Majority-All shows 0.746), while Majority-Unsupervised shows an F1-score of 0.630. The Majority-Unsupervised performs better than the two unsupervised detectors (Opiner and POME), but it is outperformed by the SentistrengthSE (Macro F1-score = 0.723). The results indicate a non-linear relationship between the correct polarity labels and the agreement among the tools when at least one of the tools can be correct.

3.3 RQ_3 What are the Misclassification Categories in the Complementary Cases?

3.3.1 Motivation. The findings from RQ_1 (Section 3.1) confirm the potential of developing a hybrid sentiment detector, RQ_2 (Section 3.2) shows that a simple majority-based voting does not perform better than the individual detectors. To better design a hybrid sentiment detector, we thus need to understand why one tool misclassifies when another tool can offer the correct label. This can be achieved by analyzing the error categories in the complementary cases.

3.3.2 Approach. We study misclassification types in 159 units from our complementarity dataset *R* (Section 3.1). This sample is statistically significant with a 99% confidence level (10 interval). Given that the majority of units are neutrals, a purely random sample would contain mostly neutral units. We thus randomly pick equal number (i.e., 53) of units from each polarity class. We manually label the misclassification reasons as follows. (1) We label the misclassification in the

Table 6. Error Categories in Complementarity Records

Category	Description	#
Context	Tool fails to understand underlying context required to determine the overall polarity	62
Polarity Diversity	The text contains multiple polarity cues with diverse interpretations	35
Domain	Polarity cues are SE-specific	23
General	Tool cannot process linguistic cues/typos.	22
Politeness	A neutral text is labeled polar due to the existence of politeness	17

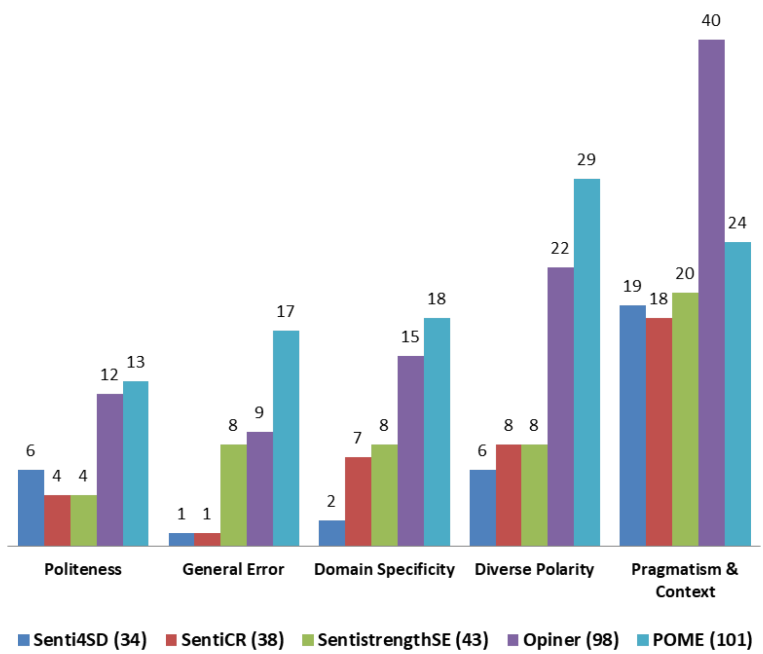

Fig. 1. Error categories in the complementary cases.

unit to one of the seven error categories from Novielli et al. [43]. The categories are: (a) Polar facts by neutral, (b) General error, (c) Politeness, (d) Implicit polarity, (e) Subjectivity in annotation, (f) Inability to deal with context information, and (g) Figurative language. (2) Novielli et al. [43] observed seven error categories in four sentiment detectors when they are all wrong on a given unit. In contrast, we aim to study the textual units of the misclassifications where at least one tool is correct. If we cannot match the misclassification to one of the seven categories, we assign it a new error category based on the analysis of the textual contents.

3.3.3 Results. We observed five error categories: three from [43] (Context/Pragmatics, General Error, and Politeness) and two new: Domain-specific polarity and presence of diverse polarity in the textual units. Table 6 and Figure 1 summarize the categories and their distributions in our analysis. We discuss the categories below.

(1) **Inability of classifiers to deal with pragmatics or context information.** We observe that the positive/negative polarity signals are mostly implicit or expressed through the status of a specific event (e.g., the task cannot be completed or the underlying API worked). The two supervised classifiers (Senti4SD and SentiCR) and SentistrengthSE performed better to detect such context-specific pragmatics. For example, only Senti4SD and SentiCR correctly assigned the following unit as negative in Jira (Ortu et al. [46]), because the developer mentions the absence of

certain features in their development context, *Maurice I don't have such option or maybe I don't know where it is.* Only Senti4SD, SentiCR, and SentistrengthSE correctly labeled the following unit as negative in SO Calefato et al. [5], because the tools detect that the developer is reporting that a tool feature is not working: *I need to pass a regex substitution as a variable: This, of course, doesn't work …* Opiner shows the most number of misclassifications followed by POME. Both tools are rule-based. POME relies on specific linguistic patterns, while Opiner counts the polar words. While SentistrengthSE is rule-based, it uses domain-specific contextual cues.

(2) **Diverse Polarity.** The tools can get confused when diverse polarized statements are made in a textual unit. For example, all the tools except Senti4SD wrongly classified this unit as neutral, *Seeing all the information and activity, it does look very constructive and perhaps it shouldn't have been closed so eagerly!* The presence of diversity of polarity in a textual unit can be confusing to the tools for two reasons: (a) The tools may consider all the polarized words to determine polarity, while the final polarity of the unit may be decided based on a subset of those words. For example, consider the following post which starts with criticism towards a tool, but ends with praises towards the tool, *this app is terrible i am in a good relationship and we are happy together.* (b) The underlying design of the tool does not allow the processing such diversity. For example, only POME misclassifies the following post, because it does not conform to any of its linguistic patterns [29], *Hey, I'm new to this site. I think it is great! Okay, here's the deal. I just downloaded Smule Ocarina. I was wondering how they made it so you can upload a song to the cloud. I might have an app idea that might incorporate this. How would I do this? What would I need?*

(3) **Domain specific polarity.** The expression of polarity can be unique to the domain of software engineering (e.g., the API is threadsafe) or differ from other domains (e.g., the API is simple to use vs the movie is too simple) [30, 43, 63]. The tools in our study are designed for different SE-domains, making them suitable to complement each other. Both Senti4SD and SentiCR label the following unit correctly as negative, *alright its okay but it freezes all the time* . Both Opiner and Senti4SD algorithms correctly label the negative polarity for the unit, *worried about IE compatibility!!.*

(4) **General Error.** The tools also differ from each other in how they process the textual contents and thus can misclassify a unit when the contents are not properly processed. For example, POME could not process the emoticons in the following unit while other tools were able to, *@BoristheSpider :(:(That is really sad. :(.* Opiner wrongly labeled the following unit as positive, because it could not properly determine the presence of negation *so I'm not happy with it.* . Opiner incorrectly labels the following unit as positive, *I also would like to see an answer to this...* The reason Opiner labels it as positive is due to the presence of 'like' in the unit, which it considered as positive. However, the presence of 'would like to' makes the expression more of an intention than of an opinion.

(5) **Politeness.** In two datasets (Jira Ortu et al. [46] and SO Calefato et al. [5]), the tools confused over the appropriate labeling of a unit that contained politeness. For example, Senti4SD and SentistrengthSE correctly label the following unit as positive, while other tools label it as neutral *"Thanks Arvind".*

In summary, the supervised detectors show lower error coverage across all the five categories. When the misclassification of a unit was due to the presence of diverse polarity, Senti4SD showed lower misclassification than SentiCR. However, when the underlying contextual information is required, SentiCR had lower misclassification rates. The underlying design of the two supervised classifiers offers some insights into this: Senti4SD computes polarity labels based on their presence in the selected parts of the unit, e.g., the first vs last sentiment word in a unit may carry different weights. In contrast, SentiCR relies specifically on a preprocessed **Bag of Words (BoW)** produced from a unit, e.g., negations are detected and explicitly annotated in the bag of words.

Table 7. Studied Features to Design the Hybrid Classifier

Features	Rationale
Polarity Labels from SE-specific detectors	The supervised classifiers can learn the underlying context and domain-specific pragmatics. The rule-based classifiers can employ domain-specific rules.
Bag of words	The vectorized representation of the bag words offers contextual information

3.4 RQ_4 Can a Supervised Ensemble of Detectors Perform Better Than Individual Detectors?

3.4.1 Motivation. The findings from RQ_1 (Section 3.1) and RQ_2 (Section 3.2) show that the SE-specific sentiment detectors can complement each other to correct the misclassfications, but an unsupervised majority voting-based detector fails to properly capture the complementarity cases. The analysis of misclassifications in RQ_3 (Section 3.3) shows that the failure of a tool to properly detect the underlying contextual information caused the most misclassifications. We also see that SentiCR offered better support for such cases. Therefore, an ensemble of the individual detectors in a supervised setting similar to SentiCR may provide more contextual information to a hybrid engine.

3.4.2 Approach. We developed a suite of supervised sentiment detectors using the features summarized in Table 7. These features are divided into two categories:

- Polarity labels from individual detectors: for each unit, we have five polarity labels, one from each of the five SE-specific sentiment detectors.
- Bag of words: for each unit, we produce a bag of words by applying preprocessing steps similar to SentiCR [1], which are as follows:
  - Removal of stop words (as SentiCR, we used stopwords from NLTK and those provided by SentiCR).
  - Identification and annotation of negation. For example, “This isn’t good” will be preprocessed as “This is NOT_good”.
  - Identification and expansion of contractions. For example, “let’s” will be expanded to “let us”.
  - Detection of emojis. For example, ‘%-(’ will be replaced by a placeholder ‘NegativeSentiment’.

For our analysis, we used the entire dataset used in RQ_1, i.e., all the 17,581 units. Also similar to RQ_1, we used a 10-fold cross validation. We investigated a total of nine algorithms, both ensemble and non-ensemble. From the ensemble algorithms, we trained and tested four classifiers: (1) **Random Forest (RF)**, (2) **Gradient Boosting Classifier (GBT)**, (3) Adaboost, and (4) XGBoost. Ensemble algorithms, like bagging and boosting, are designed to combine the predictions of several base estimators. For example, an RF classifier creates multiple decision-trees based on random samples of training data and then makes a final decision by consulting the decisions of the individual trees. Each algorithm is widely used in machine learning. Models, like RF, are robust against overfitting and they work well with both categorical and numerical data, which is useful in our case because, as features, we consider both vectorized textual contents from an input unit and the sentiment polarity labels about the unit from stand-alone tools. We use the Python Sklearn API to experiment with the models. Following the Sklearn documentation, we obtain deterministic behavior during model fitting by setting a fixed integer to define the random state of a model. For example, we set random_state = 45 for the Random Forest model.[3]

[3] https://scikit-learn.org/stable/modules/generated/sklearn.ensemble.RandomForestClassifier.html.

From the non-ensemble algorithms, we trained and tested five classifiers: (1) Logistic Regression (family = 'Multinomial'), (2) SVM, (3) LASSO, (4) Naïve Bayes (Bernoulli), and (5) From neural network category, the **Multi-layer Perceptron (MLP)**. Algorithms like SVM are used to design hybrid sentiment detection tools in other domains, like sentiment analysis in Twitter data [12]. Logistic Regression, LASSO, MLP, and Naïve Bayes are used in almost all kinds of regression and classification analysis. Naïve Bayes is found to offer reasonably good performance in sentiment detection in other domains (e.g., movie reviews) [69].

Following standard principles of machine learning, for each algorithm, we hyper tuned it on the six benchmark datasets under different configurations and picked the configurations that offered the best performance. We use the python scikit-learn GridSearchCV function for our hypertuning, which takes a list of configuration parameters and a range a values for each parameter to determine the best parameter values. For each algorithm, we studied all possible parameters. For example, we ran Random Forest algorithm by varying parameter values, such as number of decision trees, maximum tree depth, minimum number of leafs, etc. Similarly, for GBT we also tried changing the learning rate, and so on.

3.4.3 Results. Out of the nine supervised algorithms we trained and tested in our datasets, the Random Forest-based classifier performed the best based on both Macro and Micro F1-scores. This finding is not surprising, given Random Forest is by design an ensemble algorithm and it is found to be a good predictor in other SE-specific tasks (e.g., defect prediction [37]). We name the best performing Random Forest (RF) model in our analysis 'Sentisead$_{RF}$'. We ran Sentisead on two settings:

- Sentisead$_{RF-B}$: Using all the features described in Table 7.
- Sentisead$_{RF-N}$: Without using the bag of words in Table 7. That means, this classifier only uses five feature values for a given unit, each corresponding to a polarity label for the unit from an individual detector.

In Table 8, we compare the performance of Sentisead$_{RF-B}$ and Sentisead$_{RF-N}$ against the individual sentiment detectors as well as the Majority-All classifier from RQ$_2$ (Section 3.2). The comparison shows the percentage increase in performance in Sentisead for a given metric over a baseline. For example, Sentisead$_{RF-B}$ shows a 5.5% increase in F1-score Macro over Majority-All classifier, which is computed as $\frac{\text{F1 (Sentisead)}_{RF-B} - \text{F1 (Majority-All)}}{\text{F1 (Majority-All)}}$.

Overall, Sentisead$_{RF-B}$ perfoms better than Sentisead$_{RF-N}$. Therefore, the incorporation of bag of words as additional features in Sentisead$_{RF-B}$ improves the performance of sentiment detection in the hybrid engine. The Sentisead$_{RF-B}$ also outperforms the stand-alone sentiment detectors, which it combines. The increase in performance varies across the tools and the metrics. For F1-score Macro, Sentisead$_{RF-B}$ offers 4.1–98.2% increase in performance over the individual detectors. For F1-score Micro, the hybrid engine offers 4–49% increase in performance over those detectors. The increase in performance is significant over two unsupervised detectors (Opiner, POME). Across the supervised and unsupervised individual detectors, Sentisead$_{RF-B}$ offers better increase in performance to detect positive and negative polarity instead of neutral polarities. For example, excluding POME, Sentisead$_{RF-B}$ offers 4.5–54.7% increase in performance to detect positive polarity (F1-score) and 7–44% increase in performance to detect negative polarity. For POME, Sentisead$_{RF-B}$ offers more than 500% increase in performance, due mainly to the fact that POME classifies more than 90% of units as neutral. POME is designed to rely on specific linguistic rules to detect polarity aspects (e.g., performance), which when the tool identified such aspects, it was found to be very accurate. However, due to its focus on such precision, it also missed correct polarity classes where those patterns were not present.

Table 8. Performance Comparison of the Developed Hybrid Classifier (Sentisead) with the Baselines

		Macro			Micro		
	κ	**F1**	**P**	**R**	**F1**	**P**	**R**
Sentisead$_{RF-B}$	0.69	0.787	0.816	0.759	0.805	0.805	0.805
Sentisead$_{RF-N}$	0.67	0.774	0.792	0.757	0.795	0.795	0.795
Macro/Micro Performance Increase of Sentisead$_{RF-B}$ (i.e., Hybrid with Bag of Words) Over Baseline (SentiSE = SentistrengthSE)							
Majority	0.61	5.5%	1.2%	9.4%	4.8%	4.8%	4.8%
Senti4SD	0.63	4.9%	6.2%	3.7%	4.0%	4.0%	4.0%
SentiCR	0.63	4.1%	3.2%	5.0%	3.2%	3.2%	3.2%
SentiSE	0.60	8.9%	9.4%	8.3%	7.6%	7.6%	7.6%
Opiner	0.35	39.8%	42.9%	36.5%	34.8%	34.8%	34.8%
POME	0.04	98.2%	80.9%	114.4%	49.1%	49.1%	49.1%

Performance analysis for each polarity class is provided in Table 23 of Appendix A.

Table 9. Distribution of Misclassification Error Categories by Sentisead$_B$ on Dataset and Categories from Table 6

Category	**Sentisead$_B$**	**SentiCR**	**Senti4SD**	**SentistrengthSE**	**Opiner**	**POME**
Context (62)	20%	25%	34%	30%	64%	38%
Polarity (35)	23%	26%	20%	20%	63%	83%
Domain (23)	9%	35%	4%	30%	65%	78%
General (22)	5%	9%	5%	14%	41%	77%
Politeness (17)	24%	18%	35%	18%	76%	82%

Table 10. Features Used in Sentisead+ in Addition to Table 7

Features	**Rationale**
Partial polarity	The polarity label may depend more on the first/last sentence of a unit.
Polarity entropy	The information entropy of polarity may reflect diverse polarity.

In Table 9, we show the coverage of the misclassification categories on the datasets of Table 6 from RQ$_3$ (Section 3.3). Our hybrid engine Sentisead performs the best to reduce the misclassifications related to contextual information (20% in Sentisead$_{RF-B}$ vs 25% in SentiCR). Given that more than one third of the misclassifications are due to the failure of the tools to detect the contextual/pragmatics in the sentiment carrying expression, this particular focus of Sentisead$_{RF-B}$ is thus encouraging. Sentisead$_{RF-B}$ also offers performance boost over SentiCR while correcting errors related to 'Domain specificity' (9% vs 35%). Thus, the incorporation of bag of words as features helped Sentisead$_{RF-B}$ to disentagle the contextual information during polarity labeling. Overall, Sentisead$_{RF-B}$ shows lower coverage of the misclassifications over the unsupervised (except for 'Diverse Polarity' in SentistrengthSE). However, for the two misclassification categories ('Diverse Polarity' and 'Domain Specificity'), Sentisead still has slightly higher coverage than Senti4SD. For example, for units misclassified due to 'Diverse Polarity', Senti4SD shows 20% coverage while Sentisead$_{RF-B}$ shows 23% coverage. The cases related to 'Politeness' are due mainly to the heterogeneity in the labeling of the datasets, which we discuss in Section 5.

3.5 RQ_5 Can More Features Further Improve the Performance of the Hybrid Detector?

3.5.1 Motivation. The findings from RQ_4 (Section 3.4) confirm that Sentisead$_{RF-B}$, a supervised mode using Random Forest (RF) as an ensembler of stand-alone sentiment detectors can offer improved performance over the individual detectors. A closer look at the corrected misclassifications shows that the improvement occurred mainly on the understanding of underlying contextual information (see Table 9). This is not surprising, given features used in the hybrid engine were the polarity labels from individual detectors and the vectorized representation of units as bag of words. As shown in Table 9, Sentisead$_{RF-B}$ offers improved performance across the tools, but Senti4SD still slightly outperforms Sentisead$_{RF-B}$ while classifying units with diverse polarity and domain specificity. Given that the misclassifications due to the diverse polarity were second most in our manual analysis, it would be important to see if Sentisead$_{RF-B}$ (denoted as Sentisead$_{RF}$ hereon) can be further improved to handle those cases.[4]

3.5.2 Approach. In addition to the features in Table 7 (i.e., polarity labels from SE-tools + Bag of words), we investigate two new categories of features:

- *Partial Polarity.* We label a unit based on the polarity information we find in the first or last clause/sentence of the unit. We decided to use this feature based on the observations that sometimes the final polarity label of a given input unit depends more on the first or the last polarity expression in the unit. Similar features were also previously used to develop Senti4SD [5].
- *Polarity Entropy.* We compute entropy of a given unit following principles of information theory. We were motivated to use entropy as a feature, because entropy measures were previously found to be useful in various software engineering classification tasks, such as predict/triage software faults [19, 25]. Intuitively, entropy can inform us of the underlying diversity of polarity/words in the input unit, which we hoped could be useful to make a better decision on the overall polarity label of the unit.

The new features are summarized in Table 10. We explain those below.

(1) **Partial Polarity.** The relative position of a polarity word in a unit can be important to correct a misclassification. For example, ‘I like this tool. But it is slow’ could convey a negative polarity even though the first sentence is positive and the second sentence is negative. Given as input a unit, we first detect sentences using NLTK sentence detector. We then determine two polarity labels: (a) First Polarity. We label the unit the polarity of the first sentence. (b) Last Polarity. We label the unit the polarity of the last sentence.

(2) **Polarity Entropy.** If one tool is wrong but another is right, then there is some *uncertainty* in the detection process. To quantify such uncertainty, we use *Shannon Entropy* from Information Theory [54], which was previously used to predict/triage software faults [19, 25]. Shannon Entropy measures the information associated with a given input:

$$H_n(P) = -\sum_{k=1}^{n}(p_k * \log_e p_k) \tag{1}$$

where p_k is the probability of a data value in a given unit and $\sum_{i=1}^{k} p_k = 1$. Suppose we are monitoring an input text to categorize (e.g., polarity). The text has two words A and B. Using Equation (1), the entropy value of the text would be 0.69. However, if the next word of text is again B, the entropy

[4] For brevity, we will refer to Sentisead$_{RF-B}$ as Sentisead$_{RF}$ in the rest of the paper.

Table 11. Performance Comparison of the Designed Hybrid Detector (Sentisead) with Added Features from Table 10

		Macro			Micro		
	κ	**F1**	**P**	**R**	**F1**	**P**	**R**
Sentisead$_{RF-B}$**+**	0.69	0.788	0.815	0.763	0.806	0.806	0.806
Sentisead$_{RF-BNE}$**+**	0.69	0.786	0.817	0.757	0.804	0.804	0.804
Sentisead$_{RF-BNP}$**+**	0.68	0.786	0.812	0.761	0.804	0.804	0.804
SentiSead$_{RF-N}$**+**	0.65	0.757	0.77	0.744	0.779	0.779	0.779
Sentisead$_{RF-NNE}$**+**	0.66	0.768	0.786	0.751	0.791	0.791	0.791
Sentisead$_{RF-NNP}$**+**	0.65	0.756	0.769	0.744	0.779	0.779	0.779

Sentisead$_{RF-B}$*: With bag of words. Sentisead$_{RF-N}$*: No bag of words. NP = No partial polarity. NE = No polarity entropy.

Performance analysis for each polarity class is provided in Table 24 of Appendix A.

value drops to 0.63 because the uncertainty level has decreased: we see more similar contents (i.e., two Bs, one A). For each unit, we compute following metrics.

(a) *Polarity* $_{Entropy}$: We identify all the polarity lexicons in the unit and develop a data structure of the form $\{(w_1, f_1), \ldots (w_n, f_n)\}$, where each w_i is a word in the unit that corresponds to a sentiment word and f_i corresponds to its occurrence frequency in the unit. We then use Equation (1) to compute the entropy. For example, with the sentence "The API is great, but it's slow", if we only use the occurrence of polarity words in the sentence into Equation (1), we have the following data values: {'great': 1, 'slow': 1}. The entropy value would be 0.69.

(b) *Adjective* $_{Entropy}$: We compute the *diversity* of all adjectives in a given unit. A polarity word may be missing in our sentiment lexicons, but may still be found in the adjectives.

(c) *Verb* $_{Entropy}$: We compute the *diversity* of all verbs in a unit using Equation (1) similarly as we did with adjectives. Ko et al. [26] observed that developers use verbs to describe software problems (e.g., "This tool does not work").

3.5.3 Results. Sentisead$_{RF-B+}$ offers a very marginal increase in performance over Sentisead$_{RF}$ (0.001 increase in Macro F1-score). Therefore, the inclusion of the new features into Sentisead$_{RF}$ does not seem effective to further improve its performance over the stand alone detectors. To further understand the impact of the new features, in Table 11, we report the performance of Sentisead$_{RF}$ with the additional features in six different settings:

(1) Sentisead$_{RF-B}$+: with features bag of words + polarity labels from SE-tools + partial polarity + polarity entropy.
(2) Sentisead$_{RF-BNP}$+: with features bag of words + polarity labels from SE-tools + polarity entropy.
(3) Sentisead$_{RF-BNE}$+: with features bag of words + polarity labels from SE-tools + partial polarity.
(4) Sentisead$_{RF-N}$+: with features polarity labels from SE-tools + partial polarity + polarity entropy.
(5) Sentisead$_{RF-NNP}$+: with features polarity labels from SE-tools + polarity entropy.
(6) Sentisead$_{RF-NNE}$+: with features polarity labels from SE-tools + partial polarity.

The models with bag of words perform better than those without bag of words as features. Both Sentisead$_{RF-BNP}$+ (i.e., without partial polarity) and Sentisead$_{RF-BNE}$+ (i.e., without polarity entropy) show a Macro F1-score 0.786, but they differ from each other based on precisions and

Table 12. Error Categories Coverage by Sentisead+ Variations

Category	B+	BNE+	BNP+	N+	NNE+	NNP+
Context (62)	18%	21%	18%	26%	25%	26%
Polarity (35)	23%	26%	26%	23%	17%	29%
Domain (23)	9%	9%	9%	9%	9%	13%
General (22)	5%	5%	5%	9%	9%	9%
Politeness (17)	24%	24%	24%	24%	24%	24%

B+: With bag of words. N+: No bag of words.
NP = No partial polarity. NE = No polarity entropy.

recalls. The precision is higher for Sentisead$_{RF-BNE}$+ between the settings, but the recall is higher for Sentisead$_{RF-BNP}$+. Therefore, the two features while combined with bag of words features, were able to slightly improve the overall performance than Sentisead$_{RF}$.

In Table 12, we show the distribution of error categories across the six settings. Overall, the inclusion of the new features has reduced the coverage of errors due to contextual information from 20% (Sentisead$_{RF}$) to 18% (Sentisead$_{RF-B}$+). There is no impact of the new features on the other four error categories. A closer look at the settings, however, shows that the partial polarity features were successful to reduce errors related to diverse polarity. For example, Sentisead$_{RF-NNE}$+ shows a coverage of 17% for errors related diverse polarity. This setting is trained using only the polarity labels from the five SE-tools and the partial polarity labels. However, those corrections were ignored in Sentisead$_{B}$+ tool, which shows a coverage of 23% of errors related to diverse polarity. Thus, Sentisead$_{RF-B}$+ gave more preference over bag of words features than partial polarity features while making the final decisions on a polarity label. For example, the following unit was correctly labeled as positive by Sentisead$_{RF-NNE}$+ (due to the first sentence being positive), but was incorrectly labeled as neutral by Sentisead$_{RF-B}$+: *Looks excellent. Really sweet. … [4 more sentences] … + Can you add a note on how you've changed how REST works high-level? Thats all for now.*

4 STUDY PHASE 2: ENSEMBLE WITH DEEP LEARNING MODELS

In a second phase, we answer three more research questions introduced in Section 1:

(6) Can Sentisead outperform the BERT-based advanced pre-trained language based models? (Section 4.1)
(7) Can a deep learning model for Sentisead outperform the BERT-based models? (Section 4.2)
(8) Can Sentisead based on an ensemble of all the individual models offer the best performance of all tools? (Section 4.3)

4.1 RQ$_6$ Can Sentisead$_{RF}$ Outperform the BERT-based Advanced Pre-trained Language Based Models?

4.1.1 Motivation. As we explained in Section 1, we started our analysis of the feasibility of developing a hybrid tool of SE-specific sentiment detectors before deep learning models for sentiment detection in SE were described. Recent studies show that deep learning models, specially language-based pre-trained transformer models (PTMs) can be trained using SE-specific data to offer performance superior to the traditional shallow learning models [3, 74].

The most recent paper on this topic is by Zhang et al. [74], who reported that models like BERT and RoBERTa can outperform stand-alone rule-based and shallow learning models (e.g., Senti4SD,

Table 13. Performance of Stand-alone Advanced Pre-trained Advanced Language-based Deep Learning Models Across the Datasets

		Macro			Micro		
	κ	**F1**	**P**	**R**	**F1**	**P**	**R**
BERT	0.70	0.793	0.793	0.792	0.81	0.81	0.81
Macro/Micro Change Over Sentisead$_{RF}$	–	0.8%	−2.8%	4.3%	0.6%	0.6%	0.6%
RoBERTa	0.71	0.801	0.799	0.804	0.815	0.815	0.815
Macro/Micro Change Over Sentisead$_{RF}$	–	1.8%	−2.1%	5.9%	1.2%	1.2%	1.2%
ALBERT	0.69	0.787	0.795	0.78	0.806	0.806	0.806
Macro/Micro Change Over Sentisead$_{RF}$	–	0.0%	−2.6%	2.8%	0.1%	0.1%	0.1%
XLNet	0.70	0.799	0.797	0.8	0.814	0.814	0.814
Macro/Micro Change Over Sentisead$_{RF}$	–	1.5%	−2.3%	5.4%	1.1%	1.1%	1.1%

Performance analysis for each polarity class is provided in Table 25 of Appendix A.

SentiCR). Therefore, we must analyze whether such PTMs could also outperform Sentisead, i.e., our hybrid of stand-alone rule-based and shallow learning sentiment detection tools.

4.1.2 Approach. We investigate all the four PTMs from Zhang et al. [74], i.e., BERT, RoBERTa, ALBERT, and XLNet. For each of the studied datasets, we train and test the performance of each model using the same 10-fold cross-validation that we used to answer RQ$_1$ (see Section 3.1.2). Similar to Zhang et al. [74], we also fine-tune the hyperparameters for the PTMs. Following Delvin et al. [11], we investigate with the following hyperparameters: (1) Batch size, (2) Number of epochs, and (3) Learning rate. For all the PTMs, we find the following parameters after fine-tuning: batch size of 16, 4 epochs, and a learning rate of 2^{-5}. The values are similar to those reported by Zhang et al. [74].

4.1.3 Results. In Table 13, we present the performance of the four stand-alone PTMs across the six benchmark datasets. For each model, we also show how the performance of the model improved/decreased compared to Sentisead$_B$ from RQ$_4$ (i.e., the hybrid of the five rule-based and shallow learning models based on Random Forest).

In terms of Macro F1-score, all the PTMs offer slightly better performance than Sentisead$_B$. The increase in Macro F1-score in the PTMs compared to Sentisead$_B$ is due to the increase in recall but not in precision. RoBERTa shows the best Macro F1-score of 0.801, which outperforms Sentisead$_B$ by 1.8%. It also shows the best recall of 0.804, which outperforms Sentisead$_B$ by 5.9%. It also has the best precision of 0.799 among the four PTMs, which is 2.1% less than the Macro precision of Sentisead$_B$.

A closer look at the class-based performance shows that the drop in precision is 7.1% in RoBERTa compared to Sentisead$_B$ for both positive and negative polarity classes, while the increase in recall is 13.9% and 15.3% for the positive and negative classes, respectively. Thus, with PTMs like RoBERTa, we can expect to find more instances of polarity classes compared to Sentisead$_B$, but that also contributes to a loss of precision. In terms of F1-score, RoBERTa outperforms Sentisead$_B$ for all polarity by 0.5% (neutral), 3.1% (positive), and 4.1% (negative).

Table 14 shows the distribution of error categories by the four PTMs in the dataset and categories from Table 6. Among the four PTMs, BERT has the lowest number of errors related to contexts/pragmatics (16%) and politeness (18%), while RoBERTa has the lowest number of errors for two other categories (diverse polarity and domain specificity, both 9%). Both BERT and XLNet show the lowest number of errors under the general category (9%). The last column in Table 14 shows the distribution of error categories in the same dataset by Sentisead$_B$ (taken from Table 9).

Table 14. Distribution of Error Categories by Advanced Pre-trained Language-based Models on Dataset and Categories from Table 6

Category	BERT	RoBERTa	Albert	XLNet	Sentisead_B
Context/Pragmatics (62)	16%	21%	21%	26%	20%
Diverse Polarity (35)	11%	9%	17%	17%	23%
Domain Specificity (23)	9%	9%	13%	13%	9%
General Error (22)	9%	14%	14%	9%	5%
Politeness (17)	18%	24%	24%	24%	24%

Overall, the best performing PTM (in terms of Macro F1-score), RoBERTa improves on Sentisead_B in this dataset in only one of the five error categories, diverse polarity (23% in Sentisead_B vs. 9% in RoBERTa). For example, RoBERTa corrects the following misclassification of Sentisead_B from neutral to positive, *Seeing all the information and activity, it does look very constructive and _perhaps_ it shouldn't have been closed so eagerly!*. RoBERTa also corrects this misclassification of Sentisead_B from neutral to negative: *The problem is, the application is developed in Java not python* (in Lin et al. [30] app review dataset).

Therefore, when we can afford to use GPUs that are required to develop the PTMs and when we need even slight performance improvement like 1.8% (Macro F1-score) over a hybrid of shallow and rule-based models in Sentisead_B, we could use the PTMs. Otherwise, Sentisead_{RF} might be sufficient to meet real-world needs for sentiment detection in SE.

4.2 RQ_7 Can a Deep Learning Model for Sentisead Outperform the BERT-based Models?

4.2.1 Motivation. As we observed in RQ_6, the stand-alone PTMs slightly outperform Sentisead_B, i.e., the Random Forest-based hybrid of five rule-based and shallow learning SE-specific sentiment detectors (Opiner, POME, Senti4SD, SentiCR, and SentistrengthSE). The PTMs offer better recall than Sentisead_B, but suffers from lower precision. The current ensembler in Sentisead_B is a Random Forest, which is a shallow learning model. Therefore, we investigate whether we could further improve the performance of Sentisead_B by introducing a PTM as ensembler.

4.2.2 Approach. Our goal is to replace the Random Forest algorithm in Sentisead_B by the PTMs from RQ_6, as follows. For each studied dataset, we do a 10-fold cross validation using a PTM following an approach similar to RQ_1 in Section 3.1.2. Each iteration takes as input the following features for each unit: bag of words and sentiment polarity labels of the five rule-based and shallow models (i.e., OpinerPOME, Senti4SD, SentiCR, and SentistrengthSE). The target value is the manual label for each unit. Instead of Random Forest as ensembler, we use one of the PTMs. Therefore, for each studied dataset, we do a total of $4 \times 10 = 40$ runs, 10 for each of the PTMs and therefore $6 \times 40 = 240$ iterations. We call the new ensemble models Sentisead_{BERT}, $\text{Sentisead}_{RoBERTa}$, $\text{Sentisead}_{ALBERT}$, and Sentisead_{XLNet} when using BERT, RoBERTa, ALBERT, and XLNet, respectively.

4.2.3 Results. Table 15 shows the performance of the four new PTM-based ensemblers $\text{Sentisead}_{RoBERTa}$, Sentisead_{BERT}, $\text{Sentisead}_{ALBERT}$, and Sentisead_{XLNet}. It also reports how their performances are different from those of Sentisead_B and the stand-alone PTMs.

In terms of Macro F1-score, $\text{Sentisead}_{RoBERTa}$ is the best performer with an F1-score of 0.805. It outperforms Sentisead_B by 2.3% and the stand-alone RoBERTa model by 0.5%. The second best performer is Sentisead_{XLNet} with a Macro F1-score of 0.8, which outperforms Sentisead_B by 1.7% and the stand-alone XLNet by 0.1%. The other two ensemblers, Sentisead_{BERT} and $\text{Sentisead}_{ALBERT}$ perform worse than the corresponding stand-alone models (i.e., BERT and ALBERT, respectively).

Table 15. Performance of Sentisead Using Pre-trained Language Based Advanced Deep Learning Models as Ensembler

		Macro			Micro		
	κ	**F1**	**P**	**R**	**F1**	**P**	**R**
Sentisead$_{BERT}$	0.69	0.791	0.791	0.79	0.807	0.807	0.807
Macro/Micro Change Over Sentisead$_{RF}$	–	0.5%	−3.1%	4.1%	0.2%	0.2%	0.2%
Macro/Micro Change Over BERT	–	−0.3%	−0.3%	−0.3%	−0.4%	−0.4%	−0.4%
Sentisead$_{RoBERTa}$	0.72	0.805	0.807	0.803	0.82	0.82	0.82
Macro/Micro Change Over Sentisead$_{RF}$	–	2.3%	−1.1%	5.8%	1.9%	1.9%	1.9%
Macro/Micro Change Over RoBERTa	–	0.5%	1.0%	−0.1%	0.6%	0.6%	0.6%
Sentisead$_{ALBERT}$	0.68	0.779	0.779	0.779	0.798	0.798	0.798
Macro/Micro Change Over Sentisead$_{RF}$	–	−1.0%	−4.5%	2.6%	−0.9%	−0.9%	−0.9%
Macro/Micro Change Over ALBERT	–	−1.0%	−2.0%	−0.1%	−1.0%	−1.0%	−1.0%
Sentisead$_{XLNet}$	0.71	0.8	0.799	0.801	0.815	0.815	0.815
Macro/Micro Change Over Sentisead$_{RF}$	–	1.7%	−2.1%	5.5%	1.2%	1.2%	1.2%
Macro/Micro Change Over XLNet	–	0.1%	0.3%	0.1%	0.1%	0.1%	0.1%

Performance analysis for each polarity class is provided in Table 26 of Appendix A.

Table 16. Error Categories by Sentisead Based on Advanced Pre-trained Language-based Models on Dataset from Table 6

Category	**Sentisead**$_{BERT}$	**Sentisead**$_{RoBERTa}$	**Sentisead**$_{ALBERT}$	**Sentisead**$_{XLnet}$
Context/Pragmatics (62)	21%	23%	33%	30%
Diverse Polarity (35)	17%	14%	4%	11%
Domain Specificity (23)	9%	4%	17%	4%
General Error (22)	9%	9%	14%	9%
Politeness (17)	12%	29%	18%	24%

Similar to the stand-alone PTM, RoBERTa, the increase in F1-score of Sentisead$_{RoBERTa}$ over Sentisead$_B$ is due to a higher recall. However, Sentisead$_{RoBERTa}$ does have better precision than RoBERTa: the incorporation of polarity labels from the five rule-based and shallow learning models has reduced the loss of precision from Sentisead$_B$ from 7.1% (in RoBERTa) to 6.2% in Sentisead$_{RoBERTa}$.

Table 16 reports the distribution of error categories per ensembler PTMs on the dataset and categories from Table 6. The best Sentisead$_{RoBERTa}$ shows a reduction from stand-alone RoBERTa in misclassifications for two error categories (domain specificity from 9% to 4% and general error from 14% to 9%). For the other three error categories, the performance of Sentisead$_{RoBERTa}$ is actually lower than the stand-alone RoBERTa mode. For example, for errors related to lack of context/pragmatics, Sentisead$_{RoBERTa}$ shows the error in 23% of all cases. In contrast, stand-alone RoBERTa shows the error in 21% of all such cases. This means that the ensemble model was slightly more confused to correct such errors compared to the stand-alone model. This confusion could arise in the ensembler from the polarity labeling of stand-alone tools, which themselves can be confused with the exact polarity labeling of a sentence. This can happen for different reasons like the underlying context was implicit, or it was not annotated properly during the manual labeling. Indeed, subjectivity in annotation can be a problem with SE-specific datasets (see Novielli et al. [43, 63]). A detailed understanding of the exact reasons of performance drop in a deep learning ensembler compared to the stand-alone PTMs requires the analysis of the PTMs and the ensemblers based on the principles of explainable/interpretable machine learning, which we leave as our future work. Nevertheless, it is arguable whether a performance improvement of 0.5% over stand-alone PTM does warrant a PTM-based ensembler over a stand-alone PTM.

4.3 RQ_8 Can Sentisead Based on an Ensemble of All the Individual Models Offer the Best Performance?

4.3.1 Motivation. Our findings from RQ_6 show that stand-alone PTMs, like RoBERTa, can outperform $Sentisead_B$ based on shallow learning. Our findings from RQ_7 show that an ensemble of the five rule-based and shallow learning models based on PTMs, like $Sentisead_{RoBERTa}$, can offer 0.5% increase in Macro F1-score over the stand-alone RoBERTa model. Given the superiority of the stand-alone PTM over $Sentisead_B$, we want to learn whether an ensemble of all the available stand-alone models (i.e., rule-based, shallow, and PTMs) could offer the best performance of all the available ensemblers and stand-alone models.

4.3.2 Approach. We replicate the approach of RQ_7 for each dataset as follows. We perform a 10-fold cross validation similar to the approach in Section 3.1.2. For each unit in a dataset, we consider the following items as input features: textual contents of the unit as vectors in word embedding, sentiment polarity labels from the five rule-based and shallow learning models, and the four PTMs. The target is the sentiment polarity label based on manual label. For the ensembler, we use one PTM at a time. As such, similar to RQ_7, for each PTM as an ensembler, we run 10 iterations per dataset. We report the performance of the ensemblers on the test dataset. The ensembler model is named after each PTM as follows: $Sentisead_{BERT}+$, $Sentisead_{RoBERTa}+$, $Sentisead_{ALBERT}+$, and $Sentisead_{ALBERT}+$.

4.3.3 Results. Table 17 reports the performance of the four new PTM-based ensemblers that produce the sentiment polarity label for a unit by taking as input the unit and the polarity labels from eight stand-alone sentiment detectors for SE (rule-based, shallow-learning, and four PTMs). For each model, like $Sentisead_{BERT}+$, we also compare how its performance has increased/decreased against previous tools like $Sentisead_B$, stand-alone PTM (i.e, BERT), and the ensembler from RQ_7 (i.e., BERT-based ensembler of five rule-based and shallow learning models).

The best performing model is $Sentisead_{RoBERTa}+$, with a Macro F1-score of 0.8. This F1-score is lower than the F1-score of $Sentisead_{RoBERTa}$, which is 0.805. The inclusion of polarity labels from the PTMs has not increased the performance of $Sentisead_{RoBERTa}+$. However, $Sentisead_{RoBERTa}+$ does show an increase of 1.7% in Macro F1-score over $Sentisead_B$. Overall, among the four ensemblers in this RQ, only $Sentisead_{ALBERT}+$ shows a Macro F1-score lower than $Sentisead_B$. These findings are consistent with those from RQ_7 and RQ_6 and recent results from Zhang et al. [74], which show that deep learning models offer better performance than rule-based and shallow learning models for SE-specific sentiment detection.

However, as shown in RQ_6 to RQ_7, the increase in performance is at most 2.3% over the shallow learning-based ensembler (which we observed in $Sentisead_{RoBERTa}$). In addition, the lack of performance improvement in PTM-based ensembler of all stand-alone models in this RQ can serve as a guidance that a hybrid tool cannot outperform other tools by simply adding polarity labels from all available stand-alone SE-specific sentiment detectors. Table 18 shows the distribution of error categories of the PTM-based ensemblers.

5 DISCUSSIONS

In this section, we discuss major themes we observed during the analysis of our study results.

5.1 The Underwhelming Roles of Polarity Entropy Metrics in Sentiment Labeling

The findings from RQ_5 suggest that the three polarity entropy metrics were not effective to further improve the performance of our hybrid sentiment detection engine. To determine the root cause

Table 17. Performance of Sentisead Using Pre-trained Advanced Deep Learning Models as Ensembler and Stand-alone Detectors

		Macro			Micro		
	κ	**F1**	**P**	**R**	**F1**	**P**	**R**
Sentisead$_{BERT}$+	0.69	0.79	0.79	0.79	0.807	0.807	0.807
Macro/Micro Change Over Sentisead$_{RF}$	–	0.4%	−3.2%	4.1%	0.2%	0.2%	0.2%
Macro/Micro Change Over Sentisead$_{BERT}$	–	−0.1%	−0.1%	0.0%	0.0%	0.0%	0.0%
Macro/Micro Change Over BERT	–	−0.4%	−0.4%	−0.3%	−0.4%	−0.4%	−0.4%
Sentisead$_{RoBERTa}$+	0.71	0.8	0.797	0.802	0.814	0.814	0.814
Macro/Micro Change Over Sentisead$_{RF}$	–	1.7%	−2.3%	5.7%	1.1%	1.1%	1.1%
Macro/Micro Change Over Sentisead$_{RoBERTa}$	–	−0.6%	−1.2%	−0.1%	−0.7%	−0.7%	−0.7%
Macro/Micro Change Over RoBERTa	–	−0.1%	−0.3%	−0.2%	−0.1%	−0.1%	−0.1%
Sentisead$_{ALBERT}$+	0.68	0.783	0.784	0.782	0.801	0.801	0.801
Macro/Micro Change Over Sentisead$_{RF}$	–	−0.5%	−3.9%	3.0%	−0.5%	−0.5%	−0.5%
Macro/Micro Change Over Sentisead$_{ALBERT}$	–	0.5%	0.6%	0.4%	0.4%	0.4%	0.4%
Macro/Micro Change Over Albert	–	−0.5%	−1.4%	0.3%	−0.6%	−0.6%	−0.6%
Sentisead$_{XLNet}$+	0.70	0.797	0.795	0.799	0.811	0.811	0.811
Macro/Micro Change Over Sentisead$_{RF}$	–	1.3%	−2.6%	5.3%	0.7%	0.7%	0.7%
Macro/Micro Change Over Sentisead$_{XLNet}$	–	−0.4%	−0.5%	−0.2%	−0.5%	−0.5%	−0.5%
Macro/Micro Change Over XLNet	–	−0.3%	−0.3%	−0.1%	−0.4%	−0.4%	−0.4%

Performance analysis for each polarity class is provided in Table 27 of Appendix A.

Table 18. Error Categories by Sentisead Based on Ensemble of All Models and Dataset from Table 6

Category	**Sentisead$_{BERT}$+**	**Sentisead$_{RoBERTa}$+**	**Sentisead$_{ALBERT}$+**	**Sentisead$_{XLNet}$+**
Context/Pragmatics (62)	21%	26%	25%	25%
Diverse Polarity (35)	14%	14%	14%	11%
Domain Specificity (23)	9%	4%	9%	0%
General Error (22)	5%	14%	9%	14%
Politeness (17)	18%	18%	24%	24%

of this, we assessed the impact of the diversity metrics on the misclassification of the polarity labels from the three tools, Senti4SD, SentiCR, and SentistrengthSE. The three tools are the three most dominant features in Sentisead. For each tool, the response variable is “Misclassification”, i.e., for a given unit whether the tool output is wrong or not. The explanatory variables are the three diversity metrics above, i.e., entropy measurements using polarity, adjective, and verb. To fit and interpret the models, we follow standard practices in the literature [58, 67]: (1) When fitting the models, we test for multicollinearity between the explanatory variables using the **Variable Inflation Factor (VIF)**, and remove variables with VIF scores above the recommended maximum of 5 [10]. (2) When interpreting the models, we consider coefficient importance if they are statistically significant ($p-value \leq 0.05$). We also estimate their effect sizes based on ANOVA type-2 analyses (column “LR-Chisq” of Table 19). (3) We report the goodness of fit using the McFadden’s pseudo R^2. In Table 19, we show the results of regression models of each of the three tools for the inputs for which the tool was wrong but at least one of the other two tools was right, i.e., the wrong tool could have benefited from the correct classifications from other tools. The $Polarity_{Entropy}$ is statistically significant for each tool, i.e., the misclassification of each tool increases with the increase in the diversity of polarity words. The $Adjective_{Entropy}$ is significant for SentiCR and $Verb_{Entropy}$ is significant for SentistrengthSE. However, the R^2 is only between 3.06-3.86%. That means that the

Table 19. Regression Models for the Sentiment Detectors on the Misclassified Units Where At Least One Detector was Right

	Senti4SD Pseudo R^2 = 3.06%			**SentiCR** Pseudo R^2 = 3.86%			**SentistrengthSE** Pseudo R^2 = 3.74%		
	Coeffs	**Std. Err**	**LR Chisq**	**Coeffs**	**Std. Err**	**LR Chisq**	**Coeffs**	**Std. Err**	**LR Chisq**
(Intercept)	2.66***	0.27		−2.77***	0.27		−1.64***	0.22	
Polarity $_{Entropy}$	0.45***	0.11	25.72***	0.52***	0.11	36.15***	0.64***	0.10	75.16***
Adjective $_{Entropy}$	0.01	0.11	0.01	0.23*	0.11	4.57*	0.01	0.10	0.00
Verb $_{Entropy}$	0.01	0.12	0.11	−0.17	0.12	1.50	0.22*	0.10	6.07*

Signif. codes: $^{***}p < 0.001$, $^{**}p < 0.01$, $^{*}p < 0.05$.

entropy metrics can only capture around 3–4% of variations in the data while correcting a tool. This low impact can become negligible in the presence of more dominant features, e.g., bag of words.

5.2 *Issues* with Politeness in Sentiment Labeling

17 of the 153 manually analyzed misclassified units in our complementary dataset (Section 3.3) were related to the confusions among the sentiment tools to handle politeness. While around 76% of those were corrected by Sentisead, the rest, 24%, remained misclassified across all the different settings of Sentisead and Sentisead+ (see Table 9). There are two main reasons for this. First, the presence of politeness does not necessarily make a unit positive. Second, the manual annotation of units carrying politeness were judged differently by different human raters in the benchmarks. For example, the following is labeled as neutral in the Jira Ortu et al. [46] dataset, *Jacopo Thanks for diving in a pointing Scott at the appropriate screens.*. However, the following is labeled as positive in the same dataset *"Patch applied! Thanks"*.. This discrepancy can confuse a supervised classifier, including Sentisead.

5.3 The Problems with Contextual Inference Using *Surrounding* Sentences in Sentiment Detection

Among the six benchmark datasets in our study, Sentisead$_B$ achieves 0.83-0.98 F1-score to label both positive and negative classes for four datasets: Jira and App reviews from Lin et al. [30], SO Calefato et al. [5], and Jira Ortu et al. [46]. Sentisead shows only 0.31-0.47 F1-scores for the positive and negative classes in other two datasets: SO Lin et al. [30] and SO Uddin et al. [63], with the worst in SO Uddin et al. [63]. Both of these use data from Stack Overflow. The SO Uddin et al. [63] dataset is different from other benchmarks, because it is based on all the sentences from 1,338 posts from Stack Overflow. Thus, when the raters were presented those sentences, they also saw other sentences immediately after and before the sentences. This *surrounding* contextual information influenced their annotation. This particular issue was also discussed in Uddin et al. [63]. The SO Lin et al. [30] benchmark seems to have the same problem. For example, the following sentence is rated as 'negative' in the dataset *I am not sure about Map implementation in HazelCast.* No sentiment detection tool is designed to check such domain specificity and *surrounding* sentences as contexts during the polarity classification. Therefore, further improvement in Sentisead for these two datasets will need the development of algorithms to handle such domain specificity and contextual information.

5.4 Run-time Performance

In Section 1 (Introduction), we write about different use cases for Sentisead: Sentisead is useful for situations where we do not want to pick different tools for different datasets. Instead, we simply run each stand-alone tool for a given dataset and let Sentisead make the final decision on sentiment polarity.

An analysis can be done offline or online. For example, when analyzing millions of code reviews from a large software system like OpenStack, we expect that the sentiment detection would be done offline and not in real-time. Nevertheless, we were able to run all stand-alone tools within a reasonable time e.g., SentiCR for any dataset for one iteration (i.e., one of the 10-folds) took around two minutes (training + testing). Note that this was done on a Laptop machine with only two cores and 4GB RAM. During testing, for a given sentence, SentiCR [1] was able to produce a polarity label in less than a second. Please note that like any machine learning model, the training of a tool takes the most amount of time (often around 95%). This is true for any currently available SE-specific sentiment detection tool. With advances in computing infrastructure, parallel processing, and high-performance computing, we can expect to further reduce such training and testing time. Indeed, we had some performance issues with Senti4SD [5] initially, but Senti4SD was recently updated to a new and faster version to support large-scale data. Therefore, while the training of such a tool takes the majority of time in Sentisead, the tool can make a polarity label on a given unit at run-time in a short time (as noted above with SentiCR example). Indeed, Sentisead can determine the polarity label of a given unit in less than a second, given it is built on SentiCR - which as noted above is fast to produce labels. Note that, we previously analyzed a large number of OpenStack code reviews in a reasonably short time using some of the the studied tools in this paper (e.g., Senti4SD, SentiCR) (please see Asri et al. [2]).

5.5 The Role of Features in Sentisead Accuracy Improvement

An important area of research in machine learning is the handling of overfitting of a machine learning model. Indeed, in machine learning literature, it is a standard practice to prune the feature space of a model to improve model inference runtime. Models like Random Forest are designed to handle overfitting efficiently by automatically pruning the decision trees. In RQ_4 (Section 3.4), we find that a Random Forest-based hybrid tool is the best performer in each individual dataset. In RQ_5 (Section 3.5), we find that by simply adding extra features (e.g., partial polarity, polarity entropy), we could not gain notable performance improvement in the new hybrid tool in RQ_5 (i.e., Sentisead_B+) over the the hybrid tool from RQ_4 (i.e, Sentisead_B). In RQ_8, we attempted to develop a hybrid Sentisead engine by taking polarity labels of all nine stand-alone sentiment detectors (five rule-based and shallow learning and four BERT-based PTMs from Zhang et al. [74]). We found that this Sentisead engine performed a bit worse than the Sentisead engine that only took polarity labels from five rule-based and shallow learning sentiment detectors. These three observations offer insights that by simply adding new features, we cannot not improve the performance of our hybrid sentiment detection tool.

5.6 Best Performers Per Dataset

When we analyzed the results of each tool, we analyzed along two dimensions: Per dataset and across all datasets. We provide a detailed explanation of how we trained and tested each tool per dataset in Section 3.1.2. We also explain in Section 3.1.2, how we combine the test results from each dataset to report an overall performance score of a tool across the six datasets.

An overall performance score across the six datasets informs us of the overall effectiveness of a tool across the six datasets. This score is important given that these datasets are designed for different purposes. As noted in Section 1, the major motivation behind designing our hybrid tool Sentisead was to obtain better results than the stand-alone sentiment detectors on diverse datasets. This overall performance score tells us whether the tool does indeed outperform each stand-alone tool across the datasets using a single metric.

Table 20. The Three Best Performing Tools Per Dataset in Our Study (F1 = Macro F1 Score)

Dataset	First		Second		Third	
	Tool	**F1**	**Tool**	**F1**	**Tool**	**F1**
JIRA Lin et al. [30]	SentistrengthSE	0.98	Sentisead$_{BERT}$	0.97	RoBERTa	0.97
SO Uddin et al. [63]	Sentisead$_{XLNet}$	0.59	Sentisead$_{XLNet+}$	0.59	Sentisead$_{RoBERTa}$	0.59
Mobile App Lin et al. [30]	BERT	0.72	Sentisead$_{RoBERTa}$	0.68	RoBERTa	0.62
SO Lin et al. [30]	Sentisead$_{RoBERTa+}$	0.78	RoBERTa	0.77	Sentisead$_{XLNert+}$	0.74
SO Calefato et al. [5]	Sentisead$_{RoBERTa}$	0.89	Sentisead$_{XLNet}$	0.88	RoBERTa	0.88
JIRA Ortu et al. [46]	XLNet	0.83	Sentisead$_{RoBERTa}$	0.82	Sentisead$_{RoBERTa+}$	0.82

Given that we studied six benchmark datasets, nine stand-alone sentiment detection tools, various settings of our hybrid tool Sentisead by answering eight research questions in the revised manuscript, it is a non-trivial to report the performance of each studied tool and the hybrid tool for each research question per studied dataset: doing so would be lengthy and difficult to read. Nevertheless, we share the results of each tool per dataset and per research question in our online replication package [27]. In Table 20, we report the top three tools in each dataset of our study based in Macro F1 score.

Around 61% (i.e., 11 out of the 18) places of the best performing tools in Table 20 are occupied by the different ensemble combinations of Sentisead (e.g., Sentisead$_{RoBERTa}$), while 33% of the tools (i.e., 6 out of 18) are the stand-alone PTMs, and the rest (i.e., 1 out of 18) of the tools is SentistrengthSE. Among the 61% places occupied by Sentisead variations, all datasets contain different combinations of the PTMs as the ensembler. For the one dataset (JIRA Lin et al. [30]), where SentistrengthSE is the best performer with a Macro F1-score of 0.98, the second best are two tools with an F1-score of 0.97 (Sentisead$_{B}ERT$ and RoBERTa). For the other five datasets, the topmost performer is Sentisead-based in three SO-related datasets: SO Uddin et al. [63], SO Lin et al. [30], and SO Calefato et al. [5]. For the other two datasets, stand-alone PTMs are the best performers: BERT for Mobile App Lin et al. [30] and XLNet for JIRA Ortu et al. [46]. The findings denote that PTMs as a stand-alone tool or as an ensembler of existing tools can be the best candidates to detect sentiments in diverse SE-specific datasets.

6 THREATS TO VALIDITY

We discuss the threats to validity of our studies following common guidelines for empirical studies [71].

Construct Validity Threats concern the relation between theory and observations. In our study, they could be due to measurement errors. For our manual labeling of error categories, we followed the coding guide and examples of Novielli et al. [42]. This is not the first time we did the manual labeling of such error categories. In our previously published paper [63], we did a similar exercise on a different dataset. Nevertheless, similar to Novieli et al. [42], we found three error categories: Contextual information, Politeness, and General Errors. In addition, we found two new error categories: Domain-specificity and presence of diverse polarity. We provide examples of each category in Section 3 (RQ_3). In addition, we share the manually labeled dataset in our online appendix [27].

Internal validity warrants the presence of systematic error (e.g., bias) in the study. This can happen due to the reporting of a performance metric that may not be standard or due to the error in the computation of the standard metric. We mitigated the first bias by reporting standard metrics used in the literature. For example, we compute both Macro and Micro scores, following standard reporting practices in sentiment analysis for software engineering. We mitigated the second concern by confirming the computation of the metrics against the same metrics that are presently computed in widely used open source libraries (e.g., scikit-learn [52]). In addition, we also report the details of how each metric parameters are computed (e.g., true positives, false positives, etc.).

External validity concerns the *generalizability* of the results presented in this paper, which are based on six publicly-available sentiment benchmarks. Together, these benchmarks consist of more than 17K units (sentences, posts, etc.), each manually annotated for polarity by multiple coders. Therefore, our evaluation corpus is bigger than any previous research on sentiment detection for software engineering.

Reliability concerns the possibility of replicating this evaluation. We provide the necessary details in an online appendix [27].

7 RELATED WORK

Sentiment analysis in SE research has focused on two main areas: studying sentiments SE repositories/scenarios (Section 7.1) and developing tools to improve sentiment detection in SE (Section 7.2).

7.1 Studying Sentiments SE Repositories/Scenarios

A significant number of studies are conducted to analyze the prevalence and impact of sentiments in SE repositories and development scenarios, such as IT tickets [7], collaborative distributed team communications [17], Twitter feeds [16], commit logs [55], security-related discussions in GitHub [51], API reviews from developer forums by resolving API mentions and by associating opinions to the API mentions [59–66], and code reviews [2]. Studies also combined sentiment analysis with other techniques to improve domain-specific classification, such as to classify app reviews [33, 49].

In parallel, a number of research efforts analyze emotions in SE repositories. While sentiment analysis focuses on polarity (positive, negative, neutral), emotion analysis focuses finer grained expressions such as anger, love, etc. Emotion analysis is used to determine team cohesions, such as the impact of bullies in an SE team [46], the relation of VAD (Valence, Arousal, Dominance) scores [70] with productivity and burn-out in SE teams [36], and the usage of expressed emotions to prioritize improvements or raise team awareness [13, 18]. Overall, analysis of sentiments and emotions is an increasingly popular field in SE [39, 44]

Contrary to the above work, in this paper, we focus on the development of a sentiment tool for SE.

7.2 Development of Sentiment Detection Tools in SE

SE-specific sentiment tools are needed, because off-the-shelf sentiment detection tools from other domains are not very accurate [23, 24]. Subsequently, a number of sentiment and emotion detectors and benchmarks were developed for SE [1, 5, 6, 9, 21, 22, 22, 29, 34, 38, 47, 63]. Largely, sentiment detection efforts for SE can be divided into three categories: rule-based, shallow learning-based, and deep learning-based.

We started to build our hybrid tool, Sentisead, by combining the polarity labels of five SE-specific rule-based and shallow learning sentiment detection tools (Opiner [63], POME [29], SentiCR [1], Senti4SD [5], and SentistrengthSE [21]). SentistrengthSE [21] was studied by Lin et al. [30] along with other tools. Recently, Novielli et al. [43] showed that the two supervised detectors (Senti4SD [5] and SentiCR [1]) performed better once re-trained for a domain. Thus, we retrained those for each of our six benchmarks. In a subsequent study, Novielli et al. [40] found that unsupervised sentiment tools for SE, such as SentistrengthSE work better than supervised tools when applied in a cross-platform setting. Maipradit et al. [34] reported superior performance using n-grams in shallow learning models, like SVM, in multiple datasets. However, it is not clear which value of n in the n-grams provided the best result.

Recently, several deep learning models to detect sentiment in SE artifacts were described in the literature [3, 4, 9, 74]. In 2019, Chen et al. [9] proposed SentiEmoji, an emoji-powered sentiment detection tools for SE data. SentiEmoji uses posts containing emotional emojis from Tweets and GitHub posts as new data to automatically produce a new sentiment benchmark dataset. SentiEmoji combines the dataset with the existing SE sentiment benchmark dataset and proposes a representation learning approach to train/test sentiments in the dataset. The approach offered superior performance than shallow learning tools like Senti4SD.

In 2020, several papers reported language-based pre-trained transformer models (PTMs) [3, 74]. Biswas et al. [3] showed that the use of PTMs, like BERT, can offer superior performance to other deep learning models, like RNN, which they studied before [3]. At the same time, Zhang et al. [74] investigated four PTMs: BERT, ALBERT, RoBERTa, and XLNet. They found that the PTMs outperformed state-of-the-art shallow learning and rule based SE-specific sentiment detectors across multiple datasets. The findings of Zhang et al. [74] and Biswas et al. [3] motivated our inclusion of the PTMs into our hybrid tool.

Unlike our paper, none of the above studies investigate the feasibility of combining the tools. Our study, for the first time in SE, shows that a hybrid of the stand-alone SE-specific sentiment detection tools can be developed and that such a tool can indeed outperform each of those stand-alone SE-specific sentiment tools.

8 CONCLUSIONS

Sentiment detection is important to understand and support diverse scenarios in software engineering and software-engineering research, yet studies from Lin et al. [30] and Jongeling et al. [24] showed that currently available sentiment detection tools for software engineering may not accurately detect polarity in software textual contents.

To determine whether and how the existing SE-specific sentiment tools could be combined to increase their performance, we reported the results of an empirical study that explores the feasibility of creating a hybrid sentiment detection tool. We started with five stand-alone rule-based and shallow learning sentiment detectors for SE. We reported that the stand-alone tools can complement each other to offer better performance, but a majority voting-based unsupervised classifier fails to increase performance. We found that a simple supervised combination of polarity labels from individual tools and bag of words as features could increase the performance (F1-scores) of stand-alone tools by 4% to more than 100%. We named the developed supervised hybrid tool Sentisead. Therefore, Sentisead can offer better performance than stand-alone detection tools, but it was not more than 4% against the best performing stand-alone tool (Senti4SD).

Our study was motivated by the negative results reported by Lin et al. [30] in 2018 and then the positive results reported by Lin et al. [29] in 2019 based on a newly developed tool. We wanted to understand whether we could further improve the performance of the stand-alone sentiment detectors (which were publicly available at the time of our analysis). Instead of creating a new stand-alone sentiment detection tool for software engineering, we wanted to determine whether we could leverage the strengths of each individual SE-specific tool. Therefore, we studied the feasibility of developing an ensemble of those stand-alone tools. When compared with the results reported in Lin et al. [30] and Sentisead, we found that Sentisead offers better performance in the three datasets summarized in Table 21. Note that the performance of SentistrengthSE on Jira Lin et al. [30] dataset was reported as 0.91 by Lin et al. [30], but we observed it to be 0.97 as shown in in Table 20 (Section 5.6).

Table 21. Sentisead$_{RF}$ vs. SentistrengthSE Performance Reported by Lin et al. [30]

	Positive Polarity		Negative Polarity	
Dataset	**Lin [30]**	**Sentisead**	**Lin [30]**	**Sentisead**
SO	0.26	0.35	0.27	0.31
App	0.78	0.89	0.45	0.83
Jira	0.91	0.95	0.82	0.98

The recent promising results in SE-specific sentiment detection with language-based pre-trained transformer models (PTMs, e.g., BERT), as reported in several works, e.g., Zhang et al. [74], has motivated the second phase of our study, where we compared Sentisead with four stand-alone PTMs (BERT, RoBERTa, ALBERT, and XLNet). We found that a stand-alone RoBERTa outperforms Sentisead by 1.8% (in terms of Macro F1-score). When we replaced the ensembler in Sentisead from a Random Forest to RoBERTa, we observed a performance increase of 2.3% over the Random Forest-based Sentisead (i.e., Sentisead$_{RF}$). However, when we checked the performance per dataset, the findings were inconclusive against the stand-alone PTMs, i.e., Sentisead-based various ensemble marginally outperformed the stand-alone PTMs in several cases but were outperformed by those PTMs in other cases. In particular, we found in Section 4 that PTMs like RoBERTa performed considerably well as stand-alone sentiment detectors. Therefore, when such marginal improvement via Sentisead is not warranted, a stand-alone PTM might suffice for sentiment detection in SE datasets.

As noted in Section 3.3, out of the five misclassification categories, we observed that three categories were also previously observed and reported by Novielli et al. [43]. While Novielli et al. [43] observed the error categories in units where all tools were wrong, we observed the error categories for units where at least one of the tools is correct. The purpose of our study to derive the misclassification categories was twofold: first, we needed to know the specific reasons for misclassification in one tool where another tool could offer the correct result. Second, we wanted to use knowledge gained from the error categories to develop our hybrid tool, Sentisead.

Around 50% of all misclassifications were due to the inability of the wrong tool to understand the underlying contexts, while the correct tool understood the context. Based on this insight, we hypothesized that if we could combine the polarity labels of all tools (where the correct tool is also included) with the bag of words of the unit that would serve as contexts, we could offer better results in our hybrid tool, Sentisead. Indeed, Sentisead corrects the contexts-based error for 80% of the cases (i.e., it was wrong in only 20% of the 62 units where at least one stand-alone tool is wrong), which is more than any of the stand-alone tools. Table 9 offers a qualitative breakdown of how Sentisead offered improvement over each stand-alone tool based on the five error categories.

Sentisead infrastructure could be leveraged to add the polarity labels of any newly introduced SE-specific stand-alone sentiment detection tool. We envision that this flexibility of Sentisead will allow the co-creation of two new-breeds of efforts in software engineering research: (1) More efforts to create new SE-specific stand-alone sentiment detection tool focusing on diverse SE datasets, and (2) Studying (a) the incorporation of diverse complementary SE-specific stand-alone sentiment detection tools and (b) the effectiveness of diverse features, such as fine-grained emotions from various tools into Sentistead (or its extension).

As we noted in Section 1, recent advances in deep learning-based SE-specific sentiment detection shows more promising results than those observed by Lin et al. [30]. While within the limited space of this paper, we attempted to offer a comprehensive analysis by investigating the feasibility of developing a hybrid tool based on rule-based, shallow and deep learning stand-alone classifiers, we leave the following extensions to Sentisead for future work:

(1) SentiEmoji [9], a tool developed by Chen et al. shows that we can use emoji-powered data from Twitter and Facebook to further improve sentiment classification in SE. SentiEmoji [9] also shows that an embedding based on the emoji information and the textual contents can be more useful. Sentisead could leverage such emoji-powered information as features besides word-embedding or bag of words.

(2) Deep learning models, like PTMs, are found to offer good performance even with a low amount of data due to their design as pre-trained models. However, deep learning models also benefit from more data. A problem with SE-specific sentiment detection is the diversity of datasets. With emoji-powered labels like those used by SentiEmoji [9], we may be able to get more data with automatic polarity labels, which then can help more with domain-specific nuances. Therefore, Sentisead could benefit from such large-scale data, especially for PTMs like a RoBERTa as an ensembler (see RQ_6).

We conclude that our empirical study with Sentisead offers valuable insights to the field of SE-specific sentiment detection research with empirical insights, actionable results, and a hybrid tool Sentisead that shows marginal to promising performance over stand-alone SE-specific sentiment detection tools.

APPENDIX

A DETAILS OF PERFORMANCE ANALYSIS OF TOOLS

Table 22. Performance Comparison of Majority Voting-based Hybrid Classifiers with the Baselines

	Positive			**Negative**			**Neutral**		
	F1	**P**	**R**	**F1**	**P**	**R**	**F1**	**P**	**R**
Majority-All	0.721	0.825	0.640	0.645	0.859	0.516	0.819	0.734	0.925
Majority-Supervised	0.706	0.818	0.621	0.645	0.879	0.510	0.825	0.735	0.939
Majority-Unsupervised	0.536	0.736	0.422	0.490	0.762	0.361	0.755	0.647	0.907
Baseline - **Senti4SD**	0.731	0.773	0.692	0.690	0.748	0.640	0.822	0.783	0.865
Baseline - **SentiCR**	0.728	0.803	0.667	0.687	0.804	0.599	0.830	0.767	0.903
Baseline - **SentistrengthSE**	0.720	0.752	0.690	0.640	0.736	0.566	0.795	0.750	0.847
Baseline - **Opiner**	0.494	0.467	0.524	0.512	0.577	0.460	0.676	0.668	0.684
Baseline - **POME**	0.127	0.389	0.076	0.065	0.415	0.035	0.697	0.550	0.950

Extension of Table 5.

Table 23. Performance Comparison of the Developed Hybrid Classifier (Sentisead) with the Baselines

	Positive			Negative			Neutral		
	F1	**P**	**R**	**F1**	**P**	**R**	**F1**	**P**	**R**
Sentisead$_{RF-B}$	0.764	0.832	0.706	0.738	0.827	0.666	0.843	0.790	0.905
Sentisead$_{RF-N}$	0.759	0.797	0.724	0.72	0.777	0.671	0.837	0.801	0.877
Performance Increase of Sentisead$_B$ **(i.e., Hybrid with Bag of Words) Over Baseline (SentiSE = SentistrengthSE)**									
Majority	6.0%	0.8%	10.3%	14.4%	−3.7%	29.1%	2.9%	7.6%	−2.2%
Senti4SD	4.5%	7.6%	2.0%	7.0%	10.6%	4.1%	2.6%	0.9%	4.6%
SentiCR	4.9%	3.6%	5.8%	7.4%	2.9%	11.2%	1.6%	3.0%	0.2%
SentiSE	6.1%	10.6%	2.3%	15.3%	12.4%	17.7%	6.0%	5.3%	6.8%
Opiner	54.7%	78.2%	34.7%	44.1%	43.3%	44.8%	24.7%	18.3%	32.3%
POME	501.6%	113.9%	828.9%	1035.4%	99.3%	1802.9%	20.9%	43.6%	−4.7%

Extension of Table 8.

Table 24. Performance Comparison of the Designed Hybrid Detector (Sentisead) with Added Features from Table 10

	Positive			Negative			Neutral		
	F1	**P**	**R**	**F1**	**P**	**R**	**F1**	**P**	**R**
Sentisead$_{RF-B}$**+**	0.767	0.832	0.712	0.74	0.818	0.675	0.844	0.794	0.901
Sentisead$_{RF-BNE}$**+**	0.761	0.831	0.702	0.738	0.832	0.663	0.843	0.788	0.906
Sentisead$_{RF-BNP}$**+**	0.765	0.829	0.711	0.737	0.813	0.674	0.843	0.793	0.899
SentiSead$_{RF-N}$**+**	0.747	0.777	0.719	0.698	0.741	0.66	0.822	0.793	0.854
Sentisead$_{RF-NNE}$**+**	0.755	0.791	0.722	0.708	0.772	0.653	0.834	0.796	0.876
Sentisead$_{RF-NNP}$**+**	0.746	0.773	0.721	0.698	0.741	0.66	0.821	0.793	0.852
Sentisead$_{RF-B}$*: With bag of words. Sentisead$_{RF-N}$*: No bag of words. NP = No partial polarity. NE = No polarity entropy.									

Extension of Table 11.

Table 25. Performance of Stand-alone Advanced Pre-trained Advanced Language-based Deep Learning Models Across the Datasets

	Positive			Negative			Neutral		
	F1	**P**	**R**	**F1**	**P**	**R**	**F1**	**P**	**R**
BERT	0.772	0.768	0.775	0.759	0.767	0.752	0.847	0.845	0.848
Change Over Sentisead$_{RF}$	1.0%	−7.7%	9.8%	2.8%	−7.3%	12.9%	0.5%	7.0%	−6.3%
RoBERTa	0.788	0.773	0.804	0.768	0.768	0.768	0.847	0.855	0.839
Change Over Sentisead$_{RF}$	3.1%	−7.1%	13.9%	4.1%	−7.1%	15.3%	0.5%	8.2%	−7.3%
ALBERT	0.767	0.792	0.743	0.747	0.764	0.731	0.846	0.828	0.865
Change Over Sentisead$_{RF}$	0.4%	−4.8%	5.2%	1.2%	−7.6%	9.8%	0.4%	4.8%	−4.4%
XLNet	0.779	0.749	0.812	0.767	0.786	0.749	0.848	0.857	0.84
Change Over Sentisead$_{RF}$	2.0%	−10.0%	15.0%	3.9%	−5.0%	12.5%	0.6%	8.5%	−7.2%

Extension of Table 13.

Table 26. Performance of Sentisead Using Pre-trained Language Based Advanced Deep Learning Models as Ensembler

	Positive			Negative			Neutral		
	F1	**P**	**R**	**F1**	**P**	**R**	**F1**	**P**	**R**
Sentisead$_{BERT}$	0.77	0.771	0.768	0.76	0.761	0.759	0.843	0.842	0.844
Change Over Sentisead$_{RF}$	0.8%	−7.3%	8.8%	3.0%	−8.0%	14.0%	0.0%	6.6%	−6.7%
Change Over BERT	−0.3%	0.4%	−0.9%	0.1%	−0.8%	0.9%	−0.5%	−0.4%	−0.5%
Sentisead$_{RoBERTa}$	0.787	0.78	0.794	0.774	0.792	0.757	0.853	0.849	0.857
Change Over Sentisead$_{RF}$	3.0%	−6.2%	12.5%	4.9%	−4.2%	13.7%	1.2%	7.5%	−5.3%
Change Over RoBERTa	−0.1%	0.9%	−1.2%	0.8%	3.1%	−1.4%	0.7%	−0.7%	2.1%
Sentisead$_{ALBERT}$	0.764	0.769	0.76	0.735	0.731	0.738	0.839	0.838	0.84
Change Over Sentisead$_{RF}$	0.0%	−7.6%	7.6%	−0.4%	−11.6%	10.8%	−0.5%	6.1%	−7.2%
Change Over ALBERT	−0.4%	−2.9%	2.3%	−1.6%	−4.3%	1.0%	−0.8%	1.2%	−2.9%
Sentisead$_{XLNet}$	0.785	0.774	0.796	0.767	0.772	0.763	0.848	0.852	0.844
Change Over Sentisead$_{RF}$	2.7%	−7.0%	12.7%	3.9%	−6.7%	14.6%	0.6%	7.8%	−6.7%
Change Over XLNet	0.8%	3.3%	−2.0%	0.0%	−1.8%	1.9%	0.0%	−0.6%	0.5%

Extension of Table 15.

Table 27. Performance of Sentisead Using Pre-trained Advanced Deep Learning Models as Ensembler and Stand-alone Detectors

	Positive			Negative			Neutral		
	F1	**P**	**R**	**F1**	**P**	**R**	**F1**	**P**	**R**
Sentisead$_{BERT+}$	0.769	0.766	0.771	0.757	0.76	0.755	0.844	0.844	0.844
Change Over Sentisead$_{RF}$	0.7%	−7.9%	9.2%	2.6%	−8.1%	13.4%	0.1%	6.8%	−6.7%
Change Over Sentisead$_{BERT}$	−0.1%	−0.6%	0.4%	−0.4%	−0.1%	−0.5%	0.1%	0.2%	0.0%
Change Over BERT	−0.4%	−0.3%	−0.5%	−0.3%	−0.9%	0.4%	−0.4%	−0.1%	−0.5%
Sentisead$_{RoBERTa+}$	0.783	0.758	0.809	0.768	0.776	0.761	0.847	0.857	0.837
Change Over Sentisead$_{RF}$	2.5%	−8.9%	14.6%	4.1%	−6.2%	14.3%	0.5%	8.5%	−7.5%
Change Over Sentisead$_{RoBERTa}$	−0.5%	−2.8%	1.9%	−0.8%	−2.0%	0.5%	−0.7%	0.9%	−2.3%
Change Over RoBERTa	−0.6%	−1.9%	0.6%	0.0%	1.0%	−0.9%	0.0%	0.2%	−0.2%
Sentisead$_{ALBERT+}$	0.766	0.763	0.768	0.744	0.752	0.736	0.84	0.838	0.843
Change Over Sentisead$_{RF}$	0.3%	−8.3%	8.8%	0.8%	−9.1%	10.5%	−0.4%	6.1%	−6.9%
Change Over Sentisead$_{ALBERT}$	0.3%	−0.8%	1.1%	1.2%	2.9%	−0.3%	0.1%	0.0%	0.4%
Change Over Albert	−0.1%	−3.7%	3.4%	−0.4%	−1.6%	0.7%	−0.7%	1.2%	−2.5%
Sentisead$_{XLNet+}$	0.777	0.752	0.803	0.768	0.781	0.756	0.845	0.852	0.837
Change Over Sentisead$_{RF}$	1.7%	−9.6%	13.7%	4.1%	−5.6%	13.5%	0.2%	7.8%	−7.5%
Change Over Sentisead$_{XLNet}$	−1.0%	−2.8%	0.9%	0.1%	1.2%	−0.9%	−0.4%	0.0%	−0.8%
Change Over XLNet	−0.3%	0.4%	−1.1%	0.1%	−0.6%	0.9%	−0.4%	−0.6%	−0.4%

Extension of Table 17.

ACKNOWLEDGMENT

We would like to thank the anonymous reviewers, who helped improve our work with comments and suggestions. We would like to thank Nibir Mandal, Mohamed Raed El Aoun, and Houssem Ben Braiek, who helped run the experiments with the advanced language-based pre-trained models in Section 4.

REFERENCES

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

[2] Ikram El Asri, Noureddine Kerzazi, Gias Uddin, Foutse Khomh, and M. A. Janati Idrissi. 2019. An empirical study of sentiments in code reviews. *Information and Software Technology* 114 (2019), 37–54.

[3] Eeshita Biswas, Mehmet Efruz Karabulut, Lori Pollock, and K. Vijay-Shanker. 2020. Achieving reliable sentiment analysis in the software engineering domain using BERT. In *IEEE International Conference on Software Maintenance and Evolution*. 162–173.

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

[5] Fabio Calefato, Filippo Lanubile, Federico Maiorano, and Nicole Novielli. 2017. Sentiment polarity detection for software development. *Journal Empirical Software Engineering* (2017), 2543–2584.

[6] Fabio Calefato, Filippo Lanubile, and Nicole Novielli. 2017. EmoTxt: A toolkit for emotion recognition from text. In *Proc. 7th Affective Computing and Intelligent Interaction*. 2.

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

[8] Nitesh V. Chawla, Kevin W. Bowyer, Lawrence O. Hall, and W. Philip Kegelmeyer. 2002. SMOTE: Synthetic minority over-sampling technique. *Journal of Artificial Intelligence Research* 16, 1 (2002), 321–357.

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

[10] Jacob Cohen, Stephen G. West, Leona Aiken, and Patricia Cohen. 2002. *Applied Multiple Regression/Correlation Analysis for the Behavioral Sciences* (3rd ed.). Lawrence Erlbaum Associates.

[11] Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. 2018. *BERT: Pre-training of Deep Bidirectional Transformers for Language Understanding*. Technical Report. https://arxiv.org/abs/1810.04805.

[12] Pedro P. Balage Filho, Lucas Avanco, Thiago A. S. Pardo, and Maria G. V. Nunes. 2014. NILC_USP: An improved hybrid system for sentiment analysis in Twitter messages. In *Proceedings of the 8th International Workshop on Semantic Evaluation*. 428–432.

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

[14] Baljinder Ghotra, Shane McIntosh, and Ahmed E. Hassan. 2015. Revisiting the impact of classification techniques on the performance of defect prediction models. In *Proceedings of the 37th International Conference on Software Engineering*. 789–800.

[15] Pollyanna Goncalves, Matheus Araujo, Fabricio Benevenuto, and Meeyoung Cha. 2013. Comparing and combining sentiment analysis methods. In *Proceedings of the First ACM Conference on Online Social Networks*. 27–38.

[16] Emitza Guzman, Rana Alkadhi, and Norbert Seyf. 2016. A needle in a haystack: What do Twitter users say about software? In *24th IEEE International Requirements Engineering Conference*. 96–105.

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

[18] Emitza Guzman and Bernd Bruegge. 2013. Towards emotional awareness in software development teams. In *Proceedings of the 2013 9th Joint Meeting on Foundations of Software Engineering*. 671–674.

[19] Ahmed E. Hassan. 2009. Predicting faults using the complexity of code changes. In *Proc. 31st International Conference on Software Engineering*. 78–89.

[20] Minqing Hu and Bing Liu. 2004. Mining and summarizing customer reviews. In *ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*. 168–177.

[21] Md Rakibul Islam and Minhaz F. Zibran. 2017. Leveraging automated sentiment analysis in software engineering. In *Proc. 14th International Conference on Mining Software Repositories*. 203–214.

[22] Md Rakibul Islam and Minhaz F. Zibran. 2018. DEVA: Sensing emotions in the valence arousal space in software engineering text. In *33rd Annual ACM Symposium on Applied Computing*. 1536–1543.

[23] Robbert Jongeling, Subhajit Datta, and Alexandar Serebrenik. 2015. Choosing your weapons: On sentiment analysis tools for software engineering research. In *Proceedings of the 31st International Conference on Software Maintenance and Evolution*.

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

[25] Foutse Khomh, Brian Chan, Ying Zou, and Ahmed E. Hassan. 2011. An entropy evaluation approach for triaging field crashes: A case study of Mozilla Firefox. In *Proceedings of the 2011 18th Working Conference on Reverse Engineering*. 261–270.

[26] Andrew J. Ko, Brad A. Myers, and Duen Horng Chau. 2005. A linguistic analysis of how people describe software problems. In *2005 IEEE Symposium on Visual Languages and Human-Centric Computing*. 127–134.

[27] DISA Lab. 21 April 2021 (last accessed). *An Empirical Study of the Effectiveness of an Ensemble of Stand-alone Sentiment Detection Tools for Software Engineering Datasets (Online Appendix)*. https://github.com/disa-lab/HybridSESentimentTOSEM.

[28] Zhenzhong Lan, Mingda Chen, Sebastian Goodman, Kevin Gimpel, Piyush Sharma, and Radu Soricut. 2020. *ALBERT: A Lite BERT for Self-supervised Learning of Language Representations.* Technical Report. https://arxiv.org/abs/1909.11942.
[29] Bin Lin, Fiorella Zampetti, Gabriele Bavota, Massimiliano Di Penta, and Michele Lanza. 2019. Pattern-based mining of opinions in Q&A websites. In *Proc. 41st International Conference on Software Engineering.* 548–559.
[30] 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 *Proc. 40th International Conference on Software Engineering.* 94–104.
[31] Bing Liu. 2012. *Sentiment Analysis and Opinion Mining* (1st ed.). Morgan & Claypool Publishers.
[32] Yinhan Liu, Myle Ott, Naman Goyal, Jingfei Du, Mandar Joshi, Danqi Chen, Omer Levy, Mike Lewis, Luke Zettlemoyer, and Veselin Stoyanov. 2019. *RoBERTa: A Robustly Optimized BERT Pretraining Approach.* Technical Report. https://arxiv.org/abs/1907.11692.
[33] Walid Maalej, Zijad Kurtanović, Hadeer Nabil, and Christoph Stanik. 2016. On the automatic classification of app reviews. In *International Requirements Engineering Conference.* 311–331.
[34] Rungroj Maipradit, Hideaki Hata, and Kenichi Matsumoto. 2019. Sentiment classification using N-Gram inverse document frequency and automated machine learning. *IEEE Software* 36, 5 (2019), 65–70.
[35] Christopher D. Manning, Prabhakar Raghavan, and Hinrich Schütze. 2009. *An Introduction to Information Retrieval.* Cambridge Uni Press.
[36] Mika Màntylà, Bram Adams, Giuseppe Destefanis, Daniel Graziotin, and Marco Ortu. 2016. Mining valence, arousal, and dominance – possibilities for detecting burnout and productivity?. In *Proceedings of the 13th Working Conference on Mining Software Repositories.* 247–258.
[37] Ayse Tosun Misirli, Ayse Basar Bener, and Burak Turhan. 2011. An industrial case study of classifier ensembles for locating software defects. *Software Quality Journal* 19, 3 (2011), 515–536.
[38] Alessandro Murgia, Parastou Tourani, Bram Adams, and Marco Ortu. 2014. Do developers feel emotions? An exploratory analysis of emotions in software artifacts. In *Proceedings of the 11th Working Conference on Mining Software Repositories.*
[39] Nicole Novielli, Andrew Begel, and Walid Maalej. 2019. Introduction to the special issue on affect awareness in software engineering. *Journal of Systems and Software* 148 (2019), 180–182.
[40] Nicole Novielli, Fabio Calefato, Davide Dongiovanni, Daniela Girardi, and Filippo Lanubile. 2020. Can we use SE-specific sentiment analysis tools in a cross-platform setting? In *17th International Conference on Mining Software Repositories.* 11.
[41] Nicole Novielli, Fabio Calefato, and Filippo Lanubile. 2015. The challenges of sentiment detection in the social programmer ecosystem. In *Proceedings of the 7th International Workshop on Social Software Engineering.* 33–40.
[42] Nicole Novielli, Fabio Calefato, and Filippo Lanubile. 2018. A gold standard for emotion annotation in stack overflow. In *Proceedings of the 15th International Conference on Mining Software Repositories (Data Showcase).* 4.
[43] Nicole Novielli, Daniela Girardi, and Filippo Lanubile. 2018. A benchmark study on sentiment analysis for software engineering research. In *Proceedings of the 15th International Conference on Mining Software Repositories.* 12.
[44] Nicole Novielli and Alexander Serebrenik. 2019. Sentiment and emotion in software engineering. *IEEE Software* 36, 5 (2019), 6–23.
[45] Dario Di Nucci, Fabio Palomba, Rocco Oliveto, and Andrea De Lucia. 2017. Dynamic selection of classifiers in bug prediction: An adaptive method. *IEEE Transactions on Emerging Topics in Computational Intelligence* 1, 3 (2017), 202–212.
[46] Marco Ortu, Bram Adams, Giuseppe Destefanis, Parastou Tourani, Michele Marchesi, and Roberto Tonelli. 2015. Are bullies more productive? Empirical study of affectiveness vs. issue fixing time. In *Proceedings of the 12th Working Conference on Mining Software Repositories.* 303–313.
[47] Marco Ortu, Alessandro Murgia, Giuseppe Destefanis, Parastou Tourani, Roberto Tonelli, Michele L. Marchesi, and Bram Adams. 2016. The emotional side of software developers in JIRA. In *13th International Conference on Mining Software Repositories.* 480–483.
[48] Bo Pang, Lillian Lee, and Shivakumar Vaithyanathan. 2002. Thumbs up? Sentiment classification using machine learning techniques. In *Conference on Empirical Methods in Natural Language Processing.* 79–86.
[49] 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 *IEEE International Conf. on Software Maintenance and Evolution.* 281–290.
[50] Jean Petric, David Bowes, Tracy Hall, Bruce Christianson, and Nathan Baddoo. 2016. Building an ensemble for software defect prediction based on diversity selection. In *Proceedings of the 10th ACM/IEEE International Symposium on Empirical Software Engineering and Measurement.* Article No. 46.
[51] Daniel Pletea, Bogdan Vasilescu, and Alexander Serebrenik. 2014. Security and emotion: Sentiment analysis of security discussions on GitHub. In *Proceedings of the 11th Working Conference on Mining Software Repositories.* 348–351.

[52] scikit learn. 2017. *Machine Learning in Python*. http://scikit-learn.org/stable/index.html#.

[53] Fabrizio Sebastiani. 2002. Machine learning in automated text categorization. *Journal of ACM Computing Surveys* 34, 1 (2002), 1–47.

[54] C. E. Shannon. 1948. A mathematical theory of communication. *The Bell System Technical Journal* 27, 3 (1948), 379–423.

[55] Vinayak Sinha, Alina Lazar, and Bonita Sharif. 2016. Analyzing developer sentiment in commit logs. In *13th International Conference on Mining Software Repositories*. 281–290.

[56] Richard Socher, Alex Perelygin, Jean Wu, Christopher Manning, Andrew Ng, and Jason Chuang. 2013. Recursive models for semantic compositionality over a sentiment treebank. In *Proc. Conference on Empirical Methods in Natural Language Processing (EMNLP)*. 12.

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

[58] Patanamon Thongtanunam, Shane Mcintosh, Ahmed E. Hassan, and Hajimu Iida. 2017. Review participation in modern code review: An empirical study of the Android, Qt, and OpenStack projects. *Journal of Empirical Software Engineering* 22, 2 (2017), 768–817.

[59] Gias Uddin, Olga Baysal, Latifa Guerroj, and Foutse Khomh. 2019. Understanding how and why developers seek and analyze API related opinions. *IEEE Transactions on Software Engineering* (2019), 40.

[60] Gias Uddin and Foutse Khomh. 2017. Automatic summarization of API reviews. In *Proc. 32nd IEEE/ACM International Conference on Automated Software Engineering*. 159–170.

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

[62] Gias Uddin and Foutse Khomh. 2017. Opiner: A search and summarization engine for API reviews. In *Proc. 32nd IEEE/ACM International Conference on Automated Software Engineering*. 978–983.

[63] Gias Uddin and Foutse Khomh. 2019. Automatic opinion mining from API reviews from stack overflow. *IEEE Transactions on Software Engineering* (2019), 35.

[64] Gias Uddin, Foutse Khomh, and Chanchal K. Roy. 2020. Automatic API usage scenario documentation from technical Q&A sites. *ACM Transactions on Software Engineering and Methodology* (2020), 43.

[65] Gias Uddin, Foutse Khomh, and Chanchal K. Roy. 2020. Automatic mining of API usage scenarios from stack overflow. *Information and Software Technology (IST)* (2020), 16.

[66] Gias Uddin and Martin P. Robillard. 2015. *Resolving API Mentions in Forum Texts*. Technical Report. McGill University.

[67] Marat Valiev, Bogdan Vasilescu, and James Herbsleb. 2018. Ecosystem-level determinants of sustained activity in open-source projects: A case study of the PyPI ecosystem. In *Proceedings of the 2018 26th ACM Joint Meeting on European Software Engineering Conference and Symposium on the Foundations of Software Engineering*. 644–655.

[68] 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 the 38th International Conference on Software Engineering*. 14–24.

[69] Sida Wang and Christopher D. Manning. 2012. Baselines and bigrams: Simple, good sentiment and topic classification. In *Proceedings of the 50th Annual Meeting of the Association for Computational Linguistics*. 90–94.

[70] Amy Beth Warriner, Victor Kuperman, and Marc Brysbaert. 2013. Norms of valence, arousal, and dominance for 13,915 English lemmas. *Behavior Research Methods* 45, 4 (2013), 1191–1207.

[71] Claes Wohlin, Per Runeson, Martin Höst, Magnus C. Ohlsson, Bjöorn Regnell, and Anders Wesslén. 2000. *Experimentation in Software Engineering: An Introduction*. Kluwer Academic Publishers, Norwell, MA, USA.

[72] Xinli Yang, David Lo, Xin Xia, and Jianling Sun. 2017. TLEL: A two-layer ensemble learning approach for just-in-time defect prediction. *Information and Software Technology* 87, 2 (2017), 206–220.

[73] Zhilin Yang, Zihang Dai, Yiming Yang, Jaime Carbonell, Ruslan Salakhutdinov, and Quoc V. Le. 2020. *XLNet: Generalized Autoregressive Pretraining for Language Understanding*. Technical Report. https://arxiv.org/abs/1906.08237.

[74] Ting Zhang, Bowen Xu, Ferdian Thung, Stefanus Agus Haryono, David Lo, and Lingxiao Jiang. 2020. Sentiment analysis for software engineering: How far can pre-trained transformer models go?. In *IEEE International Conference on Software Maintenance and Evolution*. 70–80.

[75] Yun Zhang, David Lo, Xin Xia, and Jianling Sun. 2018. Combined classifier for cross-project defect prediction: An extended empirical study. *Frontiers of Computer Science* 22, 2 (2018), 280–296.

Received September 2020; revised September 2021; accepted October 2021