Adaptive Sentiment Detection for Software Artifacts

Anonymous
Anonymous
Anonymous
anonymous@anonymo.us

Anonymous
Anonymous
Anonymous
anonymous@anonymo.us

Anonymous
Anonymous
Anonymous
anonymous@anonymo.us

Anonymous
Anonymous
Anonymous
anonymous@anonymo.us

***Abstract*—Developers' perception of software artifacts is often influenced by other developers' opinions. Recently, several sentiment-detection tools have been proposed to detect sentiments in software artifacts. While the tools have improved accuracy over off-the-shelf tools, recent research shows that their performance could still be unsatisfactory, because each tool was designed by targeting specific software domains (e.g., developer forums, code reviews). We present a novel algorithm Sentisead, that can be used to complement and improve existing tools to detect sentiments. Sentisead learns from the polarity output of each tool and incorporates additional linguistic features into a supervised classifier to predict the polarity of a sentence. We experimented Sentisead by applying three publicly available sentiment detection tools into the Sentisead algorithm (SentistrengthSE, Senti4SD, SentiCR). In five sentiment benchmarks, we observe that Sentisead outperforms the three tools. For example, in Sentisead the misclassification of one tool was corrected by another tool. The linguistic feature was useful when all the tools were wrong. Thus, the software-engineering research community could use Sentisead to study sentiments more accurately.**

***Index Terms*—Sentiment, Opinion, Ensemble.**

I. Introduction

According to Bing Liu, "the textual information in the world can broadly be categorized into two main types: facts and opinions" [1]. Such conjecture can also be applied to software-engineering artifacts, e.g., the discussions in online developer forums, and so on. "Facts" are objective expression (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 [1], [2], i.e., given a unit of analysis (e.g., 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 software engineering [3]–[5]. The productivity in a team may depend on developers' sentiments in/of their diverse development activities [4], [6], [7], while the analysis of sentiments towards software artifacts may lead to better selections of software artifacts [3], [8]. Therefore, it is necessary to accurately detect sentiments in software-engineering artifacts [9], so such analyses can offer the right insights.

Most prior work on sentiment analysis in software engineering used cross-domain sentiment-detection tools that are not designed to work on software-related textual documents. This "off-the-shelf" usage is insufficiently accurate for software engineering (see Section II), due to the difference in the domain where the tools were designed and tested. Many cross-domain sentiment-detection tools were developed using movie reviews (e.g., by the Stanford Sentiment Detector [10]). Indeed, the context of sentiments expressed in movie reviews is different from that in software engineering (e.g., API reviews [3]).

Over the last few years, several tools have been developed to automatically detect sentiments in software artifacts, such as SentistrengthSE [11], Senti4SD [12], SentiCR [13]. These tools were designed and tested in three different domains of software engineering. SentistrenghtSE was built by analyzing JIRA issues, while Senti4SD and SentiCR used texts from Stack Overflow and Gerrit code reviews, respectively (see Section II). Consequently, these tools offer better accuracy than the off-the-shelf ones. However, a recent study by Lin et al. [14] reported that SentistrengthSE did not perform better than the Stanford sentiment-detection tool [10], when applied on a StackOverflow data-set. A subsequent study by Novieli et al. [15], however, showed that supervised tools like Senti4SD and SentiCR perform better when they are re-trained for a new domain. They cautioned for fine-tuning as well as properly calibrating the tools before applying them on a new domain.

This work is motivated by the findings of Lin et al. [14] and Novieli et al. [15]. We focus on the feasibility of an *adaptive* sentiment-detection algorithm, that can offer improved performance over the existing tools for sentiment detection by automatically learning from their strengths and weaknesses. 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 [16], [17] (see Section II). In software engineering, the choice of classification methods and the combination of diverse classifiers is effective to detect defects [18], [19]. However, no such combination has been used so far to detect sentiments in software artifacts.

To understand whether existing sentiment detection tools can be combined for better accuracy, we proceed in three phases (see Figure 1). We conduct a case study by applying SentistrengthSE, Senti4SD and SentiCR on five different sentiment benchmarks (see Section III-A). We observe that the tools differ from each other in their strengths and weaknesses, which raises the possibility of complementing each tool with one another. For example, we may correct a misclassification of one tool by another tool. Moreover, additional features (e.g., domain-specific lexicons) can be used to improve the final polarity decision, when all of the three tools are wrong.

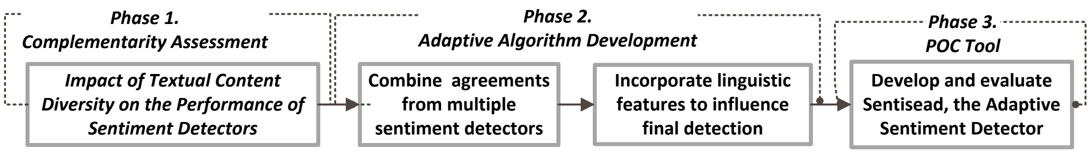
Phase 1.
Complementarity Assessment
Impact of Textual Content Diversity on the Performance of Sentiment Detectors
Phase 2.
Adaptive Algorithm Development
Combine agreements from multiple sentiment detectors
Incorporate linguistic features to influence final detection
Phase 3.
POC Tool
Develop and evaluate Sentisead, the Adaptive Sentiment Detector

Fig. 1. The three phases with the major steps used to develop the adaptive sentiment detector

Motivated by findings of the study, we developed an adaptive sentiment-detection algorithm for software artifacts, Sentisead. Given as input an input text, Sentisead first automatically determines the best possible sentiment detector for the input from a pool of available sentiment detectors. Sentisead then incorporates additional linguistic features (e.g., polarity scores and bag of words) into a supervised classifier to predict the final polarity of the input. The Sentisead algorithm allows the inclusion of any existing sentiment detection tools while picking the best classifier. As an initial feasibility study of the algorithm, we experimented its effectiveness by applying three available sentiment detection tools into the algorithm: SentistrengthSE, SentiCR and Senti4SD. Sentisead outperformed the stand-along detectors by 2.8-11.7% (F1-scores).

Contributions. This paper makes the following contributions:

1) **Case Study** We assess the feasibility of complementarity and improvement opportunities of existing sentiment detection tools by investigating the strengths and weaknesses of three publicly available sentiment detection tools for software engineering (Senti4SD, SentiCR and SentistrengthSE) in five different sentiment benchmarks. We observed that the tools can complement each other in how they determine the polarity, e.g., SentistrengthSE can benefit from the learning of contexts by the supervised classifiers.
2) **Sentisead Adaptive Algorithm.** We design and develop an algorithm Sentisead, that given as input a unit a) automatically determines the best possible classifier from a pool of stand-alone sentiment detectors, and then b) incorporates the polarity output of the best detector with additional linguistic feature to determine the final polarity of the unit. We demonstrate the accuracy of Sentisead over three stand-alone sentiment detectors (SentistrengthSE, SentiCR and Senti4SD) using the five benchmarks.

We advance the state of the art in software engineering with empirical evidence on the complementarity and improvement opportunities of the stand-alone sentiment-detection tools, with the design and development and evaluation of a new algorithm that can automatically combine multiple sentiment-detection tools to determine the best possible polarity of a unit with accuracy higher than the stand-alone tools.

II. Background and Related Work

In this section, we discuss the backgrounds and related research that formed the core concepts of this paper.

Sentiment Analysis in Software Engineering. our research was driven by the potential impact that we can gain through the analysis of sentiments of software artifacts. Ortu et al. [20] observed that bullies are not more productive than others in a software development team. Mäntylä et al. [5] correlated VAD (Valence, Arousal, Dominance) scores [21] in Jira issues with the loss of productivity and burn-out in software engineering teams. Pletea et al. [7] observed that security-related discussions in GitHub contained more negative comments. Guzman et al. [22] found that GitHub projects written in Java as well as the comments posted on Mondays have more negative comments, while the developers in a distributed team are more positive. Guzman and Bruegge [23] summarized emotions expressed across collaboration artifacts in a software team (bug reports, etc.) using LDA [24] and sentiment analysis.

Jongeling et al. [25] compared four sentiment tools on comments posted in Jira (SentiStrength [26], Alchemy [27], Stanford NLP [10], and NLTK [28]). While NLTK and SentiStrength showed better accuracy, there were little agreements between the two tools. These findings indicate that the mere application of an off-the-shelf tool may be insufficient to capture the complex sentiment found in the software artifacts.

Sentiment Tools Developed for Software Artifacts. The observed shortcomings in cross-domain sentiment detection tools motivated the development of three recent sentiment detection tools for software engineering: SentistrengthSE [11], Senti4SD [12], and SentiCR [13].

While SentistrengthSE is rule-based, the other two tools (Senti4SD and SentiCR) are supervised. Senti4SD [12] is trained on a dataset of 4000 posts (questions, answers, and comments) from Stack Overflow. SentiCR [13] was trained on a dataset of 1600 code reviews from Gerrit. Senti4SD uses linguistic features, such as ngrams, sentiment lexicons. Unlike the other two tools, Senti4SD also use semantic features based on word embeddings, which are developed by processing a large corpus of texts from Stack Overflow. The features are then projected along three dimensions to determine their proximity with the three types of polarity. While Senti4SD uses a 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) [29]

SentiStrengthSE [11] was developed on top of Sentistrength [26] by introducing rules and sentiment words specific to the domain of software engineering [11]. 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 sentence. Similar to Sentistrength, SentistrengthSE outputs both positive and negative scores for

TABLE I
STATISTICS OF THE BENCHMARKS USED IN THE STUDY

Dataset	Domain	#Units	+VE	-VE	±/#Units
SO Calefato et al. [12]	StackOverflow	4,423	1,527	1,202	61.7%
SO Lin et al. [14]	StackOverflow	1,500	131	178	20.6%
App Lin et al. [14]	App Reviews	341	186	130	92.7%
Jira Ortu et al. [6]	Jira issues	5,869	1,128	786	32.6%
SO Uddin et al. [3]	StackOverflow	4,522	1,048	839	41.7%
Overall	**SO,App,Jira**	**16,655**	**4,020**	**3,135**	43.0%

an input text. Following state of art [15], 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.

Improvement of Sentiment Detection in Software Artifacts. The above three tools were the subject of two recent studies in software engineering. The first study by Lin et al. [14] compared the outputs of SentistrengthSE with those of Stanford sentiment detector and found that SentistrengthSE did not perform better than the the off-the-shelf detector. The second study by Novieli et al. [15] showed that the two supervised detectors (SentiCR and Senti4SD) performed well once re-trained for a domain. Novieli et al. [15] also manually categorized the misclassifications where all the tools were wrong. They observed the following error types: 1) Polar facts by neutral, 2) General error, 3) Politeness, 4) Implicit sentiment polarity, 5) Subjectivity in sentiment annotation, 6) Inability of classifiers to deal with context information, and 7) Figurative language. Unlike our paper, none of the studies investigate the feasibility of combining the tools.

The development of such hybrid/ensemble classifiers is an area of extensive research in software defect prediction [18], [19]. In the field of sentiment detection, hybrids of rule-based and supervised classifiers were developed to detect sentiments in Twitter messages [16], [17]. We differ from the above work as follows: 1) Our algorithm is supervised and adaptive. 2) We automatically combine all available sentiment detectors with additional linguistic features to detect the final polarity.

III. The Impact of Diversity on the Performance of Sentiment Detectors

An adaptive sentiment classifier must learn from the strengths and weaknesses of multiple sentiment detectors. To understand how such a classifier can be built, in this section, we present a case study that offers insights into the impact of textual diversity on the output of sentiment detectors. As explained in Sections I, II, sentiment detection in software engineering is challenging due to the *diversity* of domains (e.g., Jira issues vs. mobile-app reviews). In particular, we assess three publicly-available sentiment detection tools for software artifacts (Senti4SD [12], SentistrengthSE [11], SentiCR [13]) on five different sentiment benchmarks (see Section III-A). We discuss the case study design in Section III-B and present results in Section III-C.

A. Benchmarks

We analyze the following five 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 I provides descriptive summary statistics of the benchmarks. Overall, 43% of the units in the benchmarks are labeled as positive or negative and the rest (67%) are labeled as neutral (see last column in Table I), i.e., the benchmarks are imbalanced with regards to neutral vs. non-neutral classes.

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

2) **Jira: Ortu et al. [6].** The benchmark consists of 6,000 units (2,000 issue comments and 4,000 sentences 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 labelled with three polarities by Novieli et al. [15], which we obtained from the authors for this paper. In their translation, Novieli 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.

3) **StackOverflow: Lin et al. [14].** The benchmark contains 1,500 randomly-extracted sentences. Each sentence was manually assigned a sentiment score ranging from -2 (strong negative) to +2 (strong positive) by two coders. Each sentence was divided into a number of nodes (e.g., clauses), which were assigned a sentiment score. We use the final sentiment scores available in the replication package of [14] to assign a polarity: +1 positive, -1 negative, and 0 neutral.

4) **Apps Reviews: Lin et al. [14].** The benchmark contains 341 mobile-app reviews from Villarroel et al. [30], 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 [14].

5) **StackOverflow: Uddin et al. [3].** The benchmark consists of the 4,522 sentences in 71 randomly-sampled Stack Overflow threads. Each sentence 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 sentences from the threads, i.e., contextual information. We obtained this benchmark from the authors of Opiner [3], who used it to develop an online opinion summarization engine.

Besides the above benchmarks, the authors of SentiCR also shared their benchmark [13]. However, we do not use this benchmark because it only contains two types of labels for each unit: negative and non-negative.

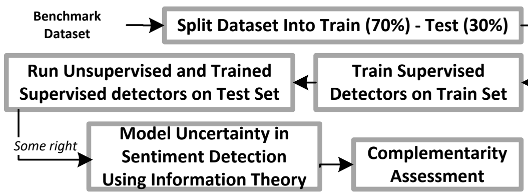
Benchmark Dataset
Split Dataset Into Train (70%) - Test (30%)
Train Supervised Detectors on Train Set
Run Unsupervised and Trained Supervised detectors on Test Set
Some right
Model Uncertainty in Sentiment Detection Using Information Theory
Complementarity Assessment

Fig. 2. The major steps of the case study

B. Study Methodology

We proceed in the following steps (Figure 2): 1) Senti4SD and SentiCR are supervised sentiment detectors, while SentistrengthSE is rule-based. We train the supervised classifiers on the benchmarks as follows: We split the benchmark into training (70%) and testing (30%) sets using the stratified sampling module `StratifiedShuffleSplit` [31]. 2) We retrain the two supervised classifiers on the training set by following the steps in the corresponding papers and Websites (accessed on Oct. 5, 2018). We train SentiCR to detect three polarity classes following Novieli et al. [15] by using numeric identifiers: positive (+1), negative (-1), and neutral (0). We assess the performance of each tool, including SentistrengthSE, on the testing set. 3) We collect all the units for which at least one tool is wrong and one tool is right. 4) For each collected unit, we compute *diversity metric* (discussed below). 5) We assess each tool based on the values of the diversity metrics.

If one tool is wrong but another is right, then there is some *uncertainty* in the detection process of the tools. To quantify uncertainty for a unit, we use *Information Theory* [32], which was previously used in a number of software-engineering research, e.g., to predict faults by quantifying the complexity of code changes [33] and to triage crashes in Mozilla [34]. In the following, we first briefly overview information theory and then explain our use to model and assess uncertainty in sentiment detection.

Information Theory. *Shannon Entropy* measures the information associated with a given input using Equation 1 [32]:

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

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

Modeling Uncertainty in Sentiment Detection. For each unit in our testing sets, we compute four types of *diversity* metrics based on Shannon Entropy.

1) **Polarity** $_{Entropy}$: We compute the *diversity* of polarity words in a given unit using Equation 1. Given a unit, we identify all the polarity lexicons in the unit and develop a data structure of the form $\{(w_1, f_1), ...(w_n, f_n)\}$, 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 diversity value, i.e., H_n.
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. With the sentence "The API is great, it's slow but it's great", the entropy value becomes 0.63 because "great" is repeated twice. The higher probability of the word "great" in the second sentence makes the sentence more positive, i.e., it reduces the amount of uncertainty of its polarity. Therefore, the higher the entropy value is for a sentence, the more difficult it may be to assign polarity (if we consider the occurrence of polarity in the sentence). The entropy measurement can also be useful when the sentiment detector does not have information of all the polarity words, e.g., when for an input "The tool is easy to use, but faulty and slow", the sentiment detector only knows that "great" is a polarity word, but not "faulty" or "slow". In such case, the increasing level of entropy would denote more uncertainty.
To detect the polarity words in a sentence, we use the following sentiment lexicons:
a) list of The sentiment lexicons and emoticons used in Sentistrength [26]. This list was used in the development of all of the three tools (Senti4SD, SentiCR, Sentistrength).
b) The list of sentiment lexicons used by Uddin et al. [3], which contains 750 software domain-specific sentiment lexicons from Stack Overflow and all sentiment words collected from three publicly-available benchmarks [35]–[37].
2) **Adjective** $_{Entropy}$: We compute the *diversity* of all adjectives in a given unit. We first identify all the adjectives and their frequency in a unit. We then use Equation 1 to compute the a diversity value based on adjectives for the unit. A polarity word may be missing in our sentiment lexicons, but may still be found in the adjectives.
3) **Verb** $_{Entropy}$: We compute the *diversity* of all verbs in a given unit using Equation 1 similarly as we did with adjectives and motivated by previous findings by Ko et al. [38], who observed that developers use verbs to describe software problems (e.g., "This tool does not work").
4) **All Words** $_{Entropy}$: We compute the *diversity* of all words in a given unit as "baseline", i.e., a truly random sentiment detector would be impacted by the increase of diversity in any words (e.g., "I developed this tool").

Impact Assessment of the Diversity Metrics. We assessed the impact of the diversity metrics on the misclassification of each sentiment tool by developing logistic regression models. 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 four diversity metrics above, i.e., entropy measurements using polarity, adjective, verb, and all words. To fit and interpret the models, we follow standard practices in the literature [39], [40]:

1) When fitting the models, we test for multicollinearity between the explanatory variables using the Variable Inflation

TABLE II
REGRESSION MODELS FOR THE SENTIMENT DETECTORS ON THE MISCLASSIFIED UNITS WHERE AT LEAST ONE DETECTOR WAS RIGHT

	Senti4SD Response: Misclassified = TRUE Pseudo R^2 = 3.06%			**SentiCR** Response: Misclassified = TRUE Pseudo R^2 = 3.86%			**SentistrengthSE** Response: Misclassified = TRUE 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*
All Words $_{Entropy}$	0.06	0.13	62.74***	0.11	0.13	77.09***	-0.31**	0.11	62.51***

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

Factor (VIF), and remove variables with VIF scores above the recommended maximum of 5 [41].

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 II).

3) We report the goodness of fit using the McFadden's pseudo R^2.

C. Results

Overall, 64.2% of the units were correctly classified by all tools. When at least one tool was wrong, one or two other tools were correct in 23.9% cases. Therefore, if we could develop a technique that could correctly predict the correct classifier for an input, the technique would be correct in 88.1% of units. In Table II, 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 variable $Polarity_{Entropy}$ is statistically significant for each tool, i.e., the misclassification of each tool increases with the increase in the diversity of polarity words. Therefore, each tool could benefit from how other tools determine the polarity of a sentence because each tool differs from the other in its assessment of polarity: Senti4SD is the most robust tool to detect polarity when the diversity of polarity increases, SentistrengthSE is the most vulnerable in those cases. Therefore, SentistrengthSE could benefit from the additional features used in the Senti4SD, such as the sentiment semantic features.

Unlike SentiCR and Senti4SD, SentistrengthSE is also more error-prone in its classifications when the diversity of verbs increases in the units because developers can express their opinions by merely using verbs, e.g., "The tool did not work" or "The tool crashed". Given that SentistrengthSE does not rely on context, it cannot learn from the training set where such instances may be present. This is evident in Table II, where the misclassification of only SentistrengthSE increases with the increase in diversity of verbs in the input texts.

Unlike SentistrengthSE and Senti4SD, SentiCR tool relies on a very small amount of sentiment lexicons. It relies more on the internal sentence structures (e.g., contractions and contexts). Therefore, SentiCR can get confused with the increase diversity of adjectives as well as sentiment words, for example, when the words were not seen in the training set (i.e., more uncertainty in the detection).

Summary: Senti4SD, SentiCR, and SentistrengthSE can complement each other at their processing of polarity lexicons. SentistrengthSE can benefit from the contextual information that the supervised classifiers gain during their training phase. The tools can benefit from the incorporation of additional sentiment lexicons to reduce uncertainty in sentiment detection.

IV. SENTISEAD: ADAPTIVE SENTIMENT DETECTOR

Based on insights gained from study in Section III, we developed an adaptive sentiment detection algorithm, Sentisead that can automatically learn from the strengths and weaknesses of multiple sentiment detection tools. In particular, the design of Sentisead leverages the following findings from Section III:

- Verb $_{Entropy}$. As we observed in Table II, the rule-based tool misclassifies more when expressed sentiments are contextual (e.g., using verbs as in 'the tool crashed'). In such cases the the incorporation of domain specific contexts as features into the adaptive engine will help it decide when and how the tools can be leveraged (e.g., prefer a supervised classifier to detect polarity).
- Polar $_{Entropy}$. The misclassification of the tools increase with the increase in the diversity in polarity. We observed that the polarity-based diversity can increase when the polarity words may not be common to each tool (e.g., semantic feature in Senti4SD vs only emoticons in SentiCR). In such cases, the adaptive classifier may correlate the underlying context of a unit (e.g., bag of words) with the tool which was more accurate in such context during the training phases. The polarity-based diversity also increases with the presence of domain-specific sentiment lexicons. In such cases, the tools may benefit from the inclusion of additional polarity information.

The Sentisead algorithm is modular, such that any number of sentiment detection classifiers can be included without any change in the design of the algorithm. We present the components of the classifier in Sections IV-A-Sections IV-B.

```
input : (1) Unit, e.g., Sentence $U$, (2) Sentisead
        Classifier $S$, (3) Combined Classifier $A$, and
        (4) Participating Classifiers in $C$ as
        $R = \{r_1, r_2, ..., r_n\}$ .
output: Sentiment $polarity$ of $U$
1  $P_C$ = getFeatureComplementary ($A$, $U$);
2  $P_D$ = getFeatureDSO ($U$);
3  $P_L$ = getFeatureBoW ($U$);
4  Features $F = \{P_C, P_D, P_L\}$;
5  foreach Classifier $r_i \in R$ do
6      $p_i$ = polarity of $S$ by $r_i$;
7      $F = F \cup p_i$;
8  $polarity$ = get polarity of $F$ by Sentisead classifier $S$;
9  procedure getFeatureComplementary($A$, $U$)
10     Participating $r$ = best classifier for $U$ by $A$;
11     return $polarity$ of $U$ by $r$;
12 procedure getFeatureDSO($U$)
13     return $polarity$ of $U$ using DSO algorithm;
14 procedure getFeatureBoW($U$)
15     Vectors $V = \emptyset$;
16     foreach word $w_i \in U$ do
17         if $w_i \notin$ stopwords then
               $V = V \cup$ Vectorizer($w_i$) ;
18     return $V$;
19 return $polarity$
```

Algorithm 1: Polarity detection using Sentisead

A. Polarity Detection in Sentisead Using Algorithm 1

Algorithm 1 presents the Sentisead algorithm. The input to the algorithm are:

1) A unit U, e.g., a sentence whose polarity we want to detect. 2) The trained Sentisead classifier S. 3) The participating classifiers R that we want to complement combine, e.g., SentistrengthSE, SentiCR, and Senti4SD. 4) The complementary classifier A that is trained based on the outputs of the participating classifiers.

The output is the polarity label of the input text unit U.

Sentisead uses three types of features to determine the polarity label of the input text U (L8):

1) The polarity label of U based on the combined classifier A (L1, 9-11): given as input U, the classifier C returns the best participating classifier (L10). We then apply the best classifier on U to determine its polarity (L11).
2) The vectorized bag of words of U (L3, 14-18). We produce a vector representation of the individual tokens of U by first removing the stopwords. Following SentiCR [13], we use a TF-IDF vectorizer. The bag of words help the classifier to learn about the contexts in a U other than the polarity labels (e.g., the usage of verbs to express opinions as we observed in Section III).
3) The polarity label of U based on the Domain Sentiment Orientation (DSO) algorithm [35] (L2, 12,13): The detailed steps of DSO is described in [35]. We are not aware of any publicly-available implementation of the algorithm. We took cues from two sources:
   - Blair-Goldensohn et al. [42] implemented it to detect sentiments in hotels and restaurants reviews.
   - Uddin et al. [3] used it to summarize online API reviews.

   We implemented DSO as follows:
   a) *Detect potential sentiment words.* We record the words in U that match the sentiment words and their part of speech in our sentiment database (all sentiment words used in Section III). The sentiment lexicons from SO Uddin et al. [3] contains parts of speech of each sentiment word. We give a score of +1 to a recorded word with positive polarity and a score of -1 to a word of negative polarity.
   b) *Handle negations.* We detect negations in U by following Ahmed et al. [13]. The process works as follows: i) Detect Parts of speech of tokens in U. ii) Detect phrases that contains negations in U by feeding a linguistic shallow parser the grammar that denotes candidate phrases with negations. Thus, "not good" has a sentiment value of -1 while the adjective "good" is +1.
   c) *Compute Sentiment Score and Label Polarity.* We take the sum of all of the sentiment values (i.e., positive and negative ones). If the score is greater than 0, then we assign a polarity label "p" (i.e., positive). If less than 0, then we assign a label "n" (negative). Else, we assign a polarity label "o" (i.e., neutral).

   We leverage polarity label from DSO into Sentisead based on the following observations in Section III:
   a) It can compensate for the use of very few sentiment lexicons in a sentiment classifier during the final decision making in the polarity, b) It can incorporate domain-specific sentiment words into the polarity labeling process.

B. Training of Combined Classifier Using Algorithm 2

We develop a combined classifier using the three tools to inform us of the best possible classifier for a given unit. The combined classifier is more appropriate than majority voter in the following scenarios:

1) When majority of the classifiers are wrong, i.e., two classifiers are wrong and one is right.
2) When none of the classifiers agree on the polarity, i.e., one labels the polarity as positive, another as negative, and the third as neutral.

The inputs to Algorithm 2 are:

1) A manually-annotated training data-set D with the polarity label of each unit. 2) The participating classifiers in R that we want to combine.

The output of Algorithm 2 is the trained combined classifier A that given as input a unit outputs the best possible sentiment detector from a pool a sentiment detectors.

First, for each unit, we collect the polarity label from each of the participating classifiers (L1-4). We then compute the performance of each participating classifier on the training data D. We compute F1-macro average based on the observation that F1-macro average favors classes with smaller representation, which is important because all of our benchmarks are

```
input : (1) Data set of manually assigned polarity
        labels $D = \{D_1, D_2, \ldots D_n\}$ where $D_i = \{U_i, S_i\}$
        where $U_i$ = Unit, $S_i$ = Polarity of $D_i$;
        (2) Participating Classifiers $R = \{r_1, r_2, ..., r_n\}$ .
output: Combined Classifier $A$
1  foreach Classifier $r_i \in R$ do
2      foreach Unit $D_i \in D$ do
3          $p_i$ = polarity of Unit $U_i$ in $D_i$ by $r_i$;
4          $D_i = D_i \cup p_i$;
5  F1-Macro Averages, $F1s = Map < String, Double >$;
6  foreach Classifier $r_i \in R$ do
7      f1 = F1-macro average of $r_i$ outputs $\in T_i$;
8      $F1s[r_i]$ = f1;
9  Training set $T = \emptyset$;
10 foreach Data unit $D_i \in D$ do
11     $B_i$ = getBestClassifier ($D_i$, $F1s$);
12     if $B$ is null then continue;
13     Training unit $T_i = \{U_i, B_i\}$;
14     $T = T \cup T_i$;
15 $A$ = Trained classifier on $T$;
16 procedure getBestClassifier($T_i$, $F1s$)
17     $S_i$ = expected polarity of unit $U_i$ in $T_i$;
18     Correct Classifiers $C = \emptyset$;
19     foreach Classifier $r_i \in R$ do
20         $G_i$ = polarity label of $U_i$ by $r_i$;
21         if $G_i = S_i$ then $C = C \cup r_i$;
22         $C = C \cup r_i$;
23     Best classifier $B$ = null;
24     if length (C) = 1 then  $B$ = only classifier in $C$ ;
25     else if length (C) > 1 then
26         $B$ = classifier with the best F1-score in $C$;
27     return $B$;
28 return $A$
```

Algorithm 2: Training of Combined Classifier

highly imbalanced (more than 50% units are neutral), but the primary focus of sentiment detection is to detect the positive and negative sentiments [12], [15].

We determine the best participating classifiers for each unit D as follows (L16-28). For a given unit, we compare the truth (i.e., the manual label) with the polarity labels of each of the participating classifiers. If only one classifier is right, then we assign it as the best performing classifier for the unit. If more than one classifier is right, then we assign the unit the classifier with the higher F1-macro average on the overall benchmark as the best performing classifier. Given two classifiers offering the right label for a given unit, the one most consistent across the benchmark is better. If no classifier is right for the given unit, then we put a *null* value as the best classifier for the unit. To train the combined classifier, we cannot use those units because we cannot determine a good classifier for them in the training data.

We train the combined classifier A by creating a new training set T as follows (L9-15). For each unit D_i in the input dataset D, we create a train unit T_i as $\{U_i, B_i\}$, where U_i in the input text in D_i and B_i is the ID of the best classifier for U_i. We discard the units for which we could not determine the best classifier (i.e., *null* above). We train a supervised classifier as the combined classifier A.

C. Training of Sentisead

The design of Sentisead (i.e., Algorithm 1) allows the use of any supervised classifier. We use the Gradient Boosting Tree (GBT) algorithm to train the Sentisead classifier because it is an Ensemble algorithm that allows the usage of multiple predictors to *boost* the final output. This algorithm employs logic where the predictors are applied sequentially so that one classifier can learn from the mistake of the other classifier. That means with multiple weak classifiers, we can expect to achieve a better performance, because the predictors will complement each other. To implement our Sentisead classifier, we modified the code of SentiCR, which also uses a GBT classifier. This decision was based on the following observations: 1) The GBT configuration in SentiCR was found to be the best performer in the original SentiCR implementation [13]. Therefore, we could reuse similar setting. 2) We can use the bag of words features from SentiCR into Sentisead algorithm.

V. Evaluation

Our evaluation focuses on the effectiveness of our Sentisead algorithm. While our algorithm can be applied to combine any sentiment detection tools, in this paper, we compare Sentisead against the three available sentiment detection tools for software artifacts (Senti4SD [12], SentiCR [13], and SentistrengthSE [11]). We answer two research questions:

1) Does Sentisead outperform Senti4sD, SentiCR and SentistrengthSE for software artifacts?
2) How effectively does Sentisead combine Senti4sD, SentiCR and SentistrengthSE for software artifacts?

An effective combination of the baselines (Senti4SD, SentiCR, SentistrengthSE) in Sentisead should perform better than the individual baselines (RQ1). The design of Sentisead should allow the optimal combination of the baselines, i.e., to detect polarity, the Sentisead classifier should be in agreement more with the correct baseline(s) than with the wrong baseline(s) (RQ2). We answer the two research questions using the five benchmarks described in Section III-A.

RQ_1 ***Does Sentisead outperform Senti4sD, SentiCR and SentistrengthSE for software artifacts?***

• ***Approach:*** We compare the performance of Sentisead against the three baselines. For each benchmark, we use 10-fold cross validation as follows:

1) We divide the dataset into 10 folds using a stratified sampling, i.e., each fold has equal proportion of each polarity.
2) We train each supervised baseline (i.e., Senti4SD and SentiCR) on nine folds. We test the supervised as well as unsupervised (i.e., SentistrengthSE) baselines on the other fold. We do this for each fold, e.g., use fold 0 as the test fold for iteration 0, but use it as a train fold for rest of the

TABLE III
CONFUSION MATRIX FOR THREE CLASS SENTIMENT CLASSIFICATION PERFORMANCE ANALYSIS

		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

nine iterations. Therefore, at the end of the iterations, we have 10 trained models for each of Senti4SD and SentiCR.

3) We assign each unit in the test folds the best baseline using the procedure `getBestClassifier` from Algorithm 2.
4) We train and test the combined classifier (i.e., Algorithm 2 using the same 10-folds. That means, we use nine folds to train the combined classifier. We then use the trained combined classifier to assign the best baseline for each unit in the test fold. Therefore, each unit in the test fold is annotated with the ID of the best baseline and the polarity of the unit based on the polarity output of that baseline.
5) We train Sentisead on the 10 annotated folds similarly.
6) We compute the performance on the test folds.

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 To assess performance, we use the confusion matrix in Table III, which is consistent with the state of the art [12], [15], [43]. We report performance for each polarity as well as the overall accuracy using both micro and macro averages.

Macro average is useful to emphasize the performance 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 the state of the art [15], [44].

• ***Results:*** In Table IV, we report the macro and micro averages of Sentisead and the three baselines using precision, recall, and F1-score for all the benchmarks and for three polarities. Sentisead outperforms the baselines in all categories.

Among the polarities, Sentisead performs better than others for negative sentiments, followed by positive ones. For both positive and negative classes, Sentisead offers similar or higher precision and recall values than the baselines. For the neutral classes, the performance of Sentisead is the same as the best performing baseline (SentiCR).

In Table V, we break down the performance of the tools by benchmarks. For each tool in Table V, we show the macro and micro averages and F1-scores for the three polarities by benchmarks. Sentisead is the best in four out of the five benchmarks based on macro averages. It is the second for the other benchmark (Jira [6]). Among the baselines, Senti4SD is the second best for StackOverflow [12], which was originally used to develop Senti4SD. For Sentisead, the biggest performance gains compared to the baselines are for benchmarks StackOverflow [14] and App [14]).

These two benchmarks were previously used by Lin et al. [14]. Applying SentistrengthSE on Stack Overflow [14], they observed F1-scores of 0.26 for positive and 0.27 for negative sentiments. The best performing classifiers were two cross domain classifiers: Stanford sentiment detector for positive (F1 = 0.28) and Sentistrength for negative (F1 = 0.41). In contrast, for the same benchmark, we observed a F1-score of 0.44 for positive and 0.50 for negative sentiments using Sentisead. Similarly, for App [14], they observed that the baseline SentistrengthSE was outperformed by Sentistrength (for positive with a F1-score = 0.8) and Stanford sentiment detector (for negative sentiments with a F1-score = 0.71). In contrast, using Sentisead and Senti4SD, we observed a F1-score of 0.86 for positive and of 0.80 for negative sentiments.

Sentisead is second best to SentiCR for Jira [6] for two reasons: 1) Due to its design to complement the three baselines, Sentisead depends on the complementary classifier to find the best baseline for a given unit. For some misclassified units, the complementary classifier picked Senti4SD, which performed worse than the other other tools. 2) The units are posts, i.e., larger amount of texts than the other benchmarks. SentiCR depends more on text contents than the other baselines and was designed to handle such comments (see Section III).

In Table VI, we show how the two features (Combined Classifier and the DSO algorithm) perform on the five datasets. In Sentisead, the combined classifier is an adaptation of the Random Forest classifier in SentiCR. We reuse the SentiCR architecture to train the combined classifier. We pick the RF-based complementary classifier after comparing its performance over two other trained models (see Table VI): GBT and Adaboost. Between the two features (Combined and DSO), the combined classifier offers better performance (macro average = 0.74 for combined classifier and 0.59 for DSO). Therefore, the major performance gain in Sentisead comes from the combined classifier, i.e., an effective combination of the baselines.

Majority Voting. The combined classifier outperforms a majority voting-based classifier that labels a unit as the polarity agreed by the most baselines. We observed a macro precision of 0.60 and a recall of 0.53 using a majority voting-based classifier (F1-score = 0.56 compared to 0.74 by the combined classifier). The low performance is due to the following reasons: 1) The majority were wrong, 2) There was no majority, and 3) All the tools were wrong.

Summary: Across the datasets, Sentisead outperformed all the three baselines. Sentisead was the best performer in four out of the five benchmarks and second best in the other. Out of the two classifiers used as features in Sentisead algorithm, the combined classifier performed better, i.e., the performance gain in Sentisead was effective to complement the baselines.

TABLE IV
OVERALL PERFORMANCE OF THE CLASSIFIERS ON THE FIVE BENCHMARK DATASETS

	Macro			Micro			Positive			Negative			Neutral		
	F1	P	R	F1	P	R	F1	P	R	F1	P	R	F1	P	R
Sentisead	**0.75**	0.76	0.73	**0.78**	0.78	0.78	**0.74**	0.79	0.69	**0.67**	0.70	0.63	**0.83**	0.80	0.87
Baseline - **Senti4SD [12]**	0.73	0.75	0.71	0.76	0.76	0.76	0.72	0.76	0.67	0.63	0.69	0.58	0.82	0.78	0.87
Baseline - **SentiCR [13]**	0.73	0.77	0.69	0.77	0.77	0.77	0.72	0.79	0.65	0.62	0.76	0.52	0.83	0.77	0.90
Baseline - **SentistrengthSE [11]**	0.71	0.73	0.69	0.75	0.75	0.75	0.70	0.74	0.67	0.60	0.69	0.54	0.80	0.76	0.85

TABLE V
THE F1-SCORE OF THE CLASSIFIERS ON EACH BENCHMARK DATASET

		Macro	Micro	+VE	-VE	Neutral
SO	Sentisead	**0.85**	**0.85**	**0.92**	**0.81**	**0.82**
Calefato	Senti4SD	0.84	0.84	0.91	0.80	0.80
et al.	SentiCR	0.82	0.83	0.89	0.76	0.81
[12]	SentistgthSE	0.79	0.79	0.86	0.76	0.75
SO	Sentisead	**0.50**	0.61	**0.32**	**0.38**	0.74
Uddin	Senti4SD	**0.50**	**0.62**	0.36	0.34	**0.75**
et al.	SentiCR	0.45	0.59	0.23	0.24	0.74
[3]	SentistgthSE	0.46	0.60	0.31	0.23	0.74
SO	Sentisead	**0.62**	**0.83**	**0.44**	**0.50**	**0.90**
Lin	Senti4SD	0.58	0.82	0.32	0.45	**0.90**
et al.	SentiCR	0.59	0.82	0.39	0.45	**0.90**
[14]	SentistgthSE	0.49	0.78	0.26	0.26	0.87
App	Sentisead	**0.59**	0.79	**0.86**	**0.80**	0.10
Lin	Senti4SD	0.55	**0.80**	0.86	**0.80**	–
et al.	SentiCR	0.56	0.78	0.85	0.77	0.05
[14]	SentistgthSE	0.55	0.59	0.78	0.45	**0.17**
Jira	Sentisead	0.79	0.84	**0.78**	0.71	0.89
Ortu	Senti4SD	0.73	0.81	0.73	0.60	0.86
et al.	SentiCR	**0.81**	**0.85**	**0.78**	**0.73**	**0.90**
[6]	SentistgthSE	0.79	0.83	**0.78**	0.71	0.87

RQ_2 How does Sentisead combine Senti4sD, SentiCR and SentistrengthSE for software artifacts?

• ***Approach:*** We measure the complementarity of Sentisead over the baselines by analyzing the agreements of the tools (Sentisead and baselines) with one another and the manual labels (i.e., expected polarities). We report the agreements using weighted Cohen kappa (κ) [45] and percent agreement. In the context of sentiments, in which positivity vs. negativity is more important than neutral vs. non-neutral units, a disagreement is severe if the polarity changes from positive to negative or vice-versa. A disagreement is mild if the polarity changes from neutral to non-neutral. Following the state of the art [15], [46], we thus distinguish between *mild* and *severe disagreements* using weights 1 and 2, respectively, in the weighted Cohen κ (Table VII). We further assess where Sentisead offers improvements over the baselines by correcting their misclassifications, when all them were wrong.

• ***Results:*** In Table VIII, we report the agreements between the tools and the manual labels. Sentisead shows almost perfect agreement ($\kappa > 0.81$) with both the supervised baselines (Senti4SD and SentiCR) and substantial agreement ($\kappa = 0.656$) with the rule-based baseline (SentistrengthSE). The supervised baselines better performed in the benchmarks and, thus, Sentisead preferred their outputs over SentistrengthSE.

Sentisead complemented the baselines by correcting both severe and mild disagreements in the baselines. However, Sentisead was more successful for mild disagreement. Severe disagreement may arise in the presence of domain-specific contexts, which may be unknown to the tools.

In Table IX, we show percentages of units corrected by Sentisead that were misclassified by all the baselines. Sentisead corrected 67 of such 2,086 units (3.2%). In Table IX we also show the fraction of those units that the DSO algorithm could have corrected if it was applied stand-alone. In Sentisead, the DSO algorithm leverages the sentiment lexicons that were not present in the Sentistrength ones, such as domain specific sentiment words. As we observed in Section III, the misclassifications in the baseline tools increases with the diversity of polarity in a unit. As shown in Table IX, the DSO algorithm could have corrected 29% of the misclassifications. It would have been more useful for those units compared to the Sentisead classifier. As we see in Table VI, out of the two features (combined and DSO), the combined classifier offered much better performance for all the benchmarks.

Summary: Sentisead complements the baselines by showing perfect agreement with the supervised classifiers on the correct classifications. For the units where all the baselines are wrong, the DSO algorithm in Sentisead offers better performance than Sentisead when DSO is applied as a stand alone sentiment detector.

VI. DISCUSSIONS

We discuss major themes emerged from the study below.

Supervised Classifier in Sentisead. As we noted in Section IV-A, we picked a GBT-classifier from SentiCR codebase to train Sentisead after comparing its performance against two other ensemble classifiers included in the SentiCR codebase (Random Forest and Adaboost). In Table X, we show the performance of the three classifiers utilizing the Sentisead algorithm on the five benchmarks. The GBT-based classifier outperformed the other two ensemble classifiers. The GBT-based classifier was also the best performer in the original implementation of SentiCR that the authors developed to detect sentiments in code reviews. The better performance of GBT is not random, because GBT leverages more hyper parameters than random forest during training. Indeed, boosting-based algorithms (e.g., GBT) are found to be better performers than

TABLE VI
THE PERFORMANCE OF INDIVIDUAL FEATURES IN SENTISEAD (GBT = GREADIENT BOOSTING TREE, RF = RANDOM FOREST, ADB = ADABOOST)

	Macro			**Micro**			**Positive**			**Negative**			**Neutral**		
	F1	**P**	**R**	**F1**	**P**	**R**	**F1**	**P**	**R**	**F1**	**P**	**R**	**F1**	**P**	**R**
Combined Classifier - RF	**0.74**	0.76	0.72	**0.78**	0.78	0.78	**0.73**	0.77	0.69	**0.65**	0.72	0.60	**0.83**	0.79	0.87
Feature Linguistic - DSO	0.59	0.59	0.59	0.63	0.63	0.63	0.54	0.50	0.59	0.53	0.54	0.52	0.70	0.72	0.68
Baseline Combined Classifier - GBT	**0.74**	0.76	0.72	**0.78**	0.78	0.78	0.72	0.77	0.68	**0.65**	0.73	0.60	**0.83**	0.79	0.88
Baseline Combined Classifier - ADB	0.74	0.76	0.72	0.78	0.78	0.78	0.72	0.77	0.68	0.65	0.72	0.59	0.83	0.79	0.88

TABLE VII
WEIGHTING SCHEME TO COMPUTE WEIGHTED COHEN κ [45]

Class	Negative	Neutral	Positive
Negative	0	1	2
Neutral	1	0	1
Positive	2	1	0

TABLE VIII
AGREEMENT BETWEEN SENTIMENT CLASSIFIERS AND MANUAL LABELS.

Tool	Metric	Manual	Senti4SD	SentiSE	SentiCR
Sentisead	Weighted κ	0.663	0.814	0.656	0.827
	Perfect Agg	78.9%	89.2%	79.6%	90.5%
	Disagree Severe	1.9%	1.2%	1.4%	1.3%
	Disagree Mild	19.1%	9.6%	19.0%	8.1%
Senti4SD	Weighted κ	0.632	–	0.656	0.698
	Perfect Agg	77.4%	–	79.6%	83.4%
	Disagree Severe	2.3%	–	1.4%	2.1%
	Disagree Mild	20.2%	–	19.0%	14.6%
SentiSE	Weighted κ	0.599			0.670
	Perfect Agg	74.8%	–	–	81.5%
	Disagree Severe	1.6%	–	–	1.6%
	Disagree Mild	23.5%	–	–	16.8%
SentiCR	Weighted κ	0.635	–	–	–
	Perfect Agg	78.0%	–	–	–
	Disagree Severe	1.9%	–	–	–
	Disagree Mild	20.1%	–	–	–

TABLE IX
SENTISEAD IMPROVEMENTS WHERE ALL BASELINES WERE WRONG

	All Baselines	**Sentisead**	**Right**	**Feat DSO**	**Right**
	Wrong	**Total**	**Ratio**	**Total**	**Ratio**
Overall	2086	67	3.2%	604	29.0%
SO Calefato [12]	313	6	1.9%	69	22.0%
SO Uddin [3]	1146	42	3.7%	276	24.1%
SO Lin [14]	158	13	8.2%	53	33.5%
App Lin [14]	27	0	0.0%	7	25.9%
Jira Ortu [6]	442	6	1.4%	199	45.0%

TABLE X
PERFORMANCE OF ENSEMBLE CLASSIFIERS AS SENTISEAD

	Macro	**Micro**	**Positive**	**Negative**	**Neutral**
Sentisead - GBT	**0.75**	**0.78**	**0.74**	**0.67**	**0.83**
Sentisead - RF	0.74	0.77	0.73	0.66	0.82
Sentisead - ADB	0.72	0.76	0.72	0.64	0.81

the bagging-based ensemble techniques (e.g., random forest) in a high-dimensional space (e.g., textual features) [47].

Features in Sentisead. As we observed in Table IX the performance of Sentisead could have been even better, if it used the polarity labels of the DSO algorithm (in Algorithm IV-A) more for the units where all baselines were wrong. For example, the following sentence was labeled as positive by DSO: *"I use straight and simple code."* DSO considers the two adjectives (straight and simple) which are included in the Uddin et al. sentiment lexicons [3], but not in the Sentinstrength sentiment lexicons. Given that the combined classifier is more accurate than the DSO algorithm in the datasets, and given that the words "simple" and "straight" are not found too often in other units of the benchmarks, the Sentisead classifier opted to pick the label of combined classifier. Therefore, more investigation can be conducted to determine the optimal combination of features in Sentisead.

Threats to Validity. *Internal validity* warrants the presence of systematic error (e.g., bias) in the study. We mitigated the bias using statistical significance analyses. *External validity* concerns the *generalizability* of the results presented in this paper, which are based on five publicly-available sentiment benchmarks. Together, these benchmarks consist of around 16K units (sentences, posts, etc.), each manually annotated for polarity by multiple coders. Therefore, our evaluation corpus is bigger compared to previous research work on sentiment detection for software engineering. Yet, we do not claim that the same results could be expected in general. *Reliability* concerns the possibility of replicating this evaluation. We provide the necessary details in an online appendix [48].

VII. CONCLUSIONS AND FUTURE WORK

Sentiment detection in software engineering can be challenging due to the diverse nature of software textual contents. To understand whether the performance of sentiment detection could be improved by combining the tools, we conducted a case study on the complementarity of three sentiment detectors in five different benchmarks. We observed that the tools exhibit strengths and weaknesses, that in most cases are complementary. Based on the insights gained from the study, we developed an adaptive sentiment detection algorithm, Sentisead. Using Sentisead, we can combine multiple sentiment detection tools to obtain accuracy better than the stand-alone detectors. We evaluated Sentisead using five different benchmarks available for software engineering. Sentisead outperformed the baselines in the data-sets with an overall increase in F1-scores by 2.8 (best baseline)- 11.7% (least performing baseline). Our future work focuses on the investigation of additional features and sentiment detection tools into the Sentisead algorithm.

References

[1] B. Liu, *Sentiment Analysis and Opinion Mining*, 1st ed. Morgan & Claypool Publishers, May 2012.

[2] B. Pang, L. Lee, and S. Vaithyanathan, "Thumbs up? sentiment classification using machine learning techniques," in *Conference on Empirical Methods in Natural Language Processing*, 2002, pp. 79–86.

[3] G. Uddin and F. Khomh, "Automatic summarization of API reviews," in *Proc. 32nd IEEE/ACM International Conference on Automated Software Engineering*, 2017, pp. 159–170.

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

[5] M. Mäntylä, B. Adams, G. Destefanis, D. Graziotin, and M. Ortu, "Mining valence, arousal, and dominance – possibilities for detecting burnout and productivity?" in *Proceedings of the 13th Working Conference on Mining Software Repositories*, 2016, pp. 247–258.

[6] M. Ortu, B. Adams, G. Destefanis, P. Tourani, M. Marchesi, and R. Tonelli, "Are bullies more productive? empirical study of affectiveness vs. issue fixing time," in *Proceedings of the 12th Working Conference on Mining Software Repositories*, 2015, pp. 303–313.

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

[8] G. Uddin and F. Khomh, "Opiner: A search and summarization engine for API reviews," in *Proc. 32nd IEEE/ACM International Conference on Automated Software Engineering*, 2017, pp. 978–983.

[9] N. Novielli, F. Calefato, and F. Lanubile, "The challenges of sentiment detection in the social programmer ecosystem," in *Proceedings of the 7th International Workshop on Social Software Engineering*, 2015, pp. 33–40.

[10] R. Socher, A. Perelygin, J. Wu, C. Manning, A. Ng, and J. Chuang, "Recursive models for semantic compositionality over a sentiment treebank," in *Proc. Conference on Empirical Methods in Natural Language Processing (EMNLP)*, 2013, p. 12.

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

[12] F. Calefato, F. Lanubile, F. Maiorano, and N. Novielli, "Sentiment polarity detection for software development," *Journal Empirical Software Engineering*, pp. 2543–2584, 2017.

[13] T. Ahmed, A. Bosu, A. Iqbal, and S. Rahimi, "Senticr: A customized sentiment analysis tool for code review interactions," in *Proceedings of the 32nd International Conference on Automated Software Engineering*, 2017, pp. 106–111.

[14] B. Lin, F. Zampetti, G. Bavota, M. D. Penta, M. Lanza, and R. Oliveto, "Sentiment analysis for software engineering: how far can we go?" in *Proc. 40th International Conference on Software Engineering*, 2018, pp. 94–104.

[15] N. Novielli, D. Girardi, and F. Lanubile, "A benchmark study on sentiment analysis for software engineering research," in *Proceedings of the 15th International Conference on Mining Software Repositories*, 2018, p. 12.

[16] P. P. B. Filho, L. Avanco, T. A. S. Pardo, and M. G. V. Nunes, "Nilc_usp: An improved hybrid system for sentiment analysis in twitter messages," in *Proceedings of the 8th International Workshop on Semantic Evaluation*, 2014, pp. 428–432.

[17] P. Goncalves, M. Araujo, F. Benevenuto, and M. Cha, "Comparing and combining sentiment analysis methods," in *Proceedings of the first ACM conference on Online social networks*, 2013, pp. 27–38.

[18] B. Ghotra, S. McIntosh, and A. E. Hassan, "Revisiting the impact of classification techniques on the performance of defect prediction models," in *Proceedings of the 37th International Conference on Software Engineering*, 2015, pp. 789–800.

[19] D. D. Nucci, F. Palomba, R. Oliveto, and A. D. Lucia, "Dynamic selection of classifiers in bug prediction: An adaptive method," *IEEE Transactions on Emerging Topics in Computational Intelligence*, vol. 1, no. 3, pp. 202–212, 2017.

[20] M. Ortu, B. Adams, G. Destefanis, P. Tourani, M. Marchesi, and R. Tonelli, "Are bullies more productive? empirical study of affectiveness vs. issue fixing time," in *Proceedings of the 12th Working Conference on Mining Software Repositories*, 2015.

[21] A. B. Warriner, V. Kuperman, and M. Brysbaert, "Norms of valence, arousal, and dominance for 13,915 english lemmas," *Behavior Research Methods*, vol. 45, no. 4, pp. 1191–1207, 2013.

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

[23] E. Guzman and B. Bruegge, "Towards emotional awareness in software development teams," in *Proceedings of the 7th Joint Meeting on Foundations of Software Engineering*, 2013, pp. 671–674.

[24] D. M. Blei, A. Y. Ng, and M. I. Jordan, "Latent dirichlet allocation," *Journal of Machine Learning Research*, vol. 3, no. 4-5, pp. 993–1022, 2003.

[25] R. Jongeling, S. Datta, and A. Serebrenik, "Choosing your weapons: On sentiment analysis tools for software engineering research," in *Proceedings of the 31st International Confernece on Software Maintenance and Evolution*, 2015.

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

[27] IBM, *Alchemy sentiment detection*, http://www.alchemyapi.com/products/alchemylanguage/sentiment-analysis, 2016.

[28] NLTK, *Sentiment Analysis*, http://www.nltk.org/howto/sentiment.html, 2016.

[29] N. V. Chawla, K. W. Bowyer, L. O. Hall, and W. P. Kegelmeyer, "Smote: synthetic minority over-sampling technique," *Journal of Artificial Intelligence Research*, vol. 16, no. 1, pp. 321–357, 2002.

[30] L. Villarroel, G. Bavota, B. Russo, R. Oliveto, and M. D. Penta, "Release planning of mobile apps based on user reviews," in *Proceedings of the 38th International Conference on Software Engineering*, 2016, pp. 14–24.

[31] Scikit-learn, *Machine Learning in Python*, http://scikit-learn.org/stable/.

[32] C. E. Shannon, "A mathematical theory of communication," *The Bell System Technical Journal*, vol. 27, no. 3, pp. 379–423, 1948.

[33] A. E. Hassan, "Predicting faults using the complexity of code changes," in *Proc. 31st International Conference on Software Engineering*, 2009, pp. 78–89.

[34] F. Khomh, B. Chan, Y. Zou, and A. E. Hassan, "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*, 2011, pp. 261–270.

[35] M. Hu and B. Liu, "Mining and summarizing customer reviews," in *ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, 2004, pp. 168–177.

[36] T. Wilson, J. Wiebe, and P. Hoffmann, "Recognizing contextual polarity in phrase-level sentiment analysis," in *In Proceedings of the conference on Human Language Technology and Empirical Methods in Natural Language Processing*, 2005, pp. 347–354.

[37] A. Nielsen, "A new ANEW: Evaluation of a word list for sentiment analysis in microblogs," in *In the 8th Extended Semantic Web Conference*, 2011, pp. 93–98.

[38] A. J. Ko, B. A. Myers, and D. H. Chau, "A linguistic analysis of how people describe software problems," in *2005 IEEE Symposium on Visual Languages and Human-Centric Computing*, 2005, pp. 127–134.

[39] P. Thongtanunam, S. Mcintosh, A. E. Hassan, and H. Iida, "Review participation in modern code review: An empirical study of the android, qt, and openstack projects," *Journal of Empirical Software Engineering*, vol. 22, no. 2, pp. 768–817, 2017.

[40] M. Valiev, B. Vasilescu, and J. Herbsleb, "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*, 2018, pp. 644–655.

[41] J. Cohen, S. G. West, L. Aiken, and P. Cohen, *Applied Multiple Regression/Correlation Analysis for the Behavioral Sciences*, 3rd ed. Lawrence Erlbaum Associates, 2002.

[42] S. Blair-Goldensohn, K. Hannan, R. McDonald, T. Neylon, G. A. Reis, and J. Reyner, "Building a sentiment summarizer for local search reviews," in *WWW Workshop on NLP in the Information Explosion Era*, 2008, p. 10.

[43] F. Sebastiani, "Machine learning in automated text categorization," *Journal of ACM Computing Surveys*, vol. 34, no. 1, pp. 1–47, 2002.

[44] C. D. Manning, P. Raghavan, and H. Schütze, *An Introduction to Information Retrieval*. Cambridge Uni Press, 2009.

[45] J. Cohen, "Weighted kappa: Nominal scale agreement provision for scaled disagreement or partial credit," *Psychological Bulletin*, vol. 70, no. 4, pp. 213–220, 1968.

[46] R. Jongeling, P. Sarkar, S. Datta, and A. Serebrenik, "On negative results when using sentiment analysis tools for software engineering research," *Journal Empirical Software Engineering*, vol. 22, no. 5, pp. 2543–2584, 2017.

[47] R. Caruana, N. Karampatziakis, and A. Yessenalina, "An empirical evaluation of supervised learning in high dimensions," in *Proceedings of the 25th international conference on Machine learning*, 2008, pp. 96–103.

[48] *Adaptive Sentiment Detection for Software Artifacts (Online Appendix)*, https://github.com/anonsubmissions2/msr2019, 20 Jan 2019 (last accessed).